



Pázmány Péter Catholic University
Faculty of Information Technology and Bionics

Basics of Mobile Application Development

Introduction, Mobile platforms

Kálmán Tornai

Room #232

tornai.kalman@itk.ppke.hu

Foreword – Examination and evaluation system

- It is mandatory to participate in the lessons
 - The maximum number of allowed absence is 3
- On each lesson a short test-paper
 - Will be rated between 0 and 10.
 - Missing one is rated as 0.
- You must hand in your homework solutions. Homework are given occasionally; the expected number of homework is 8.
 - You must submit the homework in one week after it is handed out
 - The solutions are rated, and the sum of the points is calculated at the end of the semester.
 - Missing homework is 0 points
 - Solution without issues is 25 points

Foreword – Cont'd

- You must take a final exam, at the end of the semester.
 - 300 points
 - One substitution is permitted.
- Final grade is calculated from the sum of test points, homework points and final exam points:
 - [0% 50%): fail
 - [50% 60%): pass
 - [60% 70%): satisfactory
 - [70% 80%): good
 - [80% 100%]: excellent

Important task

- SVN
 - You must fill the following form
 - <https://forms.gle/SUxPtKLa3fdZw7FQA>
 - It is required in order to create the SVN repository directories, which are going to be used to submit homework!
 - Deadline:
 - 13th September, Friday 11.59 p.m.
 - Repositories will be created on next week
 - You will be notified
- Slack
 - Announcements, information, questions and answers:
 - https://join.slack.com/t/ppkeitk-ajma/shared_invite/enQtNzM0MzMzMjYyMjk3Mzc5LTljNmMxOTYzM2EzZTI3NTc0Nzg3ZTAwYTg1YTAYOTE0ZDNINzBiMzVhY2VkMDkxMTNmMmZiYTdmMmVkyTY4NGU

Foreword – This course in the curriculum

C++ and Java

Basics of Mobile Application Development



Android Application Development



iOS Application Development

Foreword – Schedule of the semester

- Introduction
 - Technologies and capabilities of the mobile platforms.
 - Introduction to Android and iOS platform.
- Hardware capabilities of mobile platforms
 - GPS, Bluetooth, NFC, sensors, ...
- Review of basic programming techniques and definitions
 - C++, Java, OOP, C++ and Java
- Programming patterns and methods (MVC, MVVM, Delegate, Adapter, Target-Action, TDD)
- SWIFT I & II
- Objective-C
- Kotlin I & II
- Android basics
- iOS basics
- Persistent data storage on iOS and Android, Databases
- Final Exam



What is a mobile application?

What is this all about?

Mobile applications

- What makes an application mobile?
- Further questions:
 - What is a web-based application?
 - What is a native application?
 - What is the better choice for our project?
 - Different aspects ...
 - What are the differences?

Functionality and number of functions

Mobile
webpage

Web
application

Hybrid
application

Cross platform
application

Native
application

Interoperability and cross-platform usability

Advantages and disadvantages

- HTML5

- Product available instantly on multiple platforms
- Do not have to be installed, easy to use
- Web development knowledge can be enough
- Updates can be spread better
- Lower costs on development side
- Faster development and deployment
- Special expertise on multiple platforms not required

- Native

- Device-specific capabilities can be utilized
 - Sensors
 - Graphical processor
 - Optimized code
- Higher performance
- Higher security level
- Better user experience can be achieved
- Lower costs on server/infrastructure side
- Built-in payment options

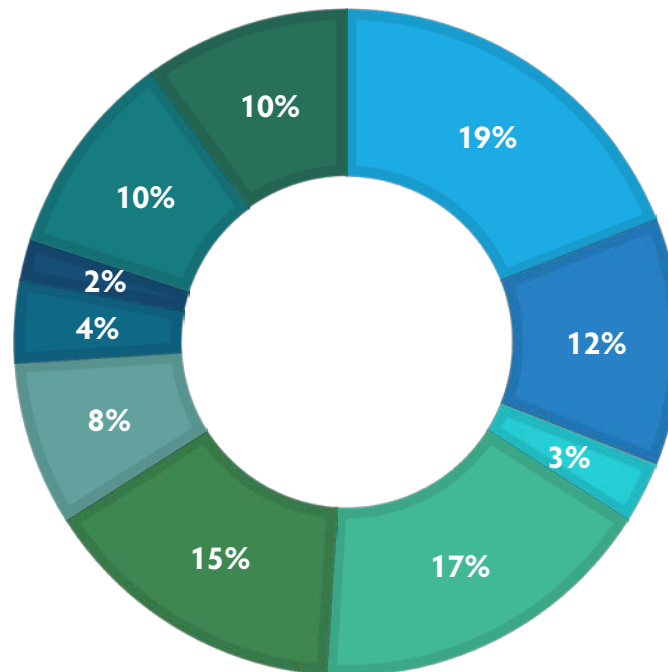
Hybrid applications

What to use?

- Depends on the nature of the application
 - Can you imagine a 3D game using HTML5 techniques?
 - Is it really required to implement a tax calculator as a native application?
- Viewpoints
 - Is 3D graphics required?
 - Is camera or sensors of the device required?
 - Location requirement does not imply native applications
 - Is there any internet connection at the location of usage?
 - Computational complexity
 - Wolfram Alpha vs Wolfram Mathematica
 - How many social media API is planned to be used?
- This is a design issue, there may not be a perfect solution
- This course concerns only with native applications

Where are we going?

DETAILS





Platforms

History and overview

IBM Simon – 1992



Microsoft Pocket PC – 2002



Apple Newton – 1993



Google Pixel 3 XL – 2018



Apple iPhone Xr – 2018



In the beginnings ...

- 1987/1993 Apple Newton
 - First mobile device, with applications – No mobile connectivity, thus it cannot be considered as a phone
- 1992 – IBM Simon
 - Fax, e-mail, calendar, calculator, clock, notepad, touch screen
- 1996 – Nokia 9000
 - Nokia cell phone + HP PDA
- 1999 – pdQ Smartphone
 - Palm PDA + Internet connection
- 1999 – NTT Docomo
- 1999 – Blackberry 850
- 2000 – Ericsson R380 Smartphone
 - First device which is called „smartphone“
- 2001 – Kyocera 6035
 - Palm PDA + Mobile phone (Verizon)
- 2002 – Microsoft PocketPC (later Windows Mobile)
 - Continuing the Windows CE platform

... the boom ...

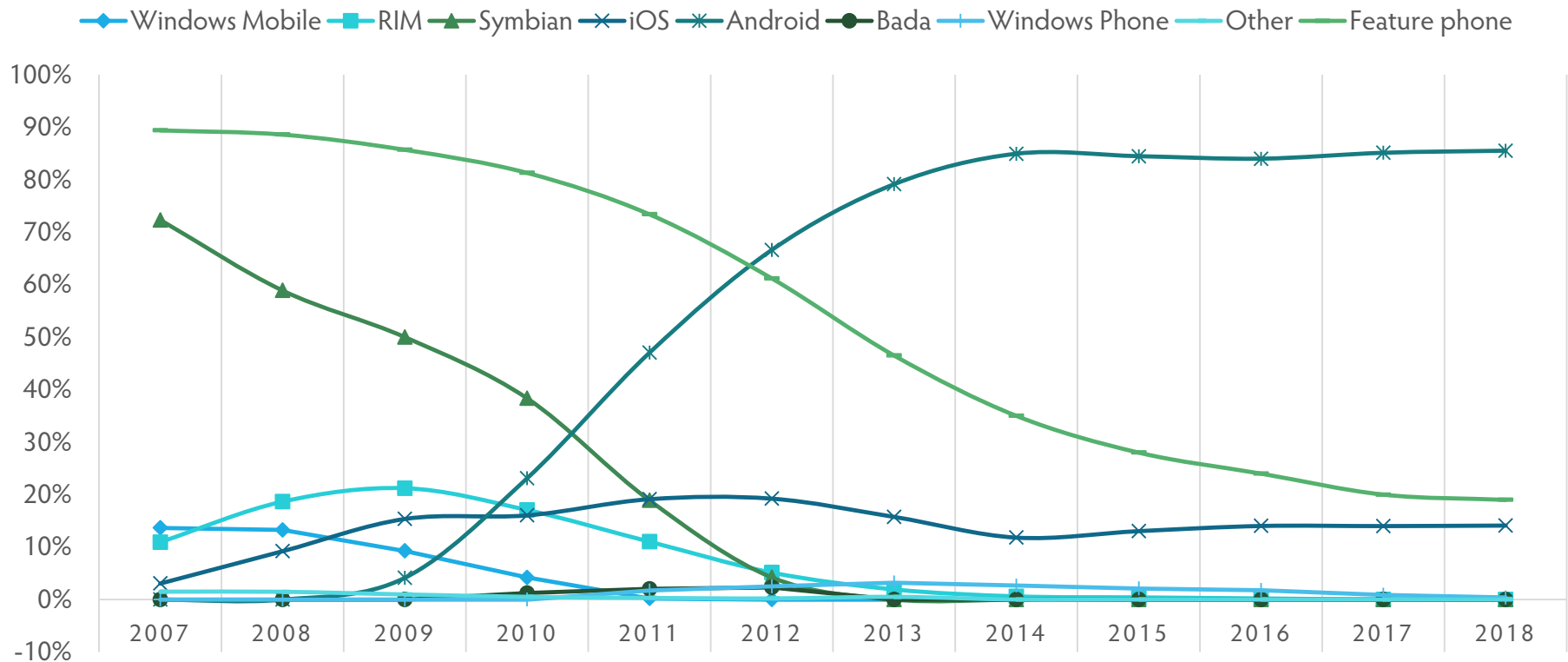
- 2007 – Apple iPhone
 - First multitouch phone (no stylus, keyboard, etc.)
- 2008 – Google Android
 - HTC Dream
- 2008 – Samsung Bada
 - Samsung Wave S8500
 - Migrated into the Tizen platform
- 2010 – Windows Phone
- 2011 – Palm Inc became part of HP
- 2011 – Nokia switches to Windows Phone from Symbian
- 2012 – Firefox OS
 - Based on HTML5
- 2012 – Samsung Tizen
- 2013 – Ubuntu Touch
- 2014 – Wearable devices boom

... nowadays ...

- 2015 – Windows 10 – multiplatform
- 2015 – Firefox OS extinct ...
- 2016 – Cyanogen OS is announced
- 2016 – Cyanogen OS disappears
- 2017 – Android on Nokia phones
- 2017 – Windows 10 Mobile still exists, but ...
 - ... zombie
- 2019 ??

... statistics

SALES – ALL PHONES



This course

- Two main platforms will be overviewed
 - Android
 - iOS
- We are going to make simple applications on that platforms

Android

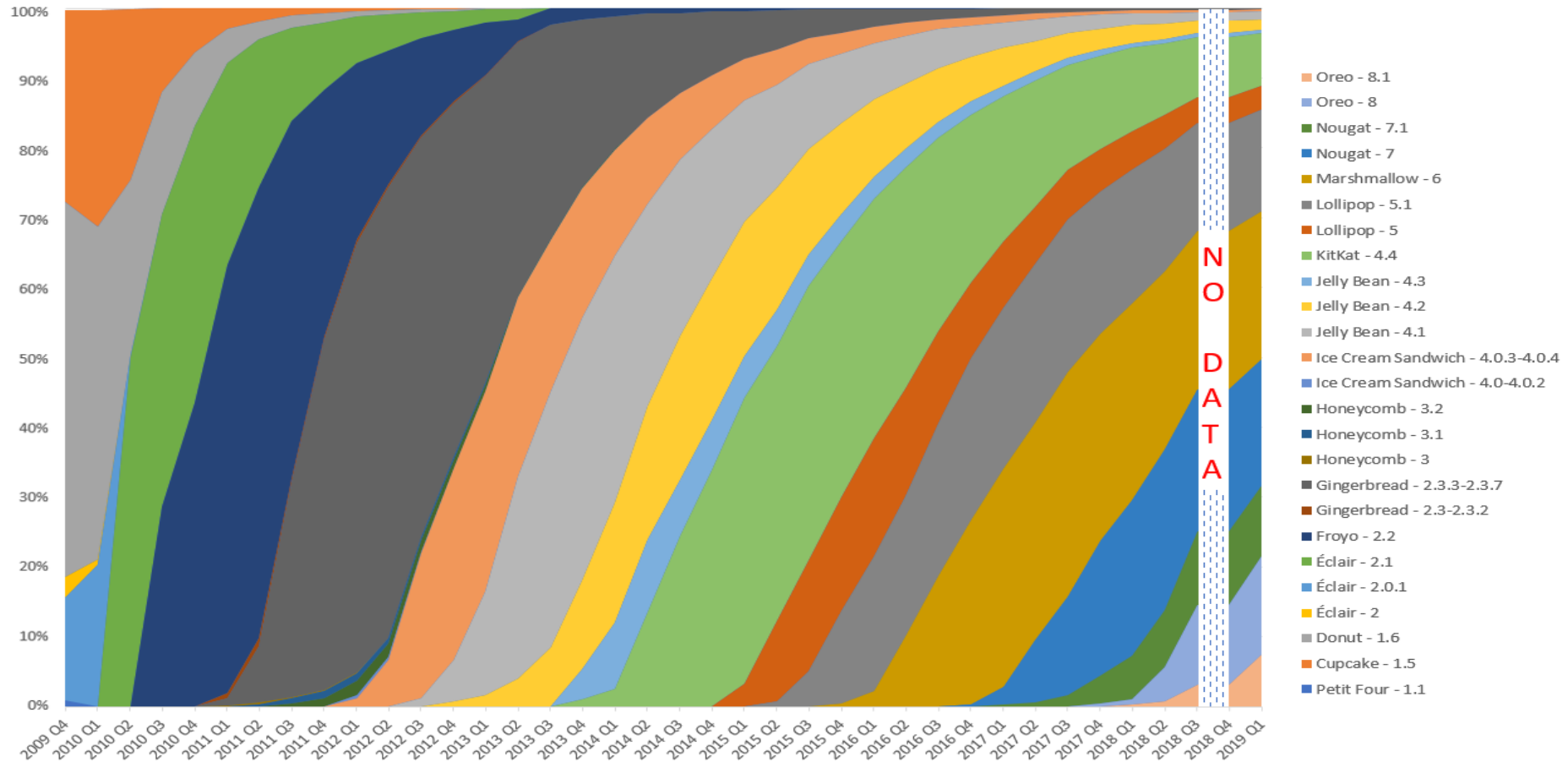


Android properties

- Supervised and developed by Google – software package
 - Linux + Android VM + Other applications
 - Many manufacturer, different hardware
 - Basically ARM and Intel processors
 - Different capabilities
 - Inaccurate implementation of specifications occur
 - Additional software are included, which are not part of the Android system
 - Development tools and emulator
 - Available for all platforms
- Main properties
 - Modular
 - Multitask, automatic memory managements, program libraries included
 - Almost arbitrary mobile communication technology is available (GSM ... LTE)
 - Wi-Fi (Client and AP), Ethernet (tethering), Bluetooth, NFC
 - Sensors: GPS, Triaxial accelerometer / magnetometer, thermometer, light sensor
 - Camera support, recording and playback, even stereo
 - HDMI support, accelerated 2D and 3D graphics, parallel computation

Introduced in	Version number	Name	API LEVEL
2007	β		β
2008	1.0		1
2009	1.1		2
2009	1.5	Cupcake	3
2009	1.6	Donut	4
2009	2.0	Eclair	5
2010	2.2	Froyo	8
2010	2.3	Gingerbread	9
2011	3.0	Honeycomb	11
2011	4.0	Ice Cream Sandwich	14
2013	4.1	Jelly Bean	16
2013	4.4	KitKat	19
2014	5.0	Lollipop	21
2015	6.0	Marshmallow	23
2016	7.X	Nougat	24
2017	8.X	Oreo	26
2018	9.X	Pie	28
2019	10	Android 10	29

Spread of different versions



Android platform – details

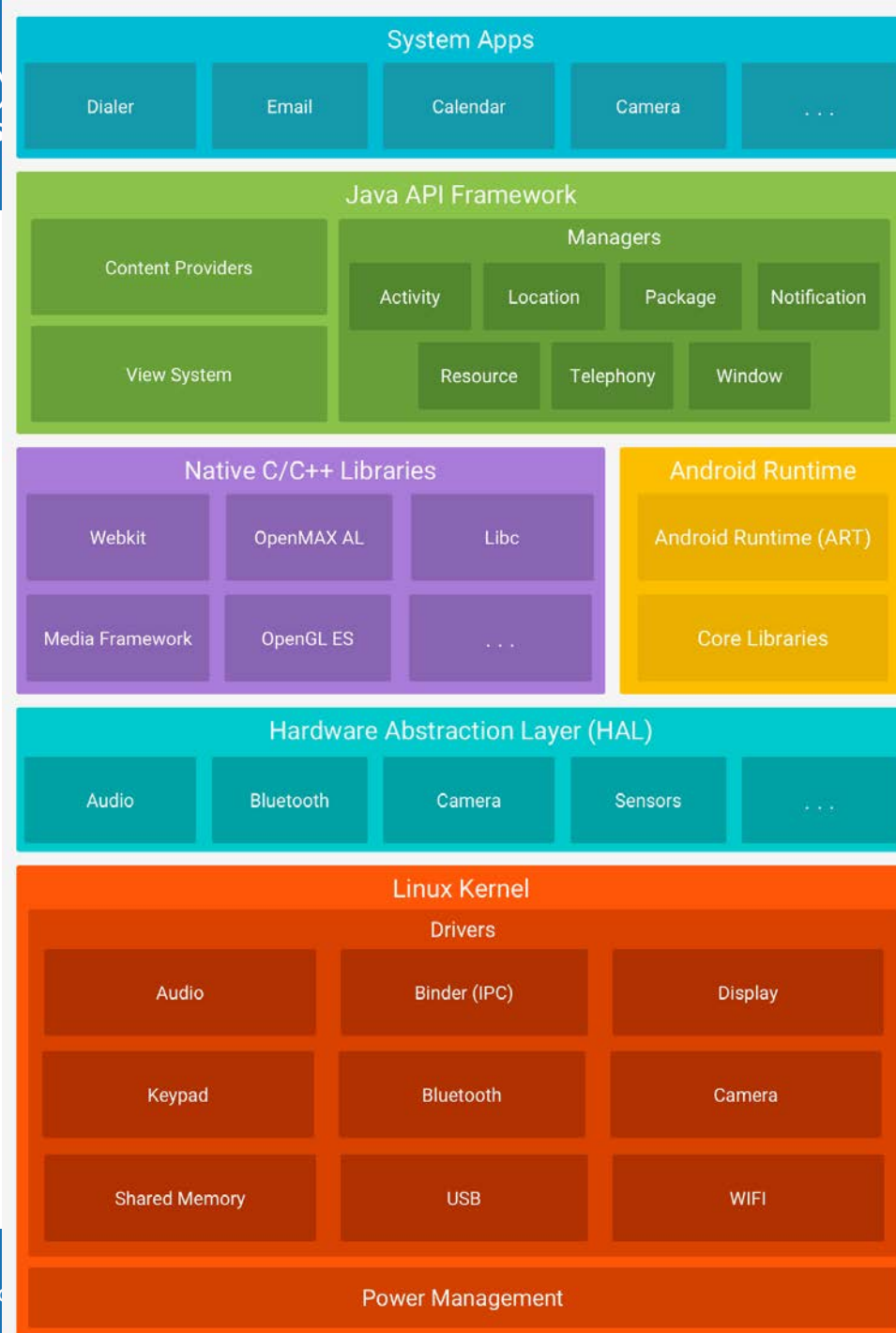
- Collection of several, different software
 - Middleware
 - Core application
 - Operating system
- Kotlin – programming language
 - On Java virtual machine
 - First appeared in 2011
 - Last stable release: 1.3.50 – August 2019.
 - For Android since 2017 Google I/O
 - Kotlin is designed to be an industrial-strength object-oriented language, and a "better language" than Java, but still be fully interoperable with Java code



Android platform – details

- Java based programming language (not equivalent to Java)
 - Some of the Java packages are excluded
 - java.applet
 - java.beans
 - javax.rmi
 - javax.print
 - Not JVM, but ART (Before Lollipop it was Dalvik VM)
 - Open source
 - Less memory is required compared to other VM-s
 - Multiple VM-s can be executed parallel, without the loss of efficiency
 - Ahead-Of-Time compiling and Just-In-Time compiling
 - *.java → *.class → *.dex → *.apk
 - The executed VM *.dex files, which are generated by compiling Java byte codes

Building blocks



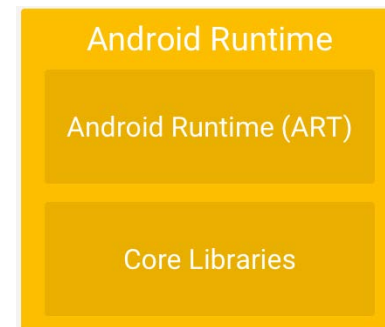
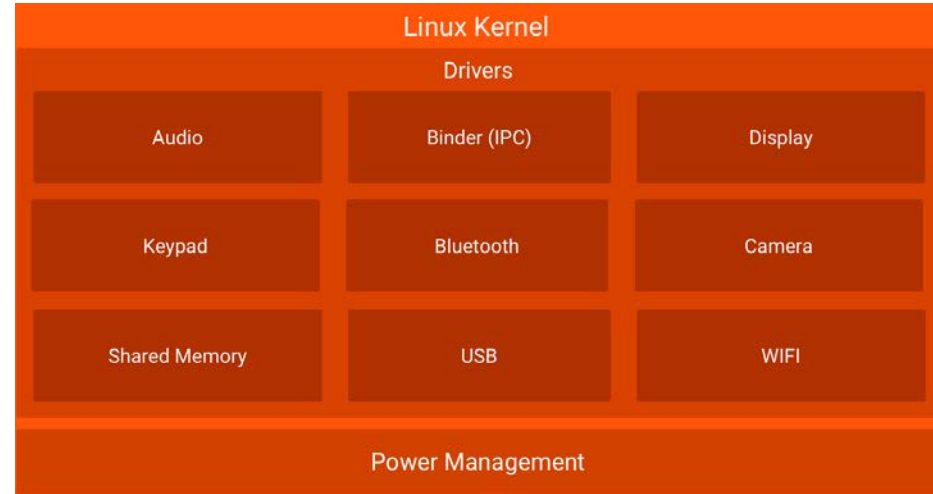
Details

- Kernel

- Most important component
- Abstraction level between hardware and software
- Memory, energy, network management
- Driver models
- Linux kernel version:
 - 4.4, 4.9 or 4.14

- Runtime

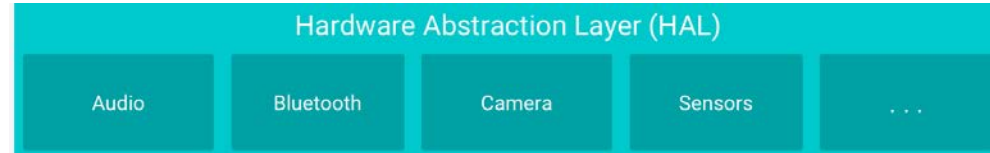
- Each android application is a separate VM process in the system
- ART
 - AOT, JIT, GC, ...
- Core Libraries
 - Remaining Java packages



Details

- HAL

- A layer over hardware components to hide the differences
- Code depends on actual device and architecture
 - Drivers



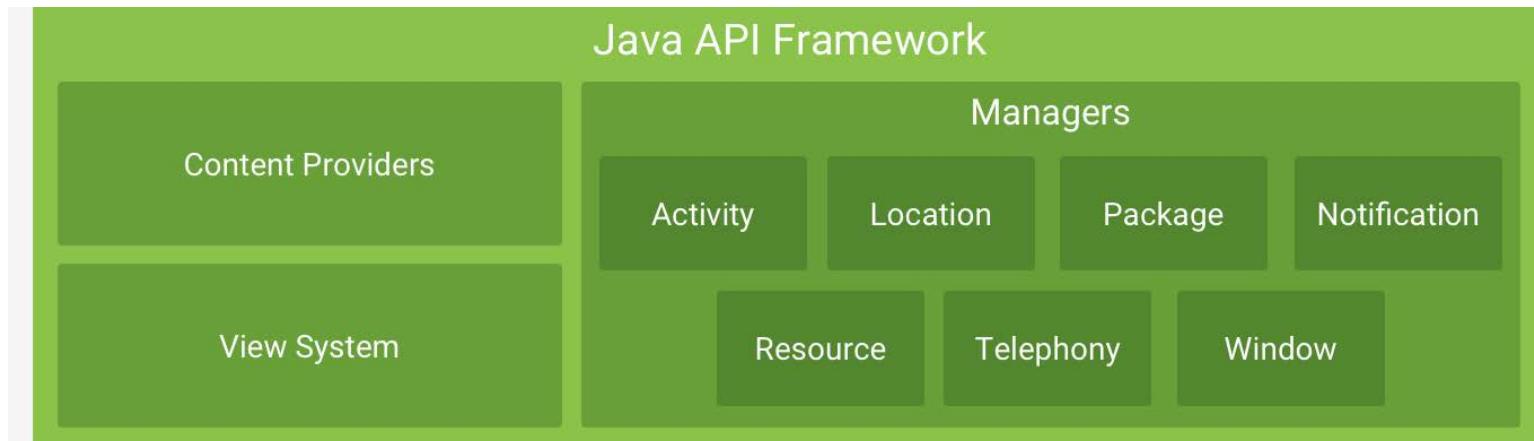
- Native C/C++ libraries

- Developers can access it through the app framework
- Media Framework:
 - Image/Music/Video can be managed, Media Codecs
- Webkit
- OpenGL, OpenAL and OpenMax
 - Graphical library, Sound library, etc.
- SQLite
 - Relational databases can be used
- NDK application components



Details

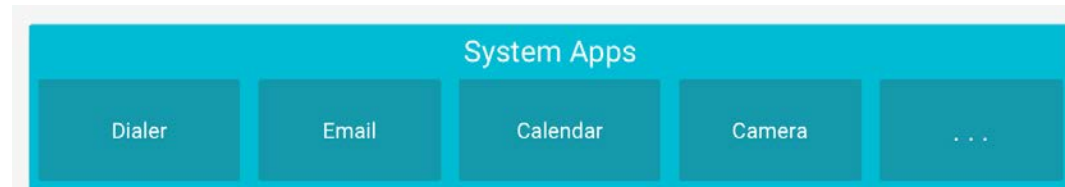
- API Framework
 - Open platform for developers
 - Makes possible to develop fast and efficient
 - High level of freedom: we can use the same resources and components as System applications do
 - In addition these components can be substituted
 - Contains the resource manager, and framework responsible for view system
 - It is possible to modify the system components, designs
 - Manufacturers often utilize this possibility



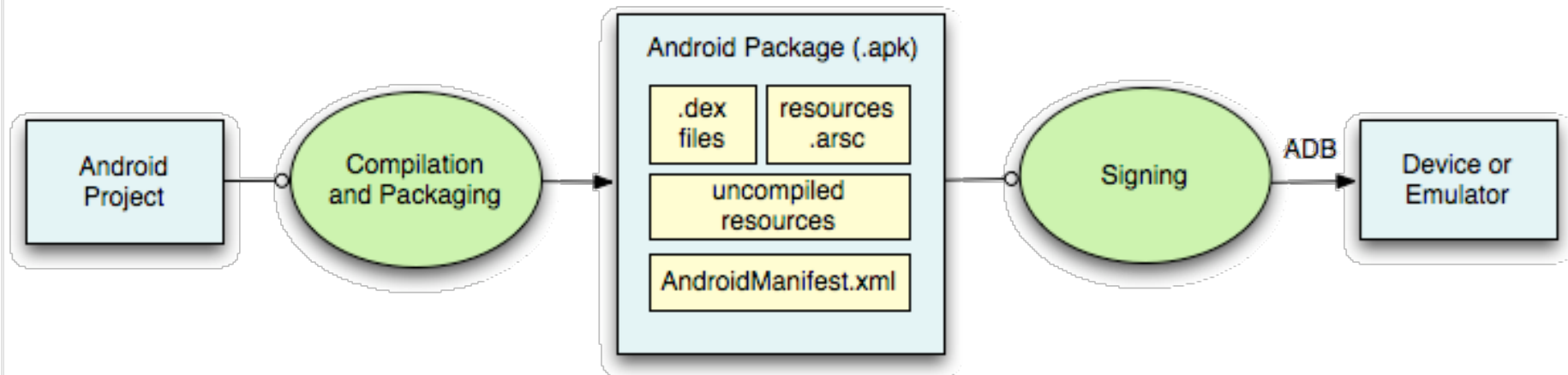
Details

- System Applications

- E-mail client
- Sms client
- Phone Book
- Calendar



- Android project compilation process



Android development tools

- Android Studio
 - Java SDK
- Android SDK
 - Compiler and program libraries
 - Emulator
 - This is an emulator, where executes the entire Android operating system over your OS on your device
- Android NDK
 - For native (C/C++) libraries
 - Currently you do not need it
- As a result we can develop Android software on any of the main desktop platforms



iOS

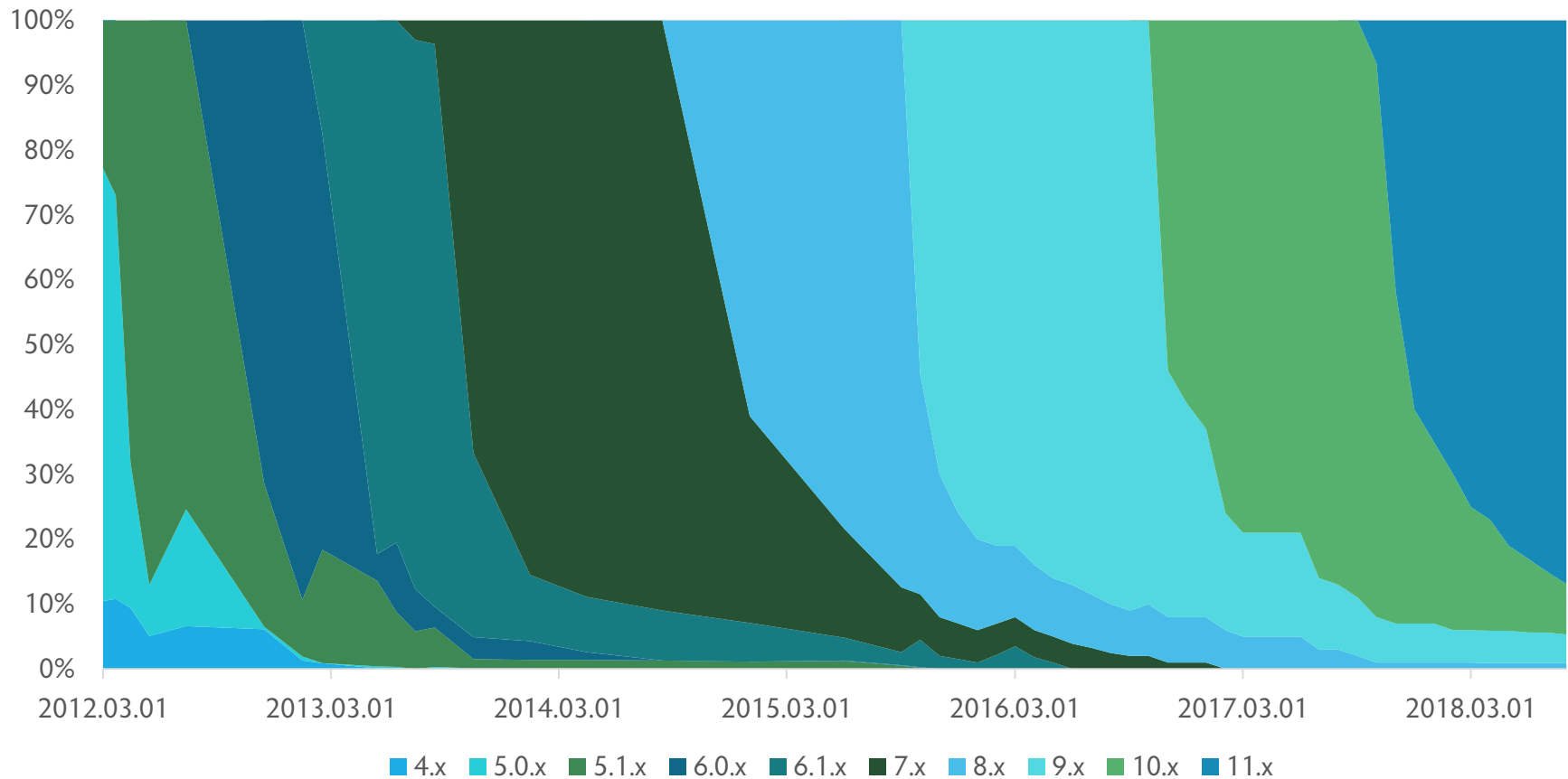
iOS

iOS properties

- System and applications developed by Apple
 - Based on Darwin, originated from Mac OS X
 - Single manufacture, a few, well-specified hardware
 - ARM processors
 - Transitions between versions are supported
 - For example: new screen resolutions are multiplications of previous ones
 - It can result weird properties, such as incompatibilities with other standards
 - Developments tools and simulator
 - Only for Apple systems
- Main properties
 - Multitask, several built-in program libraries
 - Support for mobile communication systems (GSM ... LTE)
 - WiFi, Bluetooth, NFC
 - Sensors: GPS, Triaxial accelerometer / magnetometer
 - Camera support, media recording and playback
 - HDMI support, accelerated 2D and 3D graphics

Introduction	Version	Name	Required Xcode version
2007	1.0	iPhone OS	
2008	2.0	iPhone OS 2.0 Final	
2009	3.0	iPhone OS 3.0 Final	
2010	3.2	iPhone OS 3.2 Final	
2010	4.0	iOS 4.0 Final	
2010	4.3	iOS 4.3 Final	Xcode 4
2010	5.0	iOS 5.0 Final	Xcode 4.2
2012	6.0	iOS 6.0 Final	Xcode 4.5
2013	6.1	iOS 6.1 Final	Xcode 4.6
2013	7.0	iOS 7.0 Final	Xcode 5
2014	7.1	iOS 7.1 Final	Xcode 5.1
2014	8.0	iOS 8.0 Final	Xcode 6
2015	8.4	iOS 8.4 Final	Xcode 6.4
2015	9.0	iOS 9.0 Final	Xcode 7.0
2016	9.3.5	iOS 9.3.5 Final	Xcode 7.3
2017	10.3.3	iOS 10.3.3 Final	Xcode 8.3
2018	11.4.1	iOS 10.4.1 Final	Xcode 9.x
2019	12.4.1	iOS 12.4.1 Final	Xcode 10.x

Spread of different iOS versions



iOS devices

- Low number of different devices
 - Several models, different parameters for each models
 - The maximal available iOS version can be read from the table
 - It implies the supported devices of an applications

iOS	iPhone									iPod Touch		
	5	5C	5S	6/Plus	6S/ SE	7/Plus	8/Plus	X/XS	XR	5th	6th	7th
9.3										X		
10	X	X										
12 limited			X	X								
12 full					X	X	X	X	X		X	X

- iOS 12 on
 - iPad Pro *, 1st, 2nd, 3rd
 - iPad Mini 2nd, 3rd, 4th, Mini (2019)
 - iPad Air & Air 2 & Air (2019)
 - iPad 2017 & 2018

iOS building blocks

- Different software and software levels
 - Based on OS X (Darwin)
- Development on SWIFT language
 - Brand new language, not extension for Objective-C
 - However its logic can be understand better based on Objective-C
- Development on Objective-C language
 - Based on C
 - Thin layer on the C
 - Fully object oriented
 - Smalltalk messaging model

Cocoa Touch

Media

Core Services

Core OS

Details

- Core OS

- Mac OS X Kernel
 - XNU
- TCP/IP
- Sockets
- Energy managements
- File systems
- Security

- Core Services

- Network management
- Collections
- Preference storage
- URL handling
- File accessing
- Contacts
- Embedded SQL database manager
- Core Location
 - To determine the location of the devices (GPS, Cell, etc.)
- Thread managements
- CoreMotion
 - To retrieve the accelerometer data

Details

- Media

- Core Audio
- Open AL
 - Open Audio Library
- Sound mixer
- Sound recorder
- Video playback
- Supporting media formats
- 2D acceleration
- Core Animation
- OpenGL ES

- Cocoa Touch

- Multi-touch and gestures
- Camera support
- Localization (multilingual apps)
- Views and view hierarchies
- Accelerometer support
- Web view
- Map kit
- Handling notifications
- Core Motion
 - Provide and process motion related information

iOS development tools

- XCode
 - With iOS SDK
 - An OS X required as operating system (and an Apple computer)
 - Newer iOS requires newer XCode, and newer OS X
 - And newer hardware
- SDK includes a simulator
 - This is not emulator, it is a simulator
 - A simulator calculates the corresponding output for a specific input
 - Thus the iOS systems is not running on the test-environment.



Mobile platform technologies

Next week



Pázmány Péter Catholic University
Faculty of Information Technology and Bionics

Basics of Mobile Application Development

Hardware



General capabilities

Top existing hardware

iPhone 11 Pro

- 2.65 GHz 6 core SOC
 - 2 High performance cores
 - 4 energy efficiency cores
- +3 cores GPU
- Neural Network Engine
 - Corresponding API
- 6 Gb LPDDR4X RAM

Pixel 3 XL

- 2.5 Ghz 4 cores
- 1.6 Ghz 4 cores
- Dedicated GPU
 - Adreno 630 (multicore)
- AI accelerator chip
 - Parallel processing API
- 4 Gb LPDDR4X RAM

Camera arrays + extra

iPhone X

- Rear: 3 components
 - 12 Mp, multiple lens
 - Uwide, Wide, Telephoto
 - Up to 4K
 - Up to 240 fps
- Front: 12 Mp
 - 4K@30fps
 - HDR
- FaceID
 - 30,000 infrared dots + sensor

Pixel 3

- Rear: 12.2 Mp
 - Dual Pixel Phase AF
 - HDR+
 - Up to 4K
 - Up to 240 fps
- Front: 8 Mp
 - Two cameras
 - + Wide angle lens
 - Depth sensor
 - 1080p@30fps
- Fingerprint sensor



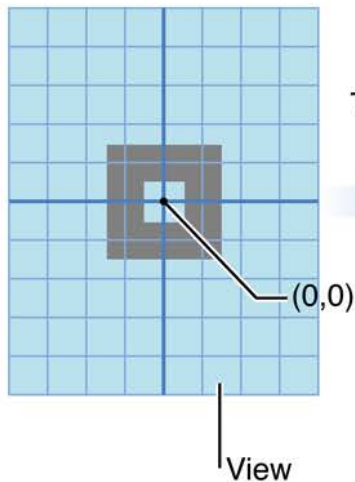
Pázmány Péter Catholic University
Faculty of Information Technology and Bionics

Sensors systems

Detour – Screen coordinate systems

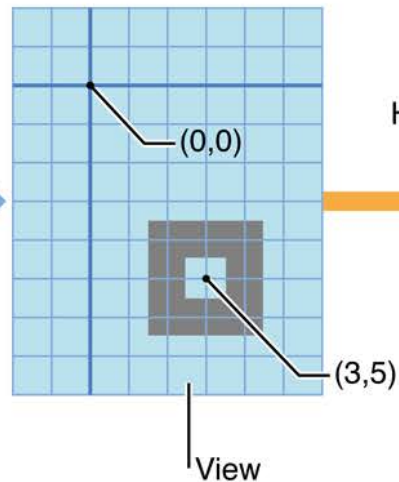
- In iOS

Drawing (User) Coordinates



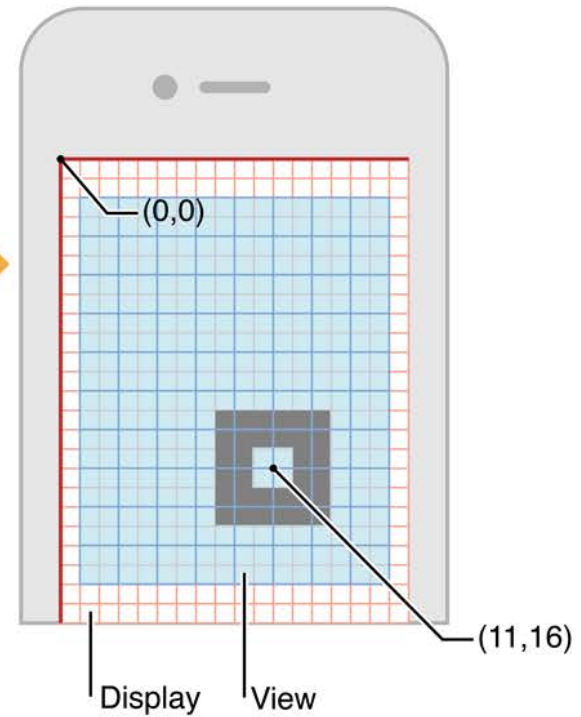
Current
Transformation
Matrix (CTM)

View Coordinates

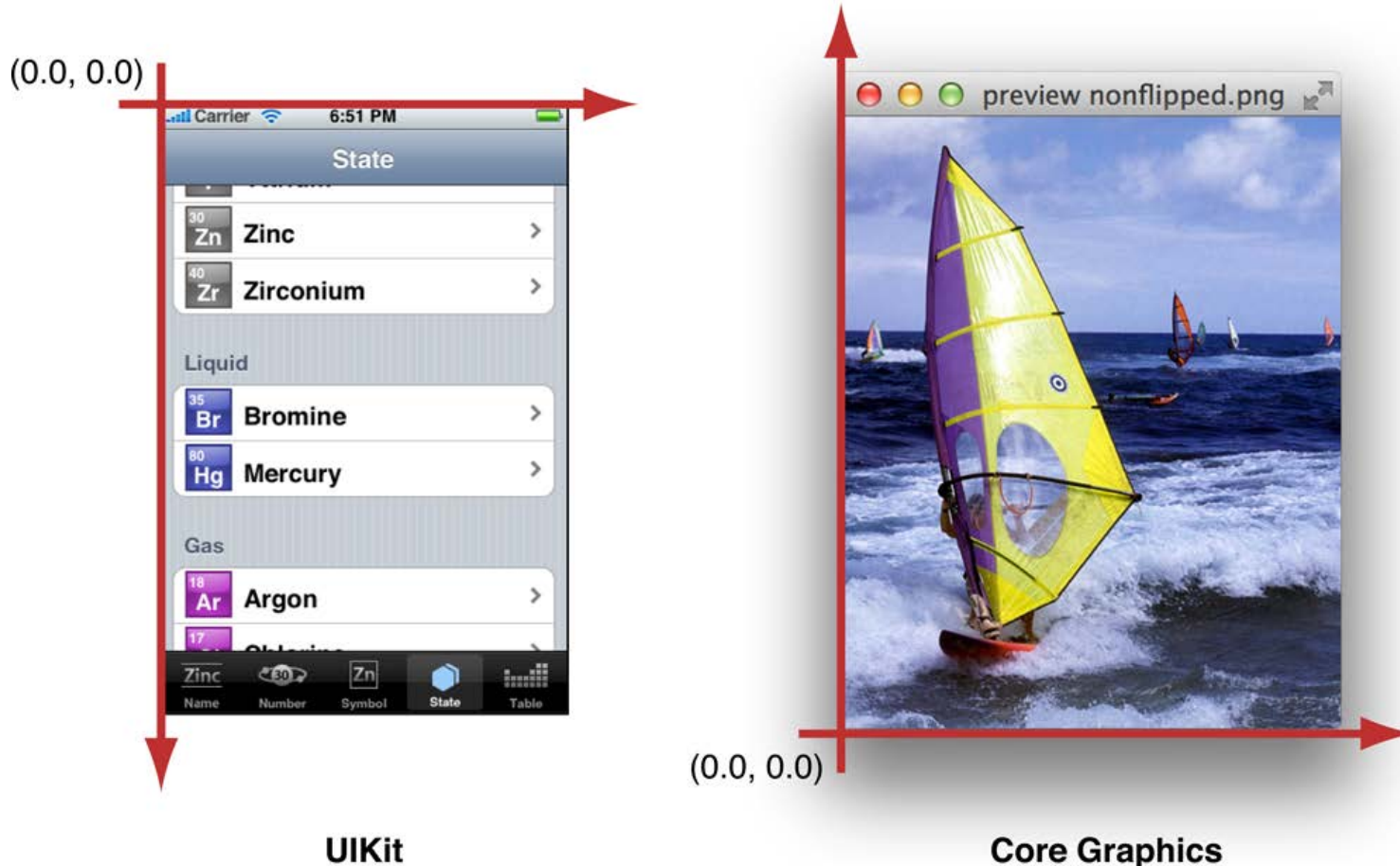


Hardware
Scaling

Hardware Coordinates



Detour – Screen coordinate systems

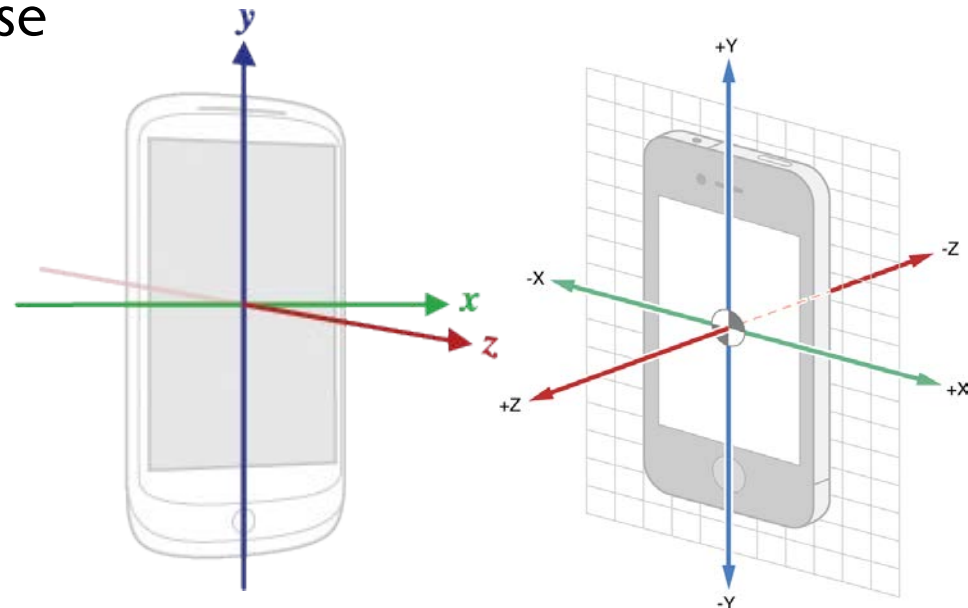
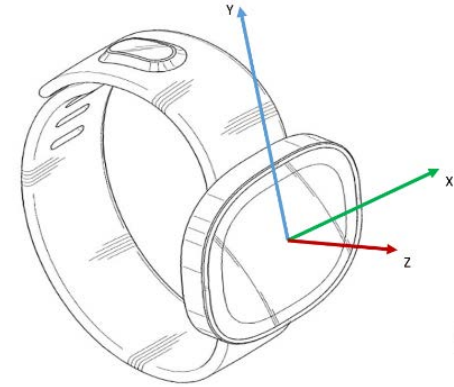


Detour – Screen coordinate systems

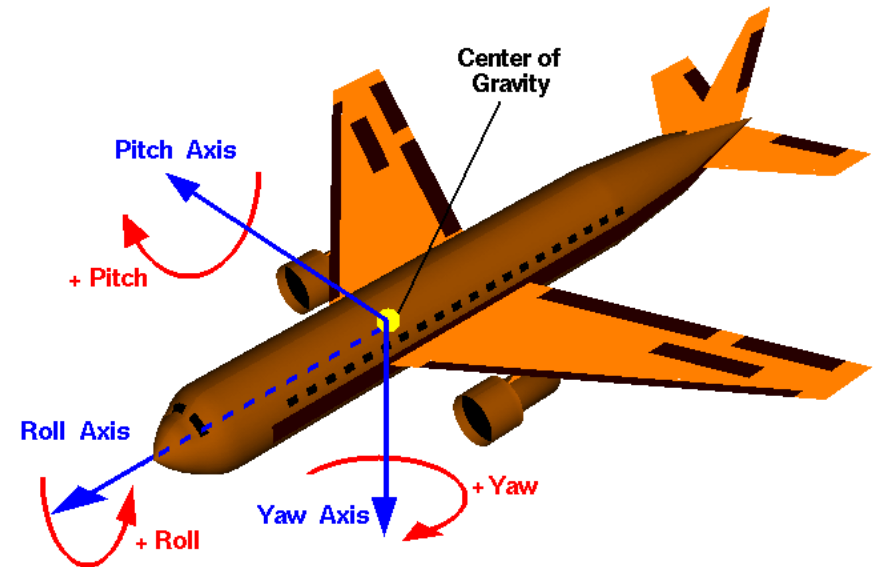
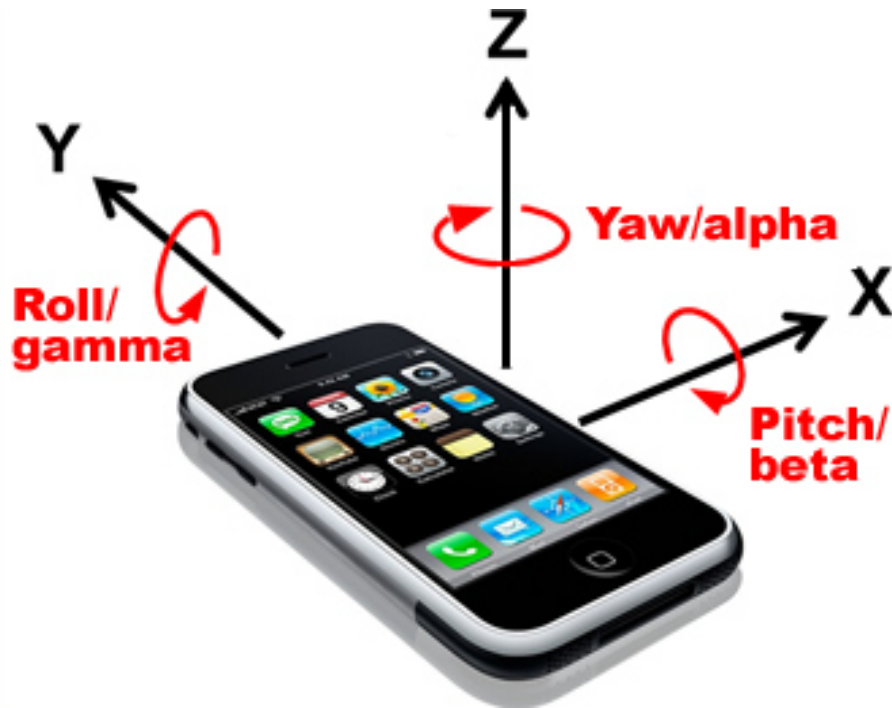
- In Android the top-left point is considered as zero point
 - In any orientation
 - The origin is not an absolute point, it depends on orientation
- 3D graphics introduces the third axis

Coordinate systems

- The tri-axial sensors use the following coordinate system
 - Always the base orientation have to be considered
 - The axes do not change in case of orientation change
 - X-Y plane is the screen itself
- Axis
 - X: short edge, left to right
 - Y: long edge, bottom to up
 - Z: back to front

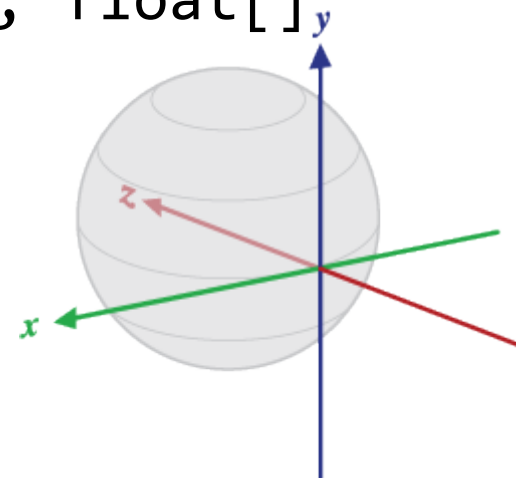


Rotational angles - iOS



Android Orientational coordinates

- Using the inclination matrix the angle can be calculated
 - `float getInclination (float[] I);`
 - Result in radian
- Using the rotation matrix the orientational angles can be calculated
 - `float[] getOrientation (float[] R, float[] values)`
 - Angles are
 - Around the Z axis (Azimuth)
 - Around the X axis (Roll)
 - Around the Y axis (Pitch)
 - All of them counter clockwise



Sensors in Android API

Sensor	Type	Description	Common Uses
TYPE_ACCELEROMETER	Hardware	Measures the acceleration force in m/s^2 that is applied to a device on all three physical axes (x, y, and z), including the force of gravity.	Motion detection (shake, tilt, etc.).
TYPE_AMBIENT_TEMPERATURE	Hardware	Measures the ambient room temperature in degrees Celsius ($^{\circ}\text{C}$). See note below.	Monitoring air temperatures.
TYPE_GRAVITY	Software or Hardware	Measures the force of gravity in m/s^2 that is applied to a device on all three physical axes (x, y, z).	Motion detection (shake, tilt, etc.).
TYPE_GYROSCOPE	Hardware	Measures a device's rate of rotation in rad/s around each of the three physical axes (x, y, and z).	Rotation detection (spin, turn, etc.).
TYPE_LIGHT	Hardware	Measures the ambient light level (illumination) in lx.	Controlling screen brightness.
TYPE_LINEAR_ACCELERATION	Software or Hardware	Measures the acceleration force in m/s^2 that is applied to a device on all three physical axes (x, y, and z), excluding the force of gravity.	Monitoring acceleration along a single axis.
TYPE_MAGNETIC_FIELD	Hardware	Measures the ambient geomagnetic field for all three physical axes (x, y, z) in μT .	Creating a compass.

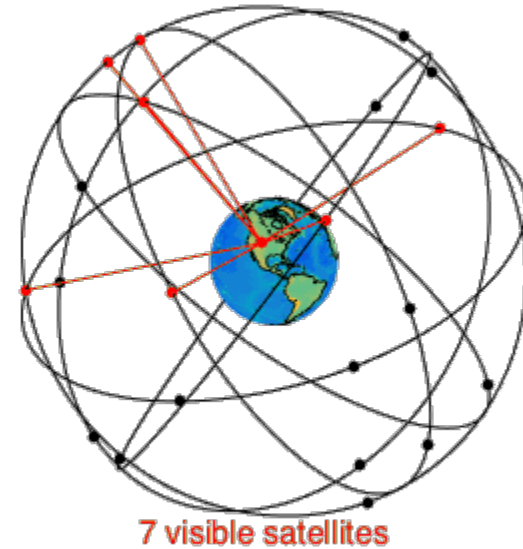
Sensors in Android API

Sensor	Type	Description	Common Uses
TYPE_ORIENTATION	Software	Measures degrees of rotation that a device makes around all three physical axes (x, y, z). As of API level 3 you can obtain the inclination matrix and rotation matrix for a device by using the gravity sensor and the geomagnetic field sensor in conjunction with the getRotationMatrix() method.	Determining device position.
TYPE_PRESSURE	Hardware	Measures the ambient air pressure in hPa or mbar.	Monitoring air pressure changes.
TYPE_PROXIMITY	Hardware	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	Phone position during a call.
TYPE_RELATIVE_HUMIDITY	Hardware	Measures the relative ambient humidity in percent (%).	Monitoring dewpoint, absolute, and relative humidity.
TYPE_ROTATION_VECTOR	Software or Hardware	Measures the orientation of a device by providing the three elements of the device's rotation vector.	Motion detection and rotation detection.
TYPE_TEMPERATURE	Hardware	Measures the temperature of the device in degrees Celsius (°C). This sensor implementation varies across devices and this sensor was replaced with the TYPE_AMBIENT_TEMPERATURE sensor in API Level 14	Monitoring temperatures.

Sensors in iOS API

- Core Motion
 - Three dimensional gyroscope
 - Accelerometer
 - Pedometer
 - Magnetic sensor – compass
 - Altitude

Positioning



Positioning systems

System	GPS	GLONASS	BeiDou	Galileo	NAVIC	QZSS
Owner	United States	Russian Federation	China	European Union	India	Japan
Coding	CDMA	FDMA	CDMA	CDMA	CDMA	CDMA
Orbital altitude	20,180 km	19,130 km	21,500 km	23,222 km	36,000 km	32,600 km - 39,000 km
Precision	5m (no DGPS or WAAS)	4.5m – 7.4m	10m (Public) 0.1m (Encrypted)	1m (Public) 0.01m (Encrypted)	10m (Public) 0.1m (Encrypted)	1m (Public) 0.1m (Encrypted)
Status	Operational	Operational	Basic services operational, To be completed in H1 2020	18 satellites operational, 12 additional satellites 2016-2020	Operational	

Common features

- Satellites in space
 - Influenced by weather conditions
 - All around on Earth, and nearby, in space
 - Mail satellites, and spares
 - Atomic clock inside
 - CDMA coding scheme
- Satellites are broadcasting its positions and time
 - Data from 3+1 satellites is enough to calculate position
- Transmitting on several different frequencies
 - For civilian and military objectives

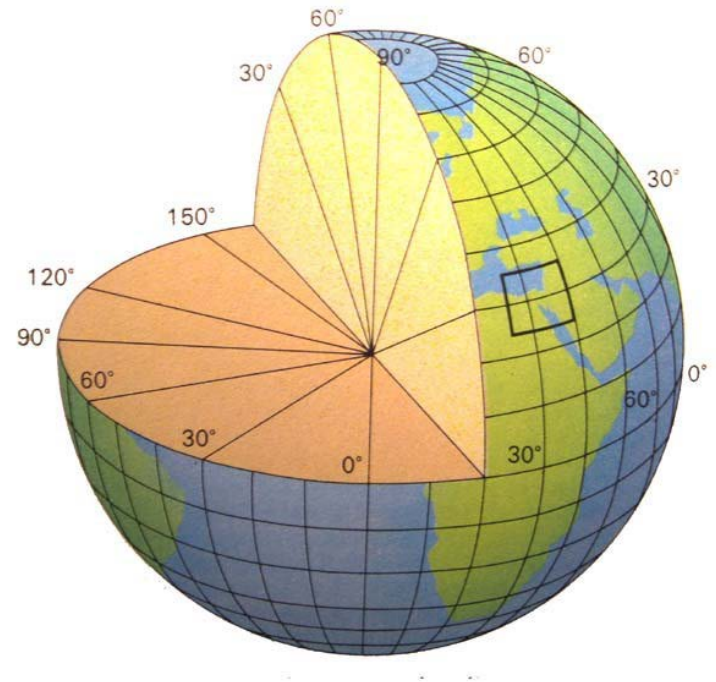


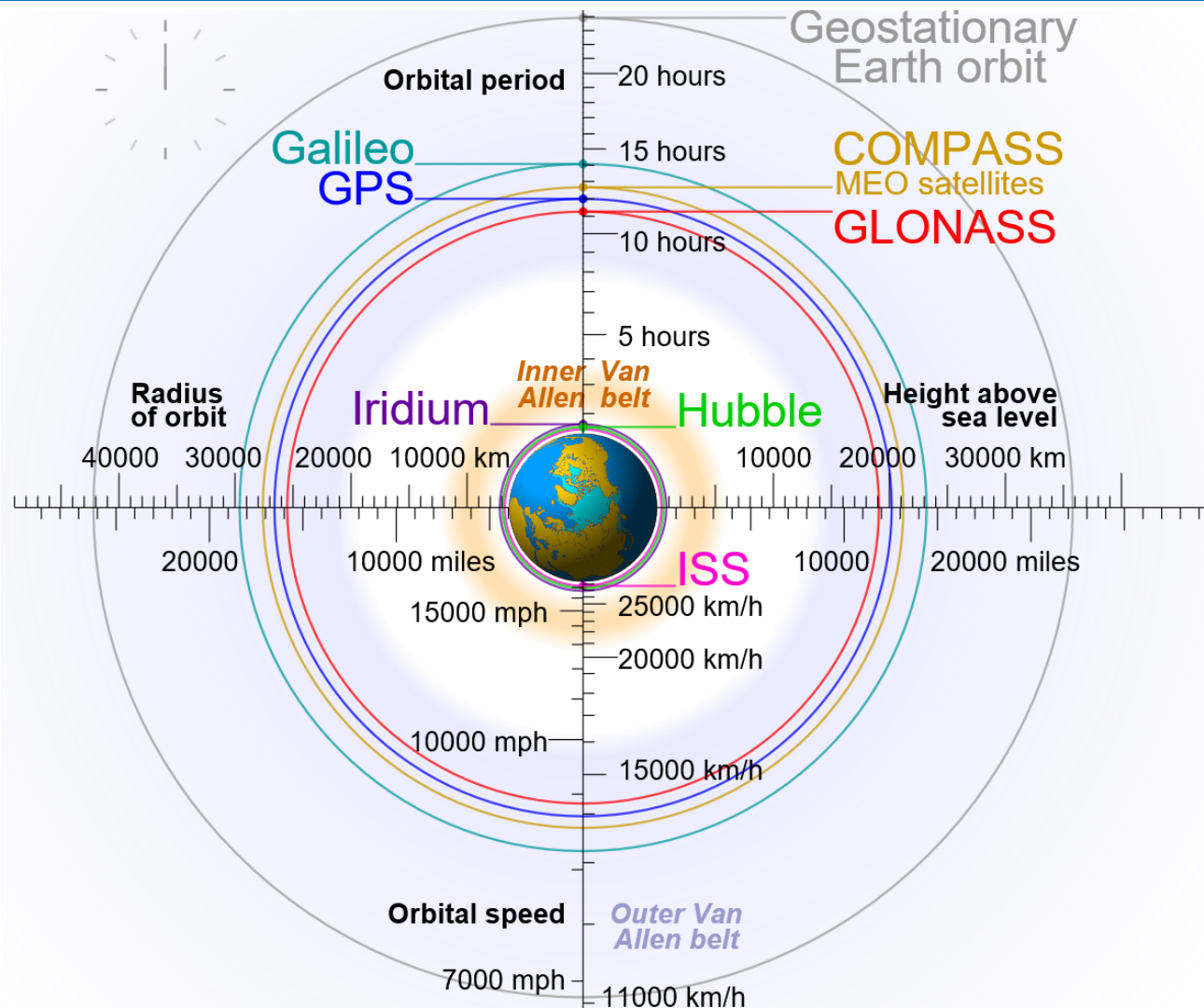
GPS History

- 1978 - 1992
 - 1987: Experimental,
 - 1989: Second generation
- 1993
 - 24 satellites is available
 - 3 channel devices.
 - Positioning requires ~10 minutes
- 1998
 - RDS in Hungary
- 2000
 - 12 channel devices
 - 15-20 seconds enough to calculate position
 - Accuracy in Civilian usage is 20 meters
- 2005
 - Next generation of satellites
 - Secondary signal to increase accuracy
- 2010
 - Second generation of earth control

Coordinate system

- World Geodetic System (WGS)
 - Pole of the system is the mass center of Earth
 - The 0th line of longitude is the IERS reference line
 - Z axis is the rotation axis of the Earth (averaged)
 - Y plane is perpendicular to the rotation axis (the mass center is on the Y plane)





Other than GPS

GLONASS

- Russian
 - 17/8 rev / day
 - 24 satellites + spares
 - FDMA
- Finished in 2011, but the system design is older
 - Beginning of development is back to the 70's

Galileo

- EU
 - 17/10 rev / day
 - 26 satellites (spares)
 - CDMA
- Will be completed in 2020

Other than GPS

BeiDou

- Chinese
 - MEO
 - 17/9 rev/day
 - 23 satellites
 - 35 by 2020
- Will be completed by H1 2020

NAVIC/QZSS

- Indian / Japanese
- Both CDMA
- Regional systems
- GEO / GSO

Supported positioning systems

- GPS
 - Almost in all device
- Other systems
 - Number is increasing
- Pixel 3 XL
 - GPS (with A-GPS), GLONASS, BeiDou, GALILEO
- Iphone 11
 - GPS (with A-GPS), GLONASS

Assisted GPS – AGPS

- A system that often significantly improves the startup performance of a GPS satellite-based positioning system
- Assistance falls into two categories
 - Mobile Station Based (MSB): Information used to acquire satellites more quickly.
 - It can supply orbital data or almanac for the GPS satellites to the GPS receiver, enabling the GPS receiver to lock to the satellites more rapidly in some cases.
 - The network can provide precise time.
 - Mobile Station Assisted (MSA): Calculation of position by the server using information from the GPS receiver.
 - The device captures a snapshot of the GPS signal, with approximate time, for the server to later process into a position.
 - The assistance server has a good satellite signal and plentiful computation power, so it can compare fragmentary signals relayed to it.
 - Accurate, surveyed coordinates for the cell site towers allow better knowledge of local ionospheric conditions and other conditions affecting the GPS signal than the GPS receiver alone, enabling more precise calculation of position.

Android Example

```
LocationManager locationManager = (LocationManager)
this.getSystemService(Context.LOCATION_SERVICE);

LocationListener locationManager = new LocationListener() {
    public void onLocationChanged(Location location) {
        makeUseOfNewLocation(location);
    }

    public void onStatusChanged(String provider, int status, Bundle extras)
        {}

    public void onProviderEnabled(String provider)
        {}

    public void onProviderDisabled(String provider)
        {}
};

locationManager.requestLocationUpdates(
    LocationManager.NETWORK_PROVIDER, 0, 0, locationManager);
```


iOS - Example

- ```
var locationManager: CLLocationManager!
locationManager = CLLocationManager();
locationManager.delegate = self;
locationManager.desiredAccuracy = kCLLocationAccuracyBest
locationManager.startUpdatingLocation();
```
- ```
func locationManager(manager: CLLocationManager!,  
    didUpdateLocations locations: [AnyObject]!) {  
    var location:CLLocation =  
        locations[locations.count-1] as CLLocation  
    println("locations = \(locations)")  
    txtLatitude.text = "\(location.coordinate.latitude)";  
    txtLongitude.text = "\(location.coordinate.longitude)";  
}
```



Bluetooth

Bluetooth

- History

- Started in 1994
- In 1998 the Bluetooth SIG has been formed (Special Interest Group)
 - IBM, Intel, Nokia, Toshiba, Lucent, Microsoft
- First working ad hoc network
- Standardized by IEEE: IEEE 802.15

- Properties

- Easy to implement
 - 1-2 cm size
- Low power consumption
 - 1-2 % of mobile energy
- Low cost
 - 5 \$

Bluetooth version

- 1.0
 - First version – with bugs
- 1.1
 - IEEE standard
 - RSSI indication, unencrypted channel managements
- 1.2
 - Faster
 - Max 721 kbit/s
- 2.0 + EDR
 - Enhanced Data Rate max 3 Mbit/s
 - EDS is optional
- 2.1 + EDR
 - SSP – Secure Simple Pairing
- 3.0 + HS
 - 24 Mbit/s
 - New working methods
 - Advances energy management

Bluetooth version

- 4.0
 - More energy protocols
 - Extension of the Core Specification
- 4.1
 - Mobile Wireless Service Coexistence Signaling
 - Train Nudging and Generalized Interlaced Scanning
 - Low Duty Cycle Directed Advertising
 - L2CAP Connection Oriented and Dedicated Channels with Credit Based Flow Control
 - Dual Mode and Topology
 - LE Link Layer Topology
 - 802.11n PAL
 - Audio Architecture Updates for Wide Band Speech
 - Fast Data Advertising Interval
 - Limited Discovery Time
- 4.2
 - IP over Bluetooth
 - Improvements

Bluetooth version

- 5.0 – 2016
- 5.1 – 2019
 - Angle of Arrival
 - Mesh-based model
 - Many improvements

Protocols

- Not every protocol is supported by all devices / platforms
 - **LMP**
 - The Link Management Protocol (LMP) is used for set-up and control of the radio link between two devices. Implemented on the controller.
 - **L2CAP**
 - The Logical Link Control and Adaptation Protocol (L2CAP) used to multiplex multiple logical connections between two devices using different higher level protocols.
 - **SDP**
 - The Service Discovery Protocol (SDP) allows a device to discover services offered by other devices, and their associated parameters
 - **RFCOMM**
 - Radio Frequency Communications (RFCOMM) is a cable replacement protocol used to generate a virtual serial data stream
 - **BNEP**
 - The Bluetooth Network Encapsulation Protocol (BNEP) is used for transferring another protocol stack's data via an L2CAP channel. Its main purpose is the transmission of IP packets in the Personal Area Networking Profile.

Protocols

- Not every protocol is supported by all devices / platforms
 - **BNEP**
 - The Bluetooth Network Encapsulation Protocol (BNEP) is used for transferring another protocol stack's data via an L2CAP channel. Its main purpose is the transmission of IP packets in the Personal Area Networking Profile.
 - **AVCTP**
 - The Audio/Video Control Transport Protocol (AVCTP) is used by the remote control profile to transfer AV/C commands over an L2CAP channel.
 - **AVDTP**
 - The Audio/Video Distribution Transport Protocol (AVDTP) is used by the advanced audio distribution profile to stream music to stereo headsets over an L2CAP channel.
 - **TCS**
 - The Telephony Control Protocol – Binary (TCS BIN) is the bit-oriented protocol that defines the call control signaling for the establishment of voice and data calls between Bluetooth devices.
 - **Further: PPP, TCP/IP/UDP, OBEX, WAE/WAP**

Bluetooth Profiles

Advanced Audio Distribution Profile (A2DP)
Attribute Profile (ATT)
Audio/Video Remote Control Profile (AVRCP)
Basic Imaging Profile (BIP)
Basic Printing Profile (BPP)
Common ISDN Access Profile (CIP)
Cordless Telephony Profile (CTP)
Device ID Profile (DIP)
Dial-up Networking Profile (DUN)
Fax Profile (FAX)
File Transfer Profile (FTP)
Generic Audio/Video Distribution Profile (GAVDP)
Generic Access Profile (GAP)
Generic Attribute Profile (GATT)
Generic Object Exchange Profile (GOEP)
Hard Copy Cable Replacement Profile (HCRP)
Health Device Profile (HDP)

Hands-Free Profile (HFP)
Human Interface Device Profile (HID)
Headset Profile (HSP)
Intercom Profile (ICP)
LAN Access Profile (LAP)
Message Access Profile (MAP)
Object EXchange (OBEX)
Object Push Profile (OPP)
Personal Area Networking Profile (PAN)
Phone Book Access Profile (PBAP, PBA)
Proximity Profile (PXP)
Serial Port Profile (SPP)
Service Discovery Application Profile (SDAP)
SIM Access Profile (SAP, SIM, rSAP)
Synchronization Profile (SYNCH)
Synchronisation Mark-up Language Profile (SyncML)
Video Distribution Profile (VDP)
Wireless Application Protocol Bearer (WAPB)

Supported profiles in iOS

Device	Hands-Free Profile (HFP 1.6)	Phone Book Access Profile (PBAP)	Advanced Audio Distribution Profile (A2DP)	Audio/Video Remote Control Profile (AVRCP 1.4)	Personal Area Network Profile (PAN)	Human Interface Device Profile (HID)	Message Access Profile (MAP)
iPhone 4 and later	✓	✓	✓	✓	✓	✓	✓
iPhone 3GS	✓	✓	✓	✓	✓	✓	-
iPhone 3G	✓	✓	✓	✓	✓	-	-
Original iPhone	✓	✓	-	-	-	-	-
iPad 2 and later	✓	-	✓	✓	✓	✓	-
iPad (1st generation)	-	-	✓	✓	✓	✓	-
iPod touch (4th generation and later)	✓	-	✓	✓	✓	✓	-
iPod touch (2nd and 3rd generation)	-	-	✓	✓	✓	✓	-

Supported profiles in Android

Feature		Android version				
Name	Description	6.0	7.0	7.1	7.1.2	8.0
SAP	SIM Access Profile	1.1	1.1	1.1	1.1	1.1
MAP	Message Access Profile for SMS	1.2	1.2	1.2	1.2	1.2
OPP	Object Push Profile	1.1	1.1	1.1	1.1	1.2
OBEX over L2CAP	OBject EXchange over Logical Link Control and Adaptation Protocol	Yes	Yes	Yes	Yes	Yes
HFP Audio Gateway	Hands-Free Profile	1.6	1.6	1.7	1.7	1.7
HSP	Headset Profile	1.2	1.2	1.2	1.2	1.2
A2DP	Advanced Audio Distribution Profile	1.2	1.2	1.2	1.2	1.2
AVRCP	Audio/Video Remote Control Profile	1.3	1.3	1.3	1.3	1.4
HID	Human Interface Device Profile	1.0	1.0	1.0	1.0	1.0

Supported profiles in Android

Feature		Android version				
Name	Description	6.0	7.0	7.1	7.1.2	8.0
HID	Human Interface Device Profile	1.0	1.0	1.0	1.0	1.0
PBAP	Phone Book Access Profile	1.1.1	1.1.1	1.1.1	1.1.1	1.2
HDP	Health Device Profile	1.0	1.0	1.1	1.1	1.1
SPP	Serial Port Profile	1.2	1.2	1.2	1.2	1.2
PAN / BNEP	Personal Area Networking Profile / Bluetooth Network Encapsulation Protocol	1.0	1.0	1.0	1.0	1.0
DIP	Device ID Profile	1.3	1.3	1.3	1.3	1.3
HOGP 1.0	HID over GATT	Yes	Yes	Yes	Yes	Yes
HD Audio¹	See "Advanced audio codecs" above	No	No	No	No	Yes



GSM-like ...

History of mobile services

- 1. generation
 - ...
- 2. generation
 - GSM + CSD (circuit switching) (Europe)
 - CDMA based system (IS-54, IS-136, IS-95) (North America)
 - PSH (Far East)
- 2.x generation
 - GSM/3GPP variant
 - HSCSD (57,6 kbps) – analogue with traditional modems
 - GPRS (~80/20 kbps)
 - EGPRS = EDGE ~177,6/118,4 kbps
 - CDMA/3GPP2 variant
 - CDMA2000

History of mobile services

- 3. generation
 - 3GPP variant
 - UMTS (**WCDMA**, TD-CDMA, TD-SCDMA)
 - Beginning of unification of standards
 - Several release
 - R4: EDGE
 - R5: HSDPA – Technology between 1.2 and 14.0 Mbps, most common: 7.2 Mbps
 - R6: HSUPA – 5,76 Mbps
 - R7-R8: Dual cell HSDPA/HSUPA:
 - 3GPP2 variant
 - Further revisions of CDMA2000
- 3.x generation
 - Further releases

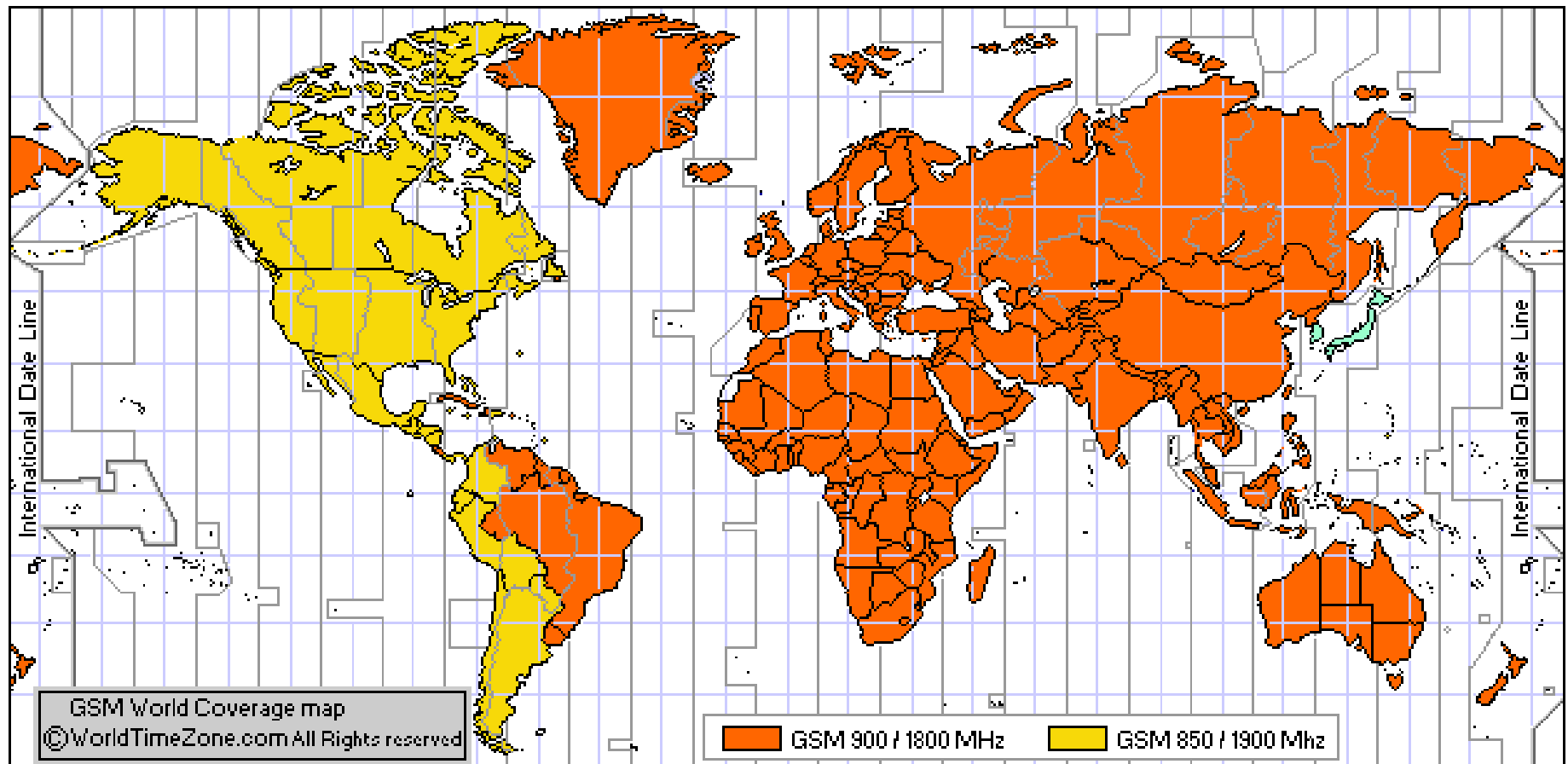
History of mobile services

- 3.9
 - LTE
- 4. generation (4G, 4.5G, 4.9G)
 - LTE – Advanced – All-in-one
 - Several Release and Category, different frequency band in different countries
- 5. generation

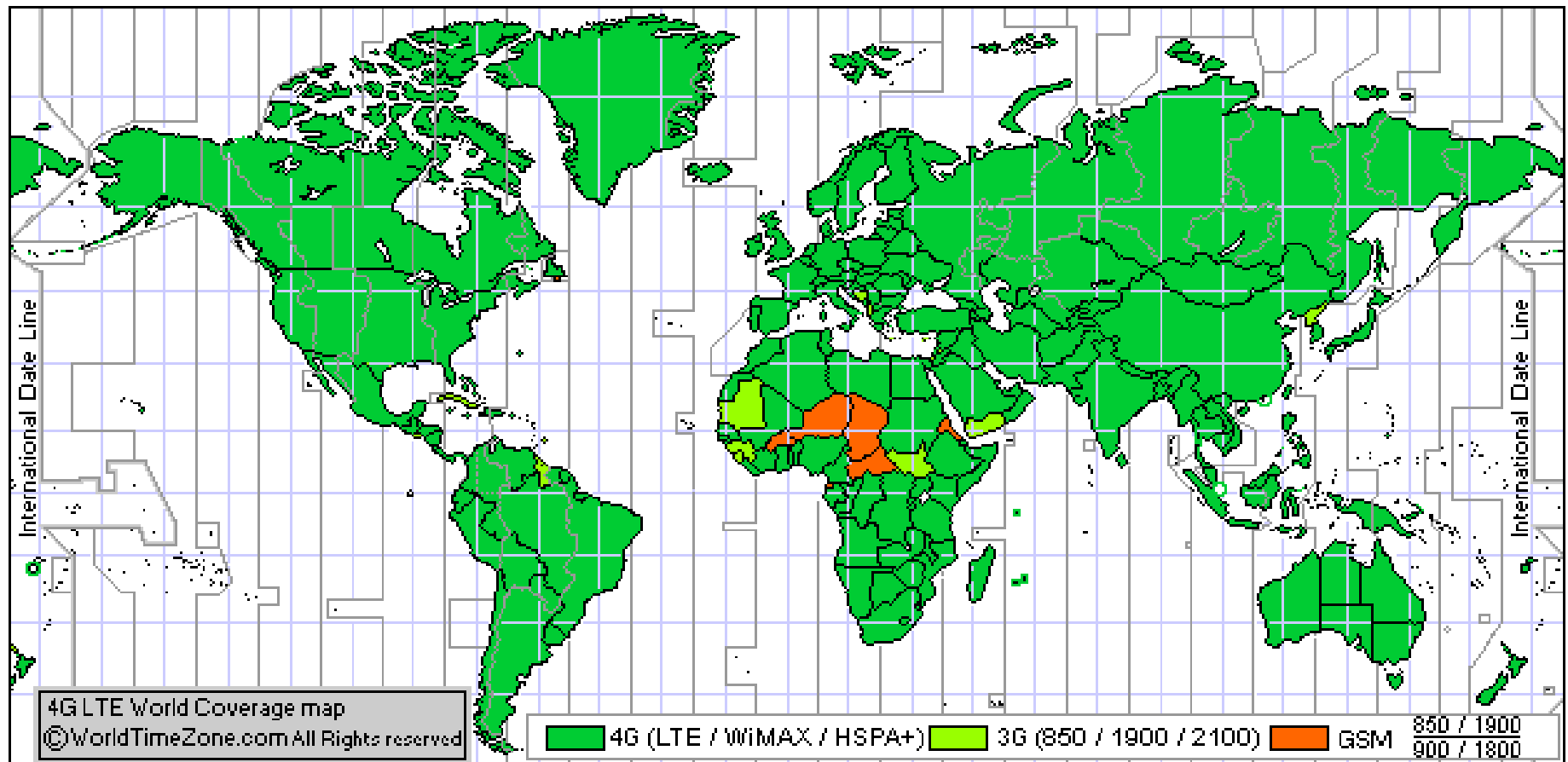
VoLTE

- Voice over LTE (VoLTE) is a standard for high-speed wireless communication for mobile phones and data terminals
- VoLTE has up to three times more voice and data capacity than 3G UMTS and up to six times more than 2G GSM.

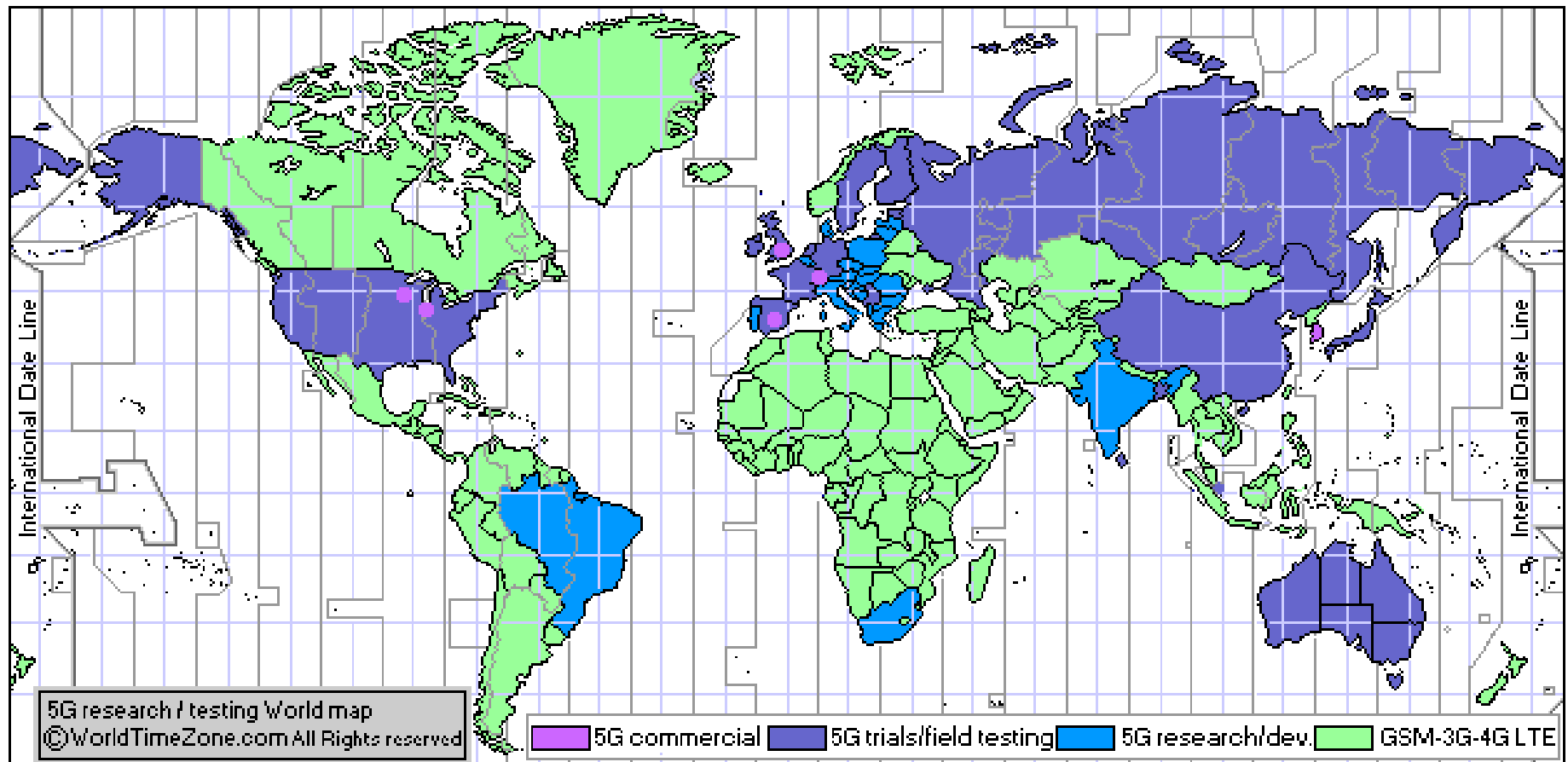
GSM



4G LTE



5G



LTE

1	South Korea	97.5%
2	Japan	96.3%
3	Norway	95.5%
4	Hong Kong	94.1%
5	United States	93%
6	Netherlands	92.8%
7	Taiwan	92.8%
8	Hungary	91.4%
9	Sweden	91.1%
10	India	90.9%

LTE / LTE Advanced

Category 0	1.0 Mbit/s	1	1.0 Mbit/s	Release 12
Category 1	10.3 Mbit/s	1	5.2 Mbit/s	Release 8
Category 2	51.0 Mbit/s	2	25.5 Mbit/s	Release 8
Category 3	102.0 Mbit/s	2	51.0 Mbit/s	Release 8
Category 4	150.8 Mbit/s	2	51.0 Mbit/s	Release 8
Category 5	299.6 Mbit/s	4	75.4 Mbit/s	Release 8
Category 6	301.5 Mbit/s	2 or 4	51.0 Mbit/s	Release 10
Category 7	301.5 Mbit/s	2 or 4	102.0 Mbit/s	Release 10
Category 8	2,998.6 Mbit/s	8	1,497.8 Mbit/s	Release 10
Category 9	452.2 Mbit/s	2 or 4	51.0 Mbit/s	Release 11
Category 10	452.2 Mbit/s	2 or 4	102.0 Mbit/s	Release 11
Category 11	603.0 Mbit/s	2 or 4	51.0 Mbit/s	Release 11
Category 12	603.0 Mbit/s	2 or 4	102.0 Mbit/s	Release 11
Category 13	391.7 Mbit/s	2 or 4	150.8 Mbit/s	Release 12
Category 14	3,917 Mbit/s	8	N/A	Release 12
Category 15	750 Mbit/s	2 or 4	N/A	Release 12
Category 16	979 Mbit/s	2 or 4	N/A	Release 12

44 LTE channel

- **Networks on LTE-bands 1, 3, 7, 28 (FDD-LTE) or 38, 40 (TDD-LTE) are suitable for future global roaming in ITU Regions 1, 2 and 3.**
- Networks on LTE-band 8 (FDD-LTE) may allow global roaming in the future (ITU Regions 1, 2 and 3) (Long-term perspective).
- Networks on LTE-band 20 (FDD-LTE) are suitable for roaming in ITU Region 1 (EMEA) only.
- Networks on LTE-bands 2 and 4 (FDD-LTE) are suitable for roaming in ITU Region 2 (Americas) only.



NFC

NFC

- Short distance communication standards
 - Communication between mobile devices
 - Touch to start communication
- NFC communication with passive „tags“.
 - Passive side has no power source
 - The information can be read by an active device
- The spread of NFC enabled devices is slow (iPhone 6)
 - Other methods: NFC integrated into SIM cards
- Wireless payment methods (credit cards) are also based on NFC

NFC vs. Bluetooth

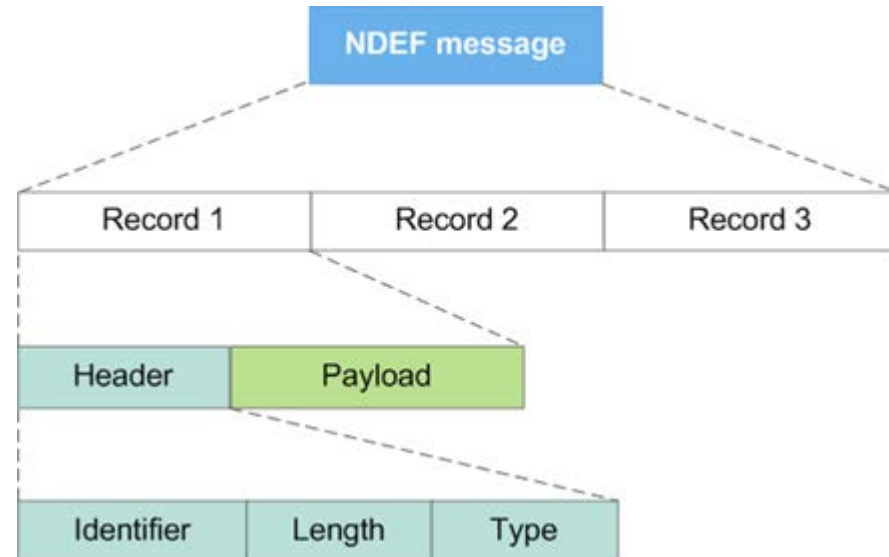
	NFC	Bluetooth	Bluetooth Low Energy
Tag requires power	No	Yes	Yes
Cost of Tag	10c	\$5	\$5
RFID compatible	ISO 18000-3	active	active
Standardisation body	ISO/IEC	Bluetooth SIG	Bluetooth SIG
Network Standard	ISO 13157 etc.	IEEE 802.15.1	IEEE 802.15.1
Network Type	Point-to-point	WPAN	WPAN
Cryptography	not with RFID	available	available
Range	< 0.2 m	~100 m (class 1)	~50 m
Frequency	13.56 MHz	2.4–2.5 GHz	2.4–2.5 GHz
Bit rate	424 kbit/s	2.1 Mbit/s	1 Mbit/s
Set-up time	< 0.1 s	< 6 s	< 0.006 s
Current consumption	< 15mA (read)	varies with class	< 15 mA (read and transmit)

NFC modes

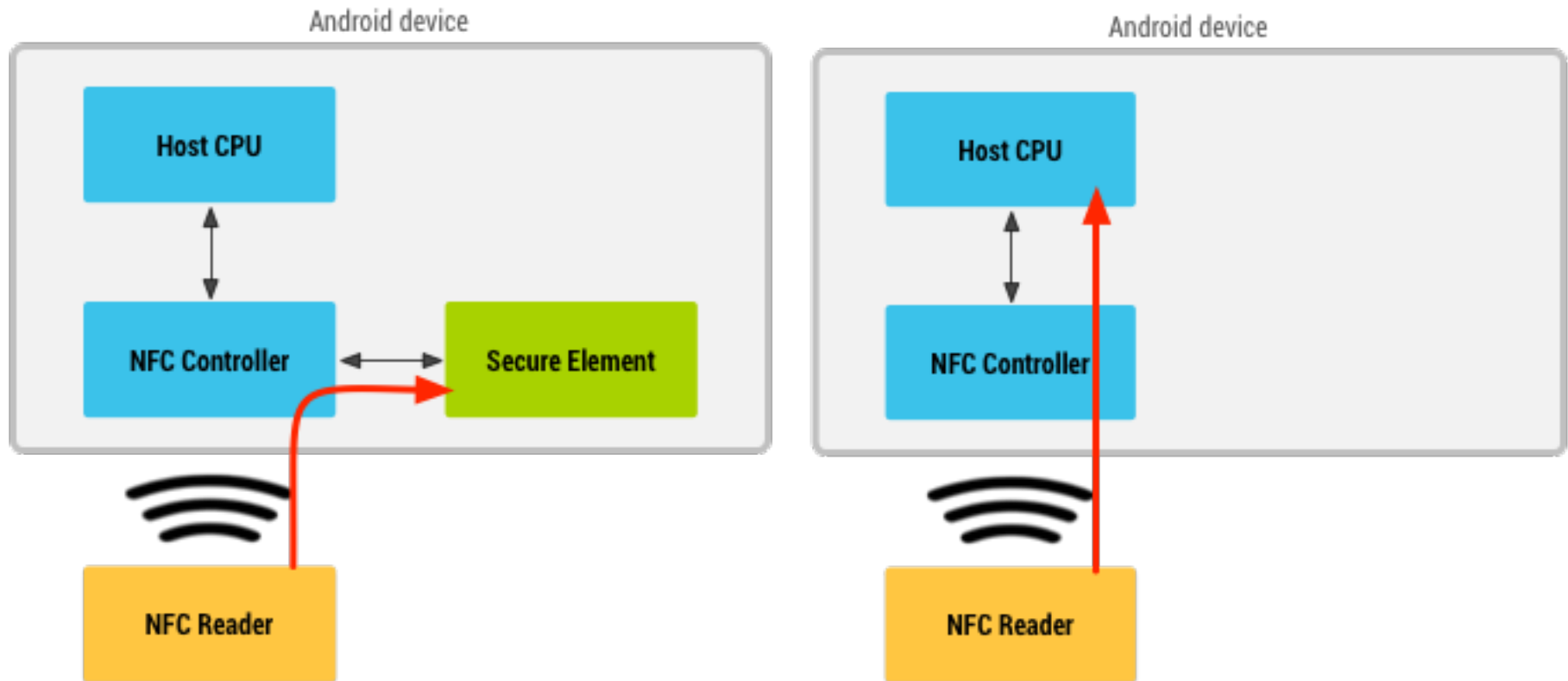
- **Reader/writer mode**, allowing the NFC device to read and/or write passive NFC tags and stickers.
- **P2P mode**, allowing the NFC device to exchange data with other NFC peers
 - This operation mode is used by Android Beam.
- **Card emulation mode**, allowing the NFC device itself to act as an NFC card.
 - The emulated NFC card can then be accessed by an external NFC reader, such as an NFC point-of-sale terminal.

NFC data structure

- NDEF message
- ID = TNF (3 bit)
 - URI: 0x0003
 - Empty: 0x0000
 - External: 0x0004
 - MIME: 0x0002
 - Well Known RTD 0x0001
 - Unknown 0x0005
 - Unchanged 0x0006
- RTD
 - URI, TEXT, Smart Poster, ...



Card Emulation





Additional devices

Apple

- iPod, iPad
 - Running the same system
 - Different screen size
 - Some of the features are missing
- Watch
 - WatchOS
 - Gen 4:
 - Cellular, Wifi, Bluetooth, GPS/GLONASS, Storage, etc.
- AppleTV

Google

- Auto
 - „External Display” for Android
 - App support
- Watch
 - Basically the same OS
 - Some of the functions are missing
 - Depending on the manufacturer
- Embedded
 - FireTV
 - Based on the open Android source
 - Other (TV) devices
- ChromeCast



Review of basic programming techniques and definitions

Next week



Pázmány Péter Catholic University
Faculty of Information Technology and Bionics

Basics of Mobile Application Development

OOP and basics of programming techniques



OOP

Concepts and practice

Modelling the world

- Principles
 - Abstraction
 - The properties and parts of the real world are simplified, thus only the essential parts are considered in order to reach the objective which has been set.
 - One abstracts from the unimportant properties and information and the important details are highlighted
 - Differentiation
 - Objects are the entities of the world which have to be modeled.
 - Objects are differentiated based on their important properties and behavior.

Modelling the world

- Principles
 - Classification
 - Object are assorted into categories, classes. Objects with similar properties belong to the same class, and objects with different properties are in different classes.
 - The classes bear the characteristics of the objects in the class. They can be considered as the templates of the objects.
 - Generalization, specialization
 - Similarities and differences are sought in order to create general or special categories and classes.

OOP – principles

- OOP principles (Benjamin C. Pierce)
 - Dynamic binding
 - In case of an object, if there are several implementations of a method, the executed one is selected runtime, dynamically.
 - Encapsulation
 - Data and operations are considered as a single unit
 - Practically it is consistent with the definition of type
 - Subtype polymorphism
 - A typed variable can refer to objects with different (other) subtypes
 - Subtype
 - A type created by specializing an existing one
 - Inheritance, or delegation
 - It is possible to create a new class using an existing one
 - It has the properties of the original class
 - And it also can extend, augment and modify the original class
 - Open recursion
 - Special variable, which enables a method to access the current instance of the class

OOP – keywords

- Object
 - Represents of entities of the real world
- Class of objects
 - Group of similar objects
 - Behavior
 - Structure
 - Template to create objects
- Method
 - A function (procedure) which manipulates the state of an object
- Field
 - A variable defining a property of an object
- Messaging
 - Interaction of objects
 - Interfaces are defined to facilitate the communication of objects
- Abstraction
 - Grouping classes
- Hierarchy
 - Design and implementation tool

OOP – Object

- Object
 - Its state is defined, information is stored
 - Perform tasks, its state can be changed
 - Communicate with another object by messaging
 - Can be identified unambiguously
- Lifecycle
 - Is born – construction and initialization
 - Initial values
 - Tasks executed for initialize
 - Setting the type invariant
 - Exists – operational phase
 - Dies – destruction
 - Freeing resources

OOP – fields and methods

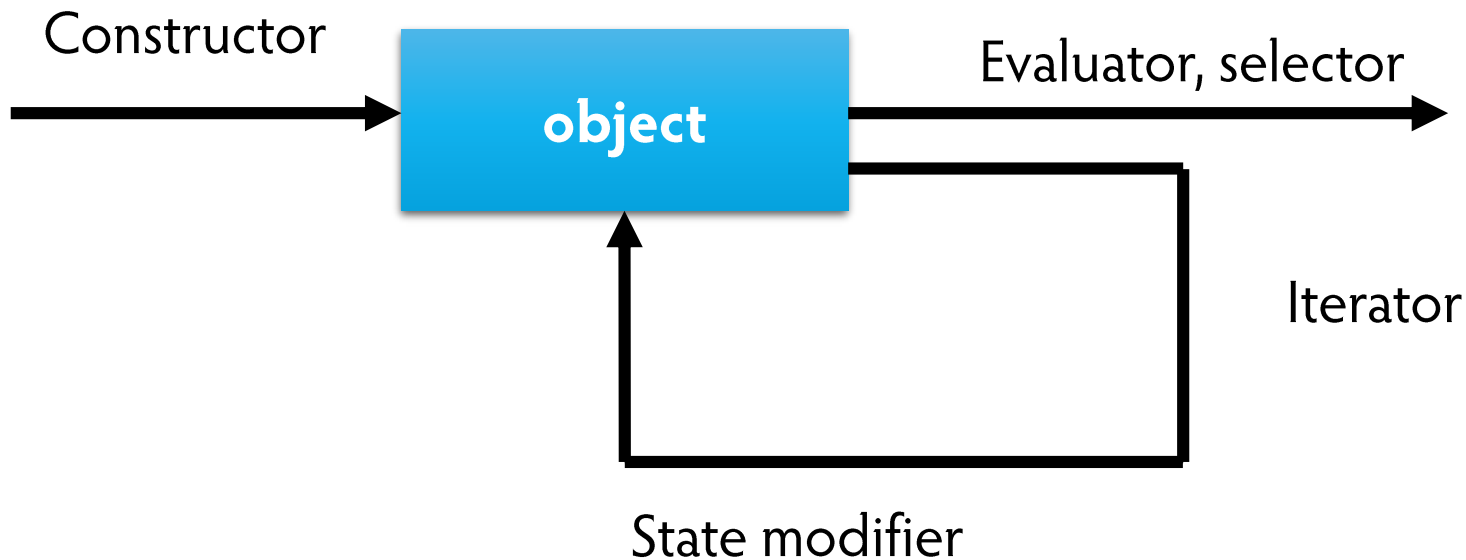
- Class definition
 - Instance variable
 - Separate instance exists for different objects
 - Instance method
 - Works on the state of an instance
 - Class variable
 - Variable for a class
 - Class method
 - Works on the state of the class

Operations of objects

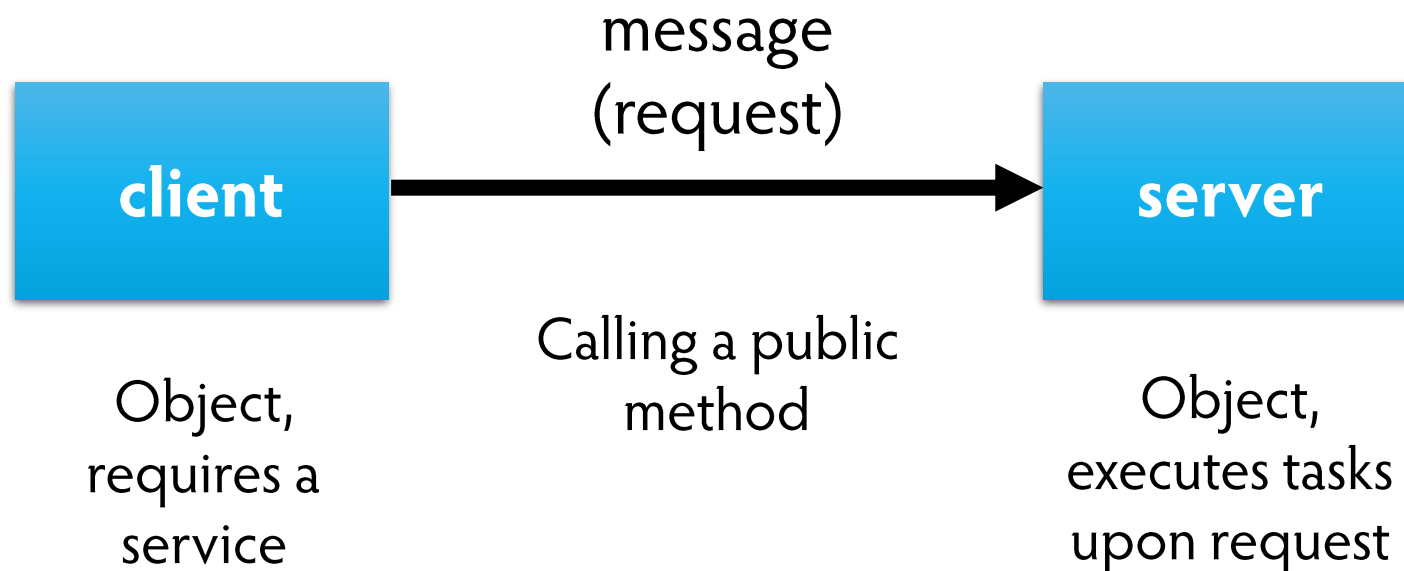
- Export operations
 - Called by other objects
- Import operations
 - Called by the object to provide its defined service
- Operations can be sorted as follows
 - Constructor: to create an object
 - State modifier
 - Selector: to select (except) a part of an object
 - Evaluator: to query features of an object
 - Iterator: to discover (or roam)

Operations of objects

- Export operations



Client sends a message



Client sends message to server

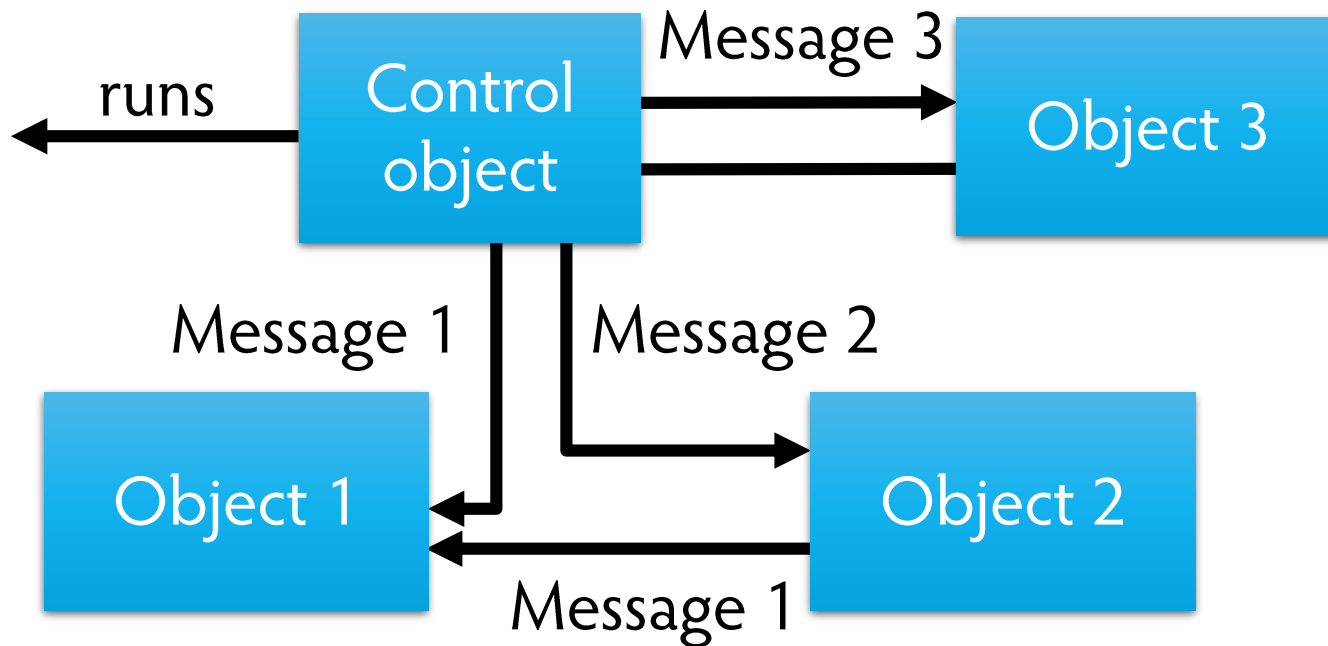
- Client
 - Active object, perform operations on other objects, but operations are not performed on it
 - Do not have any export interface
 - E.g.: Clock – performs an operation regularly
- Server
 - Passive object has only export interface
 - Waits for messages, do not require services
 - Do not have any import interface
- Agent
 - General object, both with import and export interface

this

- Separate memory allocation is made for instance variables
 - Instance methods are working in this state-space
 - However, instance methods are stored only once
 - So, how do we know the which one is the actual instance which is calling a method?
 - We need a pointer which refers to the actual instance, in any of the methods
 - The parameter „**this**” is used for this purpose
 - It means that in case of an object wants to send a message to itself; then it should call **this.message(parameters)** form.
 - So, in all methods, the reference for the actual instance should use this variable.
 - However, it is the default in several languages

OO program

- An OOP program is a set of communicating objects.
- Each object has its purpose, authority and scope of duties.



OOP – operational expectations

- Encapsulation
 - Data and operation performed on them considered a single unit
- Information hiding
 - The „private matters” of an object can only be accessed through methods
 - In case of some language this mechanism can be bypassed (not recommended)
- Code reuse
 - The code can be used to create
 - New instance with additional functions
 - New class with new, additional function, or to change existing behavior

OOP – Inheritance

- Creating a new class based on an existing one
 - The existing methods and fields are used to create new functionality
 - Augment (extend) a class
 - Override existing methods and field, according to the new function
- Design level step
 - Creating a subtype
 - IS-A type relation
 - One can create a HAS-A relation, however, it is not subtype
- Code reuse

OOP (and not OOP) – Definition

- Overload
 - Two methods with the same name, but different signature
 - The number or type of the parameters are different
 - `int add(int a, int b)`
 - `double add(double a, double b)`
- Override
 - Derived class has a method with the same name and signature
 - If and only if dynamic binding occurs as well, otherwise it is hiding
 - Some of the languages, you have to use a keyword.
 - Objective Pascal: **virtual**, **override**
- Hiding
 - Derived class has a method with the same name and signature
 - But there is no dynamic binding
 - Fields and variables can hide each other in a block as well as in a child class

Abstract class

- Design tool
 - To generalize
 - Subtype relation often requires creating abstract classes
- Abstract class: incomplete class
 - There are functions which implementations are not known in the class
 - The implementation is provided by one of the derived classes

Polymorphism, dynamic binding

- Polymorphism is the capability when a variable can refer to different type of objects
 - Now we consider only the subtype polymorphism
 - Type A is the subtype of type B if the following is true: type A can be used in all situations where type B can be used
 - Static type: defined at declaration
 - Dynamic type: the type the actual object referred by the variable
 - The dynamic type can be the static type of any of its derived classes.
 - The static type is permanent, and set in the source code, while dynamic type can vary in runtime
- Dynamic binding
 - The dynamic call is the event when we call a method of an object allowed by its static type, but the implementation executed which corresponds to the dynamic type.
 - Dynamic binding happens only when the derived class overrides the method (not hiding)



C++

Class definition

- **class** aclass : **public** parent_1, **public** parent_2 {
 // Fields
 private:
 int counter = 0;

 public:
 aclass();
 void add(**int** howmany);
 void print() **const**;
}
- **aclass::aclass**(**int** start) { counter = start; }
- **void** aclass::add(**int** howmany)
 {counter += howmany; }
- **void** aclass ::print **const** { cout << counter; }

Keywords

• Fields

- **public** – all objects have access (this, children, others)
- **protected** – this and children objects can access
- **private** – only this class can access (and friends)
- **const** – constant variable, its value cannot be changed
- **static** – class variable (can be accessed without instantiation)

• Methods

- **public protected private static**
- **const** – does not change the state of the class
- **void** – when there is no return value
- **virtual** – functions with dynamic binding (overriding instead of hiding)
- After signature = 0 ; – pure virtual, abstract function
- **throw** – throwable exception can be listed

Type and passing parameters

- In C++ regular variables and pointer as passed by value
 - Formal parameters are declared as local variables
 - They are initialized with the values of the actual parameters
 - Regular variable holds the value which is assigned
 - In case of a pointer, this is the memory address
 - Thus a copy is created of the original variable to an other part of the memory
 - As a result, the formal and actual parameters are in different places
- In C++ a reference formal parameter declares a (new) reference to the actual parameter
 - The original memory location has a new variable name
 - All of the changes performed through the formal parameter (local variable) effects the original memory location

Constant parameters

- It can be avoided that the called function can change the value of the original value
 - Then the parameter is constant
 - **const int & i**
 - You can pass large objects without copying them as well as you can prohibit any of changes
- It works with pointers as well
 - **void f(int * const p)**
 - The address is constant (as that is the value of the variable)
 - **void f(const int * p)**
 - You cannot change the referred memory
 - **void f(int const * p)** is equivalent
 - **void f(int const * const p)**
 - Both address and referred memory is constant

Inheritance

- The graph representation (directed graph) of the inheritance relations can be called as class hierarchy
- In C++ multiple inheritance exists.
 - Thus the graph is a general directed graph
- Inheritance can be
 - **public** – Visibility modifiers are unchanged, but **private** members cannot be accessed in derived classes
 - IS-A relation
 - **protected** – Public functions and fields will be **protected**
 - **private** – Public functions and fields will be **private**
- Default is **private**

Multiple inheritance

- A class can inherit (directly) from several other classes
- What happens if there are functions with the same name in different parents?
 - A decision must be made
 - Scope operator
 - `d.Base1::f()` ;
 - `d.Base2::f()` ;
- Diamond problem
 - Virtual inheritance
 - **`class C: public virtual Base`**

Constructor

- Constructor
 - Code, which is executed automatically when the class is instantiated
 - Its name is the same as the class, and it has no return value
 - It is similar to methods, but there are differences (it is not member, because it cannot be inherited)
 - All classes has constructors
 - If we do not define, the compiler creates
 - But only when there is no programmer defined constructor
 - Several constructors can exist, the can be overridden
 - The constructor of a derived class called after the constructor of the base class
 - In C++11 constructors can be delegated
 - Avoid cyclic call

Destructor

- Destructor
 - Destructor is a code which is responsible to free resources most of the cases
 - Prepare to die
 - Classes should have virtual destructors
 - If there is no chance to have a derived class it is not necessary

Friend

- A function can access to any fields of a class
 - Even private
 - Encapsulation can be violated
 - Keyword **friend** have to be used
 - A friend can access the private and protected parts of a class
 - Typical examples are the input/output stream operators (<<, >>)
friend std::ostream& **operator** <<
(std::ostream& stream, **const** Object& z);

delete and default functions

- Compiler creates a bunch of functions, if they are not defined manually
- Starting from C++11 this automatism can be controlled.
 - To create use **default**
 - To avoid use **delete**
- ```
class Car {
 public:
 Car() = default;
 Car(const _car&) = default;
 Car& operator=(const _car&) = delete;
 virtual ~C() = default;
};
```

# delete and default functions

- The class-level operators (&, \*, ->, new, delete) also can be controlled.
- **Class C {**  
    **public:**  
        **void \*operator new(size\_t) = delete;**  
        **void \*operator new[](size\_t) =**  
    **delete;**  
};
- **int main() {**  
    **C \*c = new C;**  
    **C \*t = new C[3];**  
    **C c;**  
    **C t[10];**  
}



# Assignment operator, copy constructor

- They can be used to copy an object
- Assignment operator is called when the variable is assigned to a new (existing) value
- Copy constructor called when the parameter is passed by value
- They are declared automatically
  - In case of dynamic memory allocation, the automatically generated assignment operator and copy constructor cause shallow copy
    - As only the pointer is copied not the referred memory
  - As a result they must be declared manually

# Assignment operator, copy constructor

```
class A {
 A (const A& _other);
 A& operator= (const A& _other);
 A (A&& _other); // move constructor
 A& operator= (A&& _other); // move operator
};

A e, f, g;
g(std::move(f));
e = std::move(g);
```

- [http://en.cppreference.com/w/cpp/language/move\\_constructor](http://en.cppreference.com/w/cpp/language/move_constructor)
- [http://en.cppreference.com/w/cpp/language/move\\_operator](http://en.cppreference.com/w/cpp/language/move_operator)

# User defined literals

- As of C++11 user can defined new literals

- ```
inline double operator"" _deg (long double degree) {  
    return degree * 3.14159265 / 180.0;  
}
```
- ```
double rad = 90.0_deg; // degree = 1.570796325
```
- ```
unsigned operator"" _Magic(const char* _magic) {  
    unsigned b = 0;  
    for(unsigned int i = 0; _magic[i]; ++i)  
        b = b*2 + (_magic[i] == '1');  
    return b;  
}
```
- ```
int mask = 110011_Magic; // mask = 51
```



Pázmány Péter Catholic University  
Faculty of Information Technology and Bionics

# Java

# Class definition

```
public class AClass extends Parent implements Interface
{
 private int counter;

 public AClass()
 {
 counter = 0;
 }

 public void add(int howmany) throws MyException
 {
 this.counter += howmany;
 }

 public void print()
 {
 System.out.println(counter);
 }
}
```

# Keywords

- Class
  - **public** – everyone can access
  - **final** – cannot be derived
  - **abstract** – abstract class, with abstract method(s)
  - **extends** – to provide the parent class
  - **implements** – to list the implemented interfaces
- All of them are optional. The default visibility is package private – it means that the class is accessible in the package only.
- Fields
  - **public** – everyone can access (this class, derived class, inside and outside of the package)
  - **protected** – can be accessed in this class, derived classes and in the package
  - default visibility – package private, can be access inside of the package (this class, and ...)
  - **private** – only this class
  - **final** – its value cannot be changed, constant
  - **volatile** – always committed to the shared memory in case of threading
  - **static** – class variable (can be accessed without instantiation)

# Keywords

- **final** – again
  - What is constant? The value:
    - In case of primitives it is the value
    - In case of Object it is the reference to the object
  - It has no effect on the fields of the referred object
- Methods
  - **public protected private abstract static**
  - **final** – cannot be overridden or hidden
  - **synchronized** – mutual exclusion can be achieved in threading
  - **void** – if there is no return value
  - **throws** – the throwable exception must be enumerated

# Visibility

| Originating object |         | Target object |           |                     |         |
|--------------------|---------|---------------|-----------|---------------------|---------|
| Package            | Class   | public        | protected | no modifier default | private |
| Same               | This    | X             | X         | X                   | X       |
|                    | Inner   | X             | X         | X                   | X       |
|                    | Derived | X             | X         | X                   |         |
|                    | Other   | X             | X         | X                   |         |
| Other              | Derived | X             | X         |                     |         |
|                    | Other   | X             |           |                     |         |



# Types, parameter passing

- In Java parameters always passed by value
  - Thus the value is copied
  - In case of reference type the value is the memory address, so the address is copied this the parameter passing seems „by reference”
    - However the wrapper counterparts are immutable, as a result they behave as the primitive ones
- Primitives:
  - byte, short, int, long, float, double, char, boolean
- Everything other is reference type, derived from Object class
- Primitive – Wrapper pairs
  - Byte, Short, Integer, Long, Float, Double, Character, Boolean
    - Immutable – new instance is created once it is changed
- Warning! String is immutable

# Inheritance

- The graph representation (directed graph) of the inheritance relations can be called as class hierarchy
- In Java there is an universal base class, the `Object`, everything is derived from `Object`
- In Java there is no multiple inheritance, thus the class hierarchy is represented by a tree
- Implicit extends `Object` in case of no manual extends given
- `Object`: pre defined in `java.lang`
  - There are methods required to exist in all objects

# Initialization of an object

- The order of execution
  - Static initialization block of the base class
  - Static initialization block of the derived class
  - Initialization block(s) of the base class
  - Constructor of the base class
  - Initialization block(s) of the derived class
  - Constructor of the derived class
- The constructor of the base class is executed before any constructor of the derived class
  - Even if it is not called explicitly
    - In that case the constructor with same signature is called
    - If it does not exist then an Exception is thrown
- The constructor without parameters is created by the compiler if and only if there is no other constructor is defined

# Initialization block

- Initialization block: Block of statements (instance level and class level as well) to initialize variables

```
public class A{
 static int i=10;
 static int ifact;
 static {
 ifact=1;
 for (int j=2; j<=10; j++){
 ifact*=j;
 }
 }
 int [] array = new int[10];
 {
 for (int i=0; i<10; i++){
 array[i]=(int)Math.random();
 }
 }
}
```

# Initialization block

- Class level: executed when the class is initialized, substitutes the class constructor (as they do not exist)
- Object level: executed when the object is instantiated, augmenting the constructors.
  - In case of anonymous classes, it substitutes the constructors
- There may be several initialization blocks in a class
- Execution order is the order of the definition (merged with the variable initializations)
  - Variables defined later cannot be referenced
- return statement is not allowed

# Java – implicit type conversion

- Subtypes
  - Subtypes can be handled as general types
  - So a derived class can be handled as its ancestors
- The object is not converted
  - `java.lang.ClassCastException` is raised when the type is not compatible
  - Use **`instanceof`** operator!

# Interfaces

- Interfaces can be implemented by classes
- Keyword
  - **public class** Apple **implements** Edible
  - If a class implements an interface then the interface can be used as static type
- Static type vs dynamic type:
  - Edible apple = **new** Apple();  
Edible pear = **new** Pear();
  - Supposing that
    - Apple **implements** Edible
    - Pear **implements** Edible
- Interfaces can be inherited
  - All the constants and member functions can be inherited from the superclass
    - Implementation not, but that cannot be there.
- Multiple inheritance is possible!
  - As interfaces are „abstract” there is no problem with the inheritance of implementation

# Modifiers

- Functions
  - Methods are always **public abstract**.
    - These keywords are optional
    - Using others is error
- Constants
  - All fields are **public static final**
  - Not required
- What are the differences between abstract class and interfaces?



# Homework – Deadline: 10/01/18 10.15 am

- Create a C++ program which demonstrates the proper and improper usage of friend
  - Read first!
  - Defining friend operators might be appropriate!
  - Violating the encapsulation and data hiding is inappropriate!
- Create a Java demonstration where it is a good option to use default interface methods.
  - Read first!
  - Also, demonstrate why it should be considered as the last resort!



Pázmány Péter Catholic University  
Faculty of Information Technology and Bionics

# Programming patterns

Next week



Pázmány Péter Catholic University  
Faculty of Information Technology and Bionics

# Basics of Mobile Application Development

Design patterns

# Design patterns

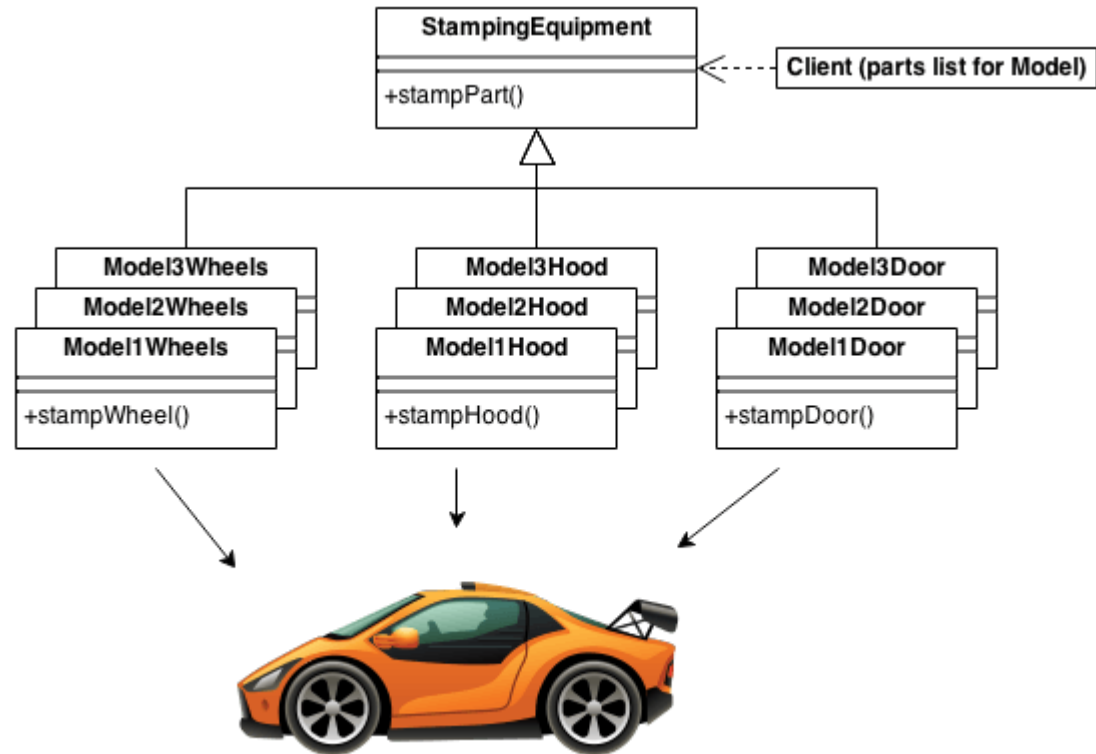
- Design patterns are description of collaborative objects and classes.
- General patterns can be specialized and implemented in order to solve software design issues.
  - It is beneficial to use object-oriented programming languages (C++, Java, Smalltalk, ...)
  - In case of procedural languages further patterns may be required:
    - Inheritance
    - Encapsulation
    - Polymorphism
- Categories
  - Objective
    - Creational patterns
    - Structural patterns
    - Behavioral patterns
  - Scope
    - Class
    - Object

# Some of patterns

- Creational
  - Abstract Factory
  - Builder
  - Factory Method
  - Prototype
  - Singleton
- Structural
  - Adapter
  - Bridge
  - Composite
  - Decorator
  - Facade
  - Flyweight
  - Proxy
- Behavioral
  - Chain of Responsibility
  - *Command*
  - Interpreter
  - *Iterator*
  - Mediator
  - Memento
  - Observer
  - State
  - Strategy
  - Template Method
  - Visitor

# Abstract Factory

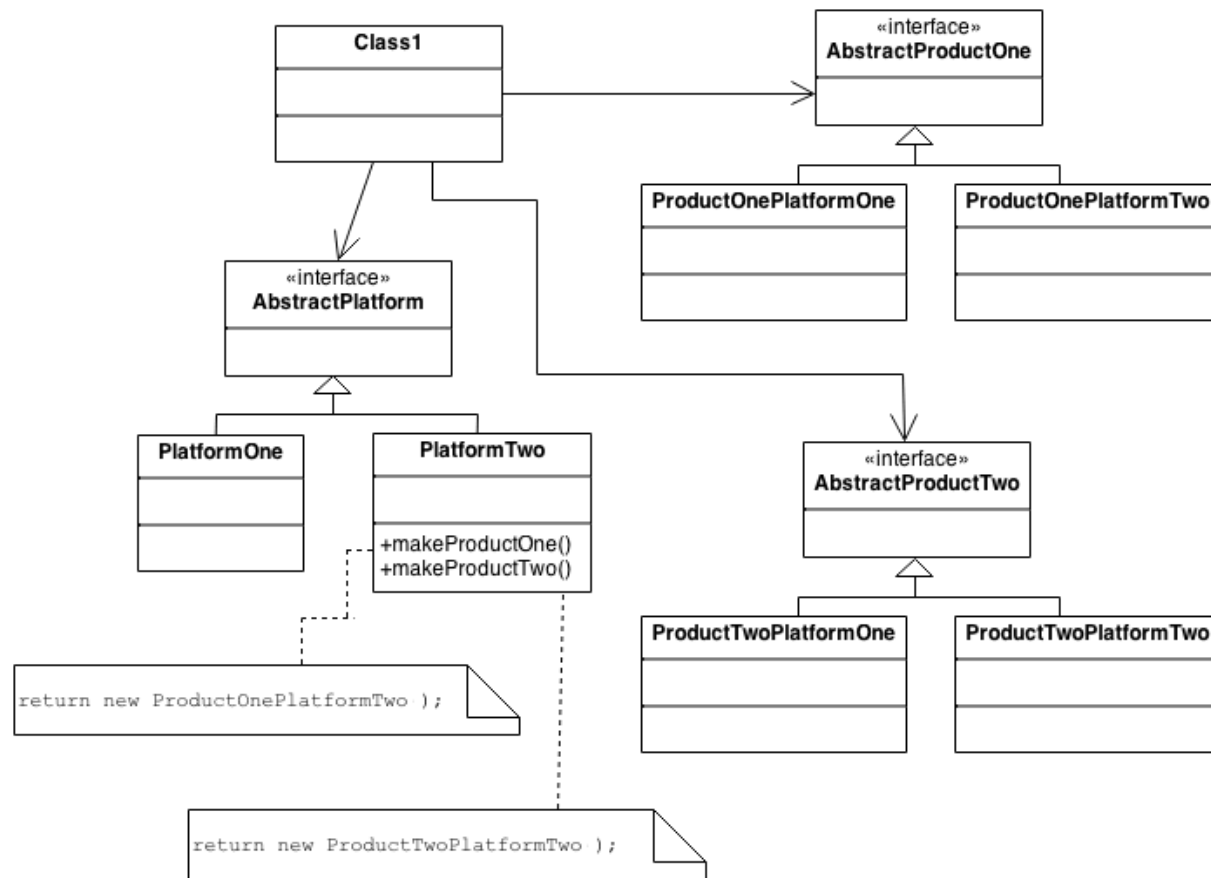
- Creational Pattern
  - Provide an interface for creating families of related or dependent objects without specifying their concrete classes.
  - A hierarchy that encapsulates: many possible "platforms", and the construction of a suite of "products".



# Abstract Factory

- Provide a level of indirection that abstracts the creation of families of related or dependent objects without directly specifying their concrete classes.
  - The "factory" object has the responsibility for providing creation services for the entire platform family.
  - Clients never create platform objects directly, they ask the factory to do that for them.

# Abstract Factory





# Abstract Factory

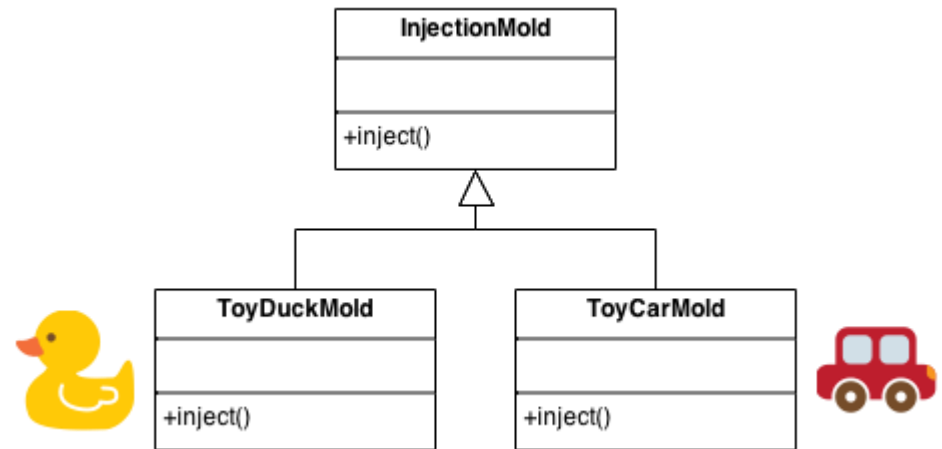
```
abstract class CPU {}
class EmberCPU extends CPU {}
class EnginolaCPU extends CPU {}
abstract class MMU {}
class EmberMMU extends MMU {}
class EnginolaMMU extends MMU {}
class EmberToolkit extends AbstractFactory {
 @Override
 public CPU createCPU() {
 return new EmberCPU();
 }
 @Override
 public MMU createMMU() {
 return new EmberMMU();
 }
}
class EnginolaToolkit extends AbstractFactory {
 @Override
 public CPU createCPU() {
 return new EnginolaCPU();
 }
 @Override
 public MMU createMMU() {
 return new EnginolaMMU();
 }
}
enum Architecture {
 ENGINOLA, EMBER
}
```

```
abstract class AbstractFactory {
 private static final EmberToolkit EMBER_TOOLKIT = new
 EmberToolkit();
 private static final EnginolaToolkit ENGINOLA_TOOLKIT = new
 EnginolaToolkit();
 static AbstractFactory getFactory(Architecture architecture) {
 AbstractFactory factory = null;
 switch (architecture) {
 case ENGINOLA:
 factory = ENGINOLA_TOOLKIT;
 break;
 case EMBER:
 factory = EMBER_TOOLKIT;
 break;
 }
 return factory;
 }
 public abstract CPU createCPU();
 public abstract MMU createMMU();
}
public class Client {
 public static void main(String[] args) {
 AbstractFactory factory =
 AbstractFactory.getFactory(Architecture.EMBER);
 CPU cpu = factory.createCPU();
 }
}
```

# Factory Method

- Creational pattern

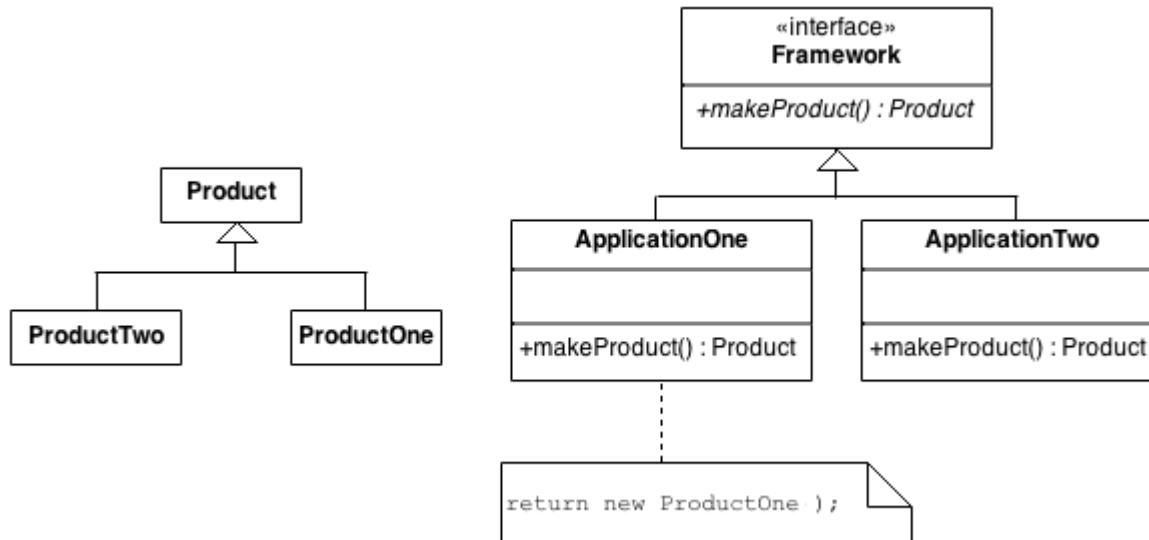
- Define an interface for creating an object, but let subclasses decide which class to instantiate. Factory Method lets a class defer instantiation to subclasses.
- Defining a "virtual" constructor.



# Factory Method

- A framework needs to standardize the architectural model for a range of applications
  - But allow for individual applications to define their own domain objects and provide for their instantiation.
- Factory Method is to creating objects as Template Method is to implementing an algorithm.
  - A superclass specifies all standard and generic behavior (using pure virtual "placeholders" for creation steps)
  - Then delegates the creation details to subclasses that are supplied by the client.

# Factory Method



# Factory Method

```
interface ImageReader {
 DecodedImage getDecodeImage();
}

class DecodedImage {
 private String image;

 public DecodedImage(String image) {
 this.image = image;
 }

 @Override
 public String toString() {
 return image + ": is decoded";
 }
}

class GifReader implements ImageReader {
 private DecodedImage decodedImage;

 public GifReader(String image) {
 this.decodedImage = new DecodedImage(image);
 }

 @Override
 public DecodedImage getDecodeImage() {
 return decodedImage;
 }
}
```

```
class JpegReader implements ImageReader {
 private DecodedImage decodedImage;

 public JpegReader(String image) {
 decodedImage = new DecodedImage(image);
 }

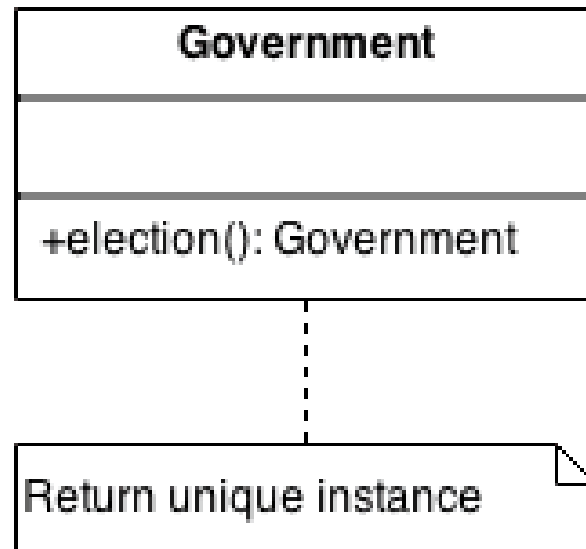
 @Override
 public DecodedImage getDecodeImage() {
 return decodedImage;
 }
}

public class FactoryMethodDemo {
 public static void main(String[] args) {
 DecodedImage decodedImage;
 ImageReader reader = null;
 String image = args[0];
 String format = image.substring(image.indexOf('.') + 1,
 (image.length()));
 if (format.equals("gif")) {
 reader = new GifReader(image);
 }
 if (format.equals("jpeg")) {
 reader = new JpegReader(image);
 }
 assert reader != null;
 decodedImage = reader.getDecodeImage();
 System.out.println(decodedImage);
 }
}
```

# Singleton

- Creational pattern
- Prohibits to create more than one instance of a class
  - Allows to create one and provides an access point to it
- Example
  - Only a single file system manager, or window manager is allowed
- Implementation
  - A hidden class function can only instantiate the class, ensuring that only one instance is allowed
  - Subclasses may be created

# Singleton



# Singleton

```
#include <iostream>
#include <string>
#include <stdlib.h>
using namespace std;

class Number
{
public:
 // 2. Define a public static accessor func
 static Number *instance();
 static void setType(string t)
 {
 type = t;
 delete inst;
 inst = 0;
 }
 virtual void setValue(int in)
 {
 value = in;
 }
 virtual int getValue()
 {
 return value;
 }
protected:
 int value;
 // 4. Define all ctors to be protected
 Number()
 {
 cout << ":ctor: ";
 }
 // 1. Define a private static attribute
private:
 static string type;
 static Number *inst;
};

string Number::type = "decimal";
Number *Number::inst = 0;
```

```
class Octal: public Number
{
 // 6. Inheritance can be supported
public:
 friend class Number;
 void setValue(int in)
 {
 char buf[10];
 sprintf(buf, "%o", in);
 sscanf(buf, "%d", &value);
 }
protected:
 Octal(){}
};

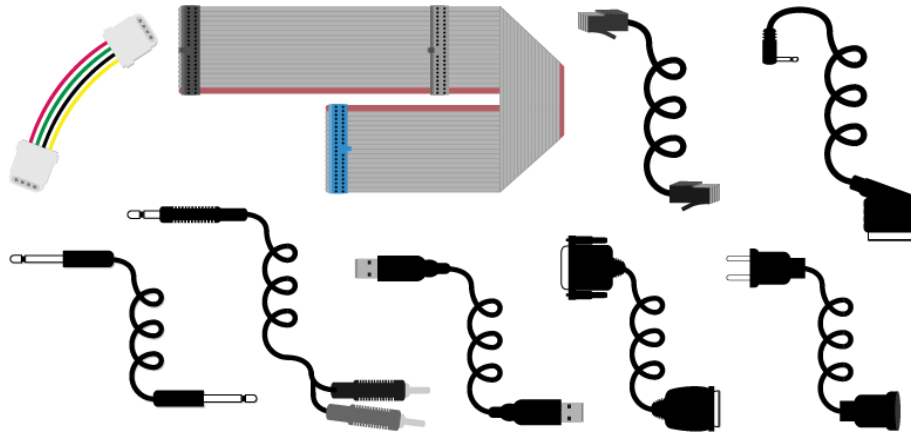
Number *Number::instance()
{
 if (!inst)
 // 3. Do "lazy initialization" in the accessor function
 if (type == "octal")
 inst = new Octal();
 else
 inst = new Number();
 return inst;
}

int main()
{
 // Number myInstance; - error: cannot access protected constructor
 // 5. Clients may only use the accessor function to manipulate the
 Singleton
 Number::instance()->setValue(42);
 cout << "value is " << Number::instance()->getValue() << endl;
 Number::setType("octal");
 Number::instance()->setValue(64);
 cout << "value is " << Number::instance()->getValue() << endl;
}
```



# Adapter

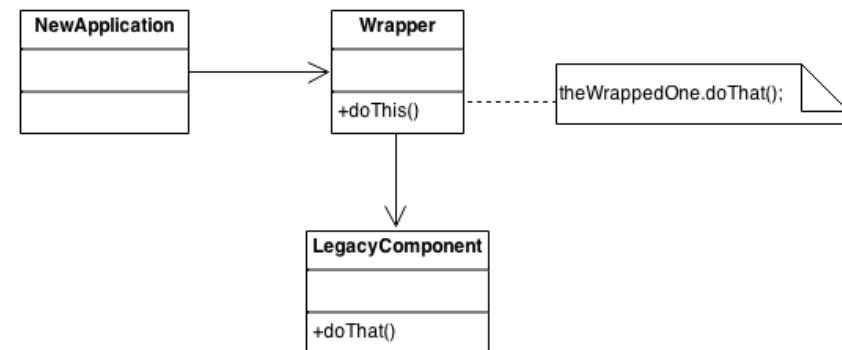
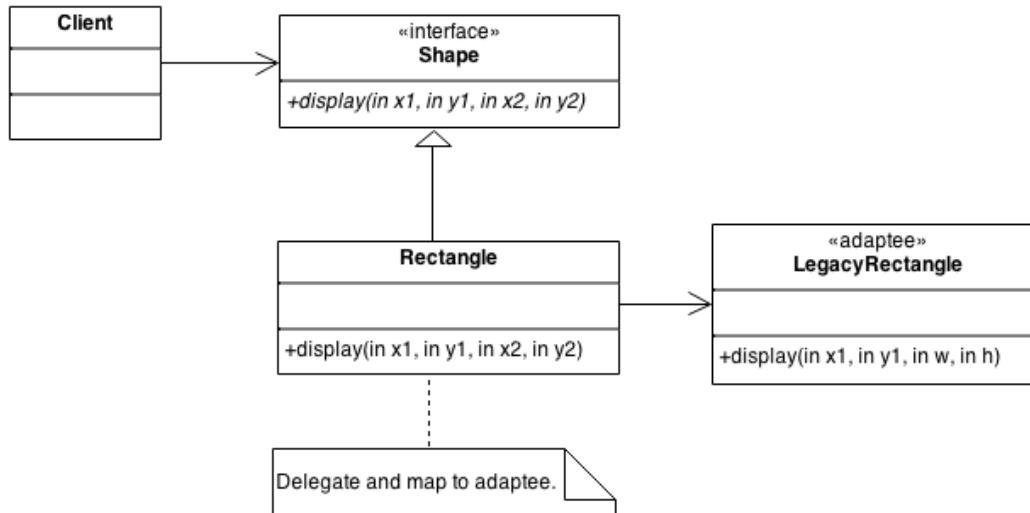
- Structural class/object pattern
  - An "off the shelf" component offers compelling functionality that you would like to reuse
  - But its "view of the world" is not compatible with the philosophy and architecture of the system currently being developed.



# Adapter

- Reuse has always been painful and elusive.
  - One reason has been the tribulation of designing something new, while reusing something old.
  - There is always something not quite right between the old and the new.
  - It may be physical dimensions or misalignment. It may be timing or synchronization.
  - It may be unfortunate assumptions or competing standards.

# Adapter



# Adapter

```
import abc

class Target(metaclass=abc.ABCMeta):
 """
 Define the domain-specific interface that Client uses.
 """
 def __init__(self):
 self._adaptee = Adaptee()

 @abc.abstractmethod
 def request(self):
 pass

class Adapter(Target):
 """
 Adapt the interface of Adaptee to the Target
 interface.
 """
 def request(self):
 self._adaptee.specific_request()
```

```
class Adaptee:
 """
 Define an existing interface that needs adapting.
 """
 def specific_request(self):
 pass

def main():
 adapter = Adapter()
 adapter.request()

if __name__ == "__main__":
 main()
```

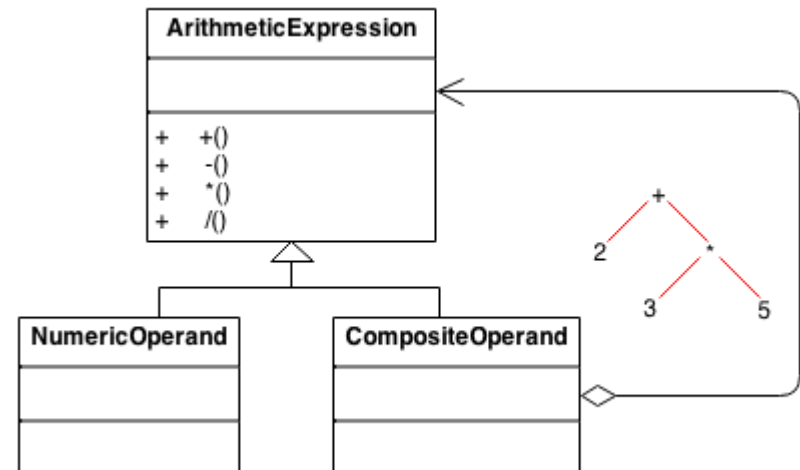
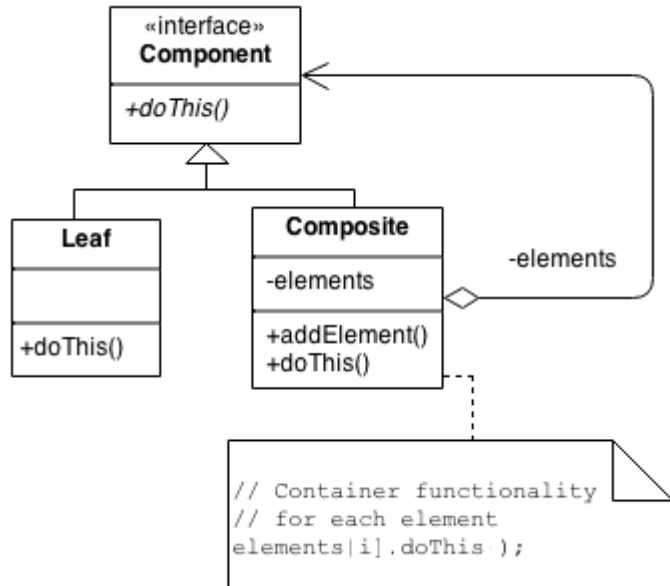
# Composition

- Structural pattern
  - Application needs to manipulate a hierarchical collection of "primitive" and "composite" objects.
    - Processing of a primitive object is handled one way, and processing of a composite object is handled differently.
    - Having to query the "type" of each object before attempting to process it is not desirable.
  - For example in a graphical application, the basic and complex view components can be treated with the same functions
    - Composite elements can be created by grouping objects

# Composition

- Define an abstract base class (Component) that specifies the behavior that needs to be exercised uniformly across all primitive and composite objects.
  - Subclass the Primitive and Composite classes off of the Component class.
  - Each Composite object "couples" itself only to the abstract type Component as it manages its "children".
- Use this pattern whenever you have "composites that contain components, each of which could be a composite".

# Composition



```
#include <iostream>
#include <vector>
using namespace std;

class Component
{
public:
 virtual void traverse() = 0;
};

class Primitive: public Component
{
 int value;
public:
 Primitive(int val)
 {
 value = val;
 }
 void traverse()
 {
 cout << value << " ";
 }
};

class Composite: public Component
{
 vector < Component * > children;
 int value;
public:
 Composite(int val)
 {
 value = val;
 }
 void add(Component *c)
 {
 children.push_back(c);
 }
 void traverse()
 {
 cout << value << " ";
 for (int i = 0; i < children.size(); i++)
 children[i]->traverse();
 }
};
```

```
class Row: public Composite
{
public:
 // Two different kinds of "con-
 Row(int val): Composite(val){}
 // tainer" classes. Most of the
 void traverse()
 {
 // "meat" is in the Composite
 cout << "Row"; // base class.
 Composite::traverse();
 }
};

class Column: public Composite
{
public:
 Column(int val): Composite(val){}
 void traverse()
 {
 cout << "Col";
 Composite::traverse();
 }
};

int main()
{
 Row first(1); // Row1
 Column second(2); //
 Column third(3); // --- Col2
 Row fourth(4); //
 Row fifth(5); //
 first.add(&second); // --- Col3
 first.add(&third); //
 third.add(&fourth); // --- Row4
 third.add(&fifth); //
 first.add(&Primitive(6)); //
 second.add(&Primitive(7)); //
 third.add(&Primitive(8)); //
 fourth.add(&Primitive(9)); //
 fifth.add(&Primitive(10)); //
 first.traverse(); // --- 6
 cout << '\n';
}
```

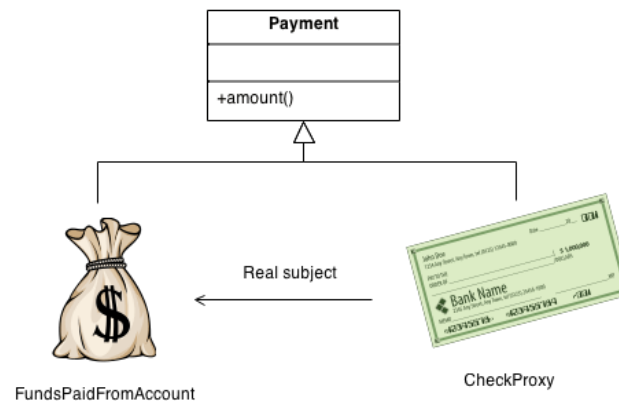
```

// Row1
//
// --- Col2
//
// --- 7
// --- Col3
// --- Row4
// --- 9
// --- Row5
// --- 10
// --- 8
// --- 6
```



# Proxy

- Structural pattern
- To control an object through another object
- You need to support resource-hungry objects, and you do not want to instantiate such objects unless and until they are actually requested by the client.



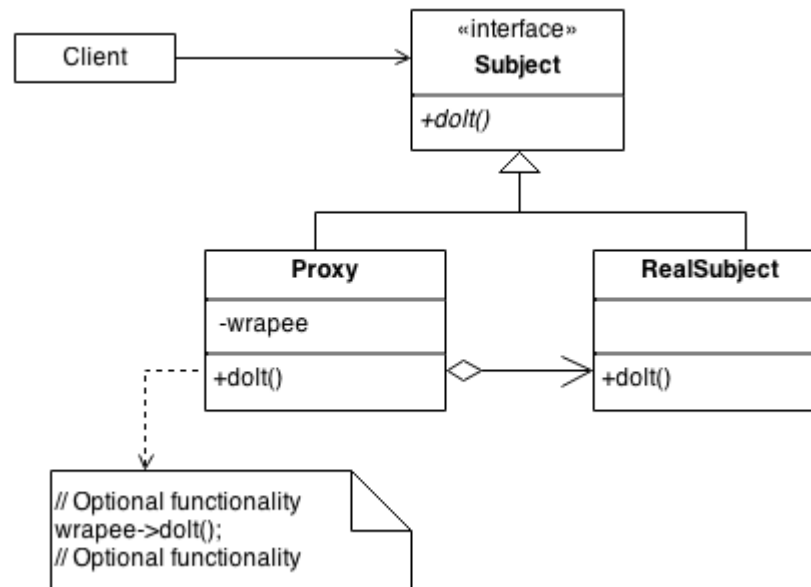
# Proxy

- Design a surrogate, or proxy, object that:
  - instantiates the real object the first time the client makes a request of the proxy
  - remembers the identity of this real object
  - forwards the instigating request to this real object
- Then all subsequent requests are simply forwarded directly to the encapsulated real object.

# Proxy

- There are four common situations in which the Proxy pattern is applicable.
  - A virtual proxy is a placeholder for "expensive to create" objects. The real object is only created when a client first requests/accesses the object.
  - A remote proxy provides a local representative for an object that resides in a different address space. This is what the "stub" code in RPC and CORBA provides.
  - A protective proxy controls access to a sensitive master object. The "surrogate" object checks that the caller has the access permissions required prior to forwarding the request.
  - A smart proxy interposes additional actions when an object is accessed. Typical uses include:
    - Counting the number of references to the real object so that it can be freed automatically when there are no more references (aka smart pointer),
    - Loading a persistent object into memory when it's first referenced,
    - Checking that the real object is locked before it is accessed to ensure that no other object can change it.

# Proxy



# Proxy

```
interface SocketInterface {
 String readLine();
 void writeLine(String str);
 void dispose();
}

class SocketProxy implements SocketInterface {
 private Socket socket;
 private BufferedReader in;
 private PrintWriter out;

 public SocketProxy(String host, int port, boolean wait) {
 try {
 if (wait) {
 // 2. Encapsulate the complexity/overhead of the target in the wrapper
 ServerSocket server = new ServerSocket(port);
 socket = server.accept();
 } else {
 socket = new Socket(host, port);
 }
 in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
 out = new PrintWriter(socket.getOutputStream(), true);
 } catch (IOException e) {
 e.printStackTrace();
 }
 }

 public String readLine() {
 String str = null;
 try {
 str = in.readLine();
 } catch (IOException e) {
 e.printStackTrace();
 }
 return str;
 }
}
```

```
public void writeLine(String str) {
 // 4. The wrapper delegates to the target
 out.println(str);
}

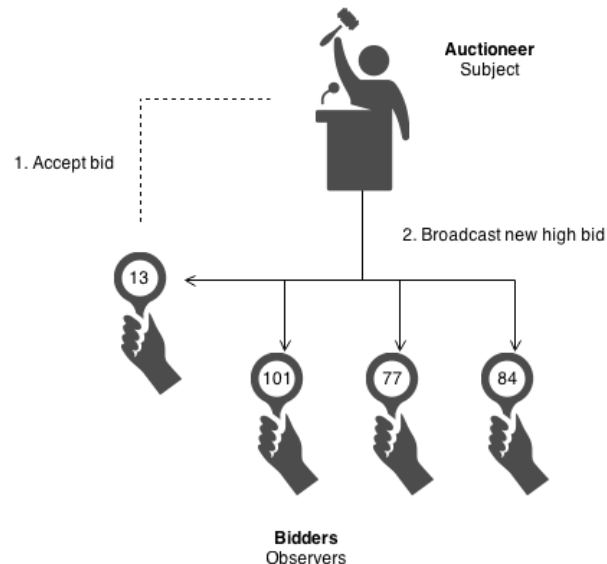
public void dispose() {
 try {
 socket.close();
 } catch(IOException e) {
 e.printStackTrace();
 }
}

}

public class ProxyDemo {
 public static void main(String[] args) {
 // 3. The client deals with the wrapper
 SocketInterface socket = new SocketProxy("127.0.0.1", 8080, args[0].equals("first") ? true : false);
 String str;
 boolean skip = true;
 while (true) {
 if (args[0].equals("second") && skip) {
 skip = !skip;
 } else {
 str = socket.readLine();
 System.out.println("Receive - " + str);
 if (str.equals(null)) {
 break;
 }
 }
 System.out.print("Send ---- ");
 str = new Scanner(System.in).nextLine();
 socket.writeLine(str);
 if (str.equals("quit")) {
 break;
 }
 }
 socket.dispose();
 }
}
```

# Observer

- Behavioral pattern
- Creates a 1 to N dependency between objects
  - When the state of the observed object changes the others are notified about the event and also can change their state

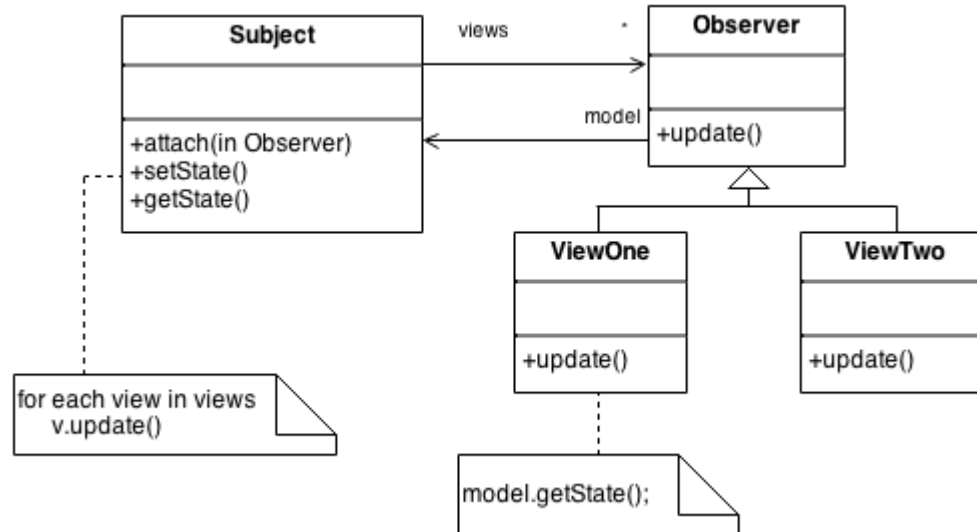


# Observer

- Define an object that is the "keeper" of the data model and/or business logic (the Subject).
  - Delegate all "view" functionality to decoupled and distinct Observer objects.
  - Observers register themselves with the Subject as they are created.
  - Whenever the Subject changes, it broadcasts to all registered Observers that it has changed, and each Observer queries the Subject for that subset of the Subject's state that it is responsible for monitoring.
- This allows the number and "type" of "view" objects to be configured dynamically, instead of being statically specified at compile-time.



# Observer



# Observer

```
interface AlarmListener {
 void alarm();
}

class SensorSystem {
 private Vector listeners = new Vector();

 public void register(AlarmListener alarmListener) {
 listeners.addElement(alarmListener);
 }

 public void soundTheAlarm() {
 for (Enumeration e = listeners.elements();
 e.hasMoreElements();) {
 ((AlarmListener) e.nextElement()).alarm();
 }
 }
}

class Lighting implements AlarmListener {
 public void alarm() {
 System.out.println("lights up");
 }
}

class Gates implements AlarmListener {
 public void alarm() {
 System.out.println("gates close");
 }
}
```

```
class CheckList {
 // Template Method design pattern
 public void byTheNumbers() {
 localize();
 isolate();
 identify();
 }

 protected void localize() {
 System.out.println(" establish a perimeter");
 }

 protected void isolate() {
 System.out.println(" isolate the grid");
 }

 protected void identify() {
 System.out.println(" identify the source");
 }
}

// class inherit.
// type inheritance
class Surveillance extends CheckList implements AlarmListener {
 public void alarm() {
 System.out.println("Surveillance - by the numbers:");
 byTheNumbers();
 }

 protected void isolate() {
 System.out.println(" train the cameras");
 }
}
```

# Delegate

- A specific object do not execute its task, instead of it delegates the task to another object
  - The other can be considered as a helper-object
- The responsibility is also delegated
  - It considered as a server, with responsibility
- Implementation
  - Through interface classes
  - In case of a function call, the implementation refers to an implementation of other class

# Delegation – Example

```
class I {
public:
 virtual void f() = 0;
 virtual void g() = 0;
 virtual ~I() {}
};

class A : public I {
public:
 void f() { cout << "A: doing f()" << endl; }
 void g() { cout << "A: doing g()" << endl; }
 ~A() { cout << "A: cleaning up." << endl; }
};

class B : public I {
public:
 void f() { cout << "B: doing f()" << endl; }
 void g() { cout << "B: doing g()" << endl; }
 ~B() { cout << "B: cleaning up." << endl; }
};
```

# Delegation – Example

```
class C : public I {
public:
 C() : i(new A()) { }
 virtual ~C() { delete i; }
private:
 I* i;
public:
 void f() { i->f(); }
 void g() { i->g(); }
 void toA() { delete i; i = new A(); }
 void toB() { delete i; i = new B(); }
};

int main() {
 C c;
 c.f(); //A: doing f()
 c.g(); //A: doing g()
 c.toB(); //A: cleaning up.
 c.f(); //B: doing f()
 c.g(); //B: doing g()
}
```

# Target-Action

- Used in event-driven applications
  - Programs with GUI
- The objects of the program are in dynamic connection with each other
  - Target: a specific object that is the target of a message (task)
  - Action: what target should execute

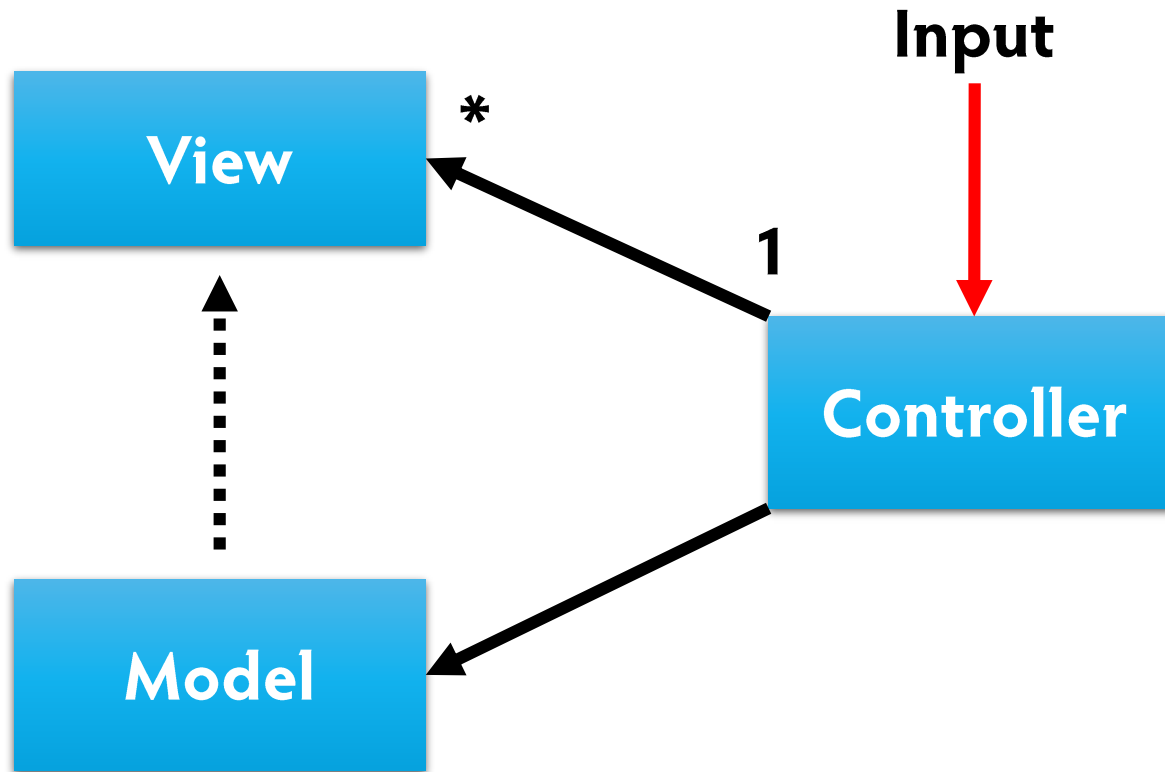
## ObjectiveC example

```
[button setTarget: self];
[button setAction: @selector(doSomething)];
```

# Design paradigms

- Complex structural patterns
- The design of entire structure is governed by these principles
  - Modell-View-Controller
  - Modell-View-Presenter
  - Modell-View-ViewModell
  - Target-Action

# MVC

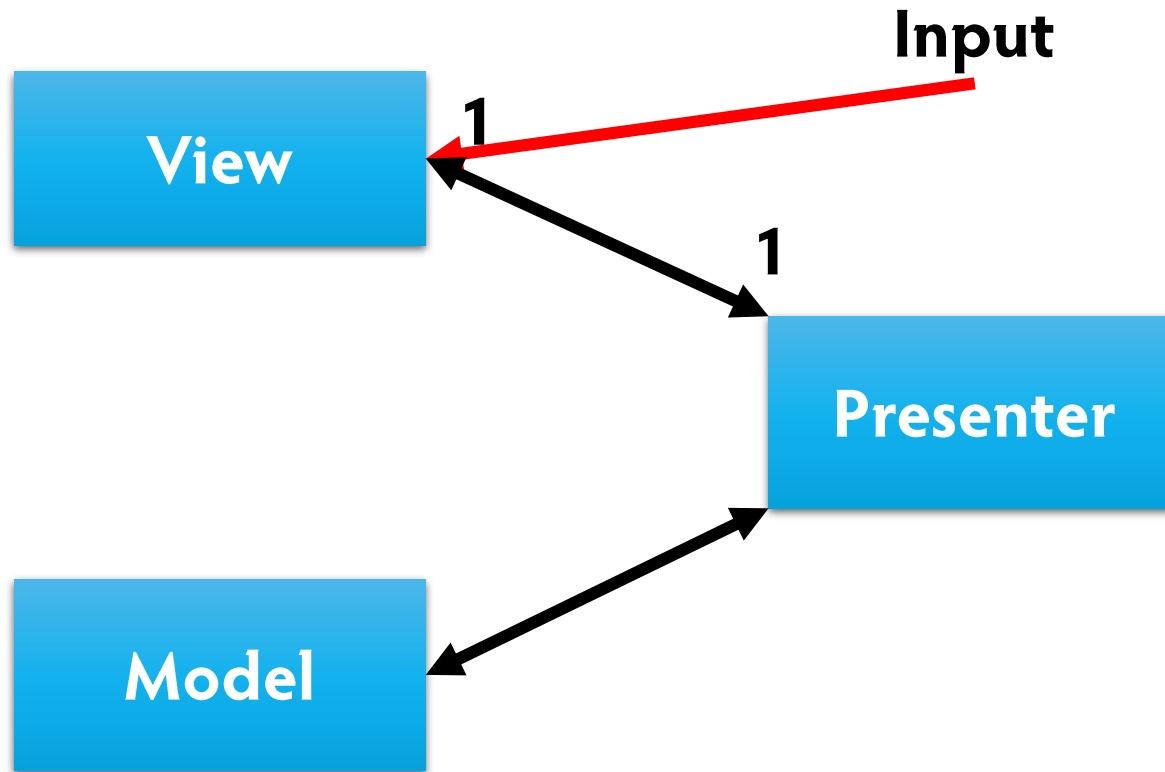




# MVC

- The application has three layers
  - Model
    - The representation of the information stored by the application
      - Plain data is augmented with meta data to provide meaning
    - Applications use permanent storing procedures to save data
    - The data access layer is part of the model, most of the cases
  - View
    - Visualize the model in the correct form, which is capable of user interaction
    - Typically it is a UI element
    - Different view for different objective may be exist
  - Controller
    - Events (mostly user interactions) are processed and appropriate response is generated
    - May change the model

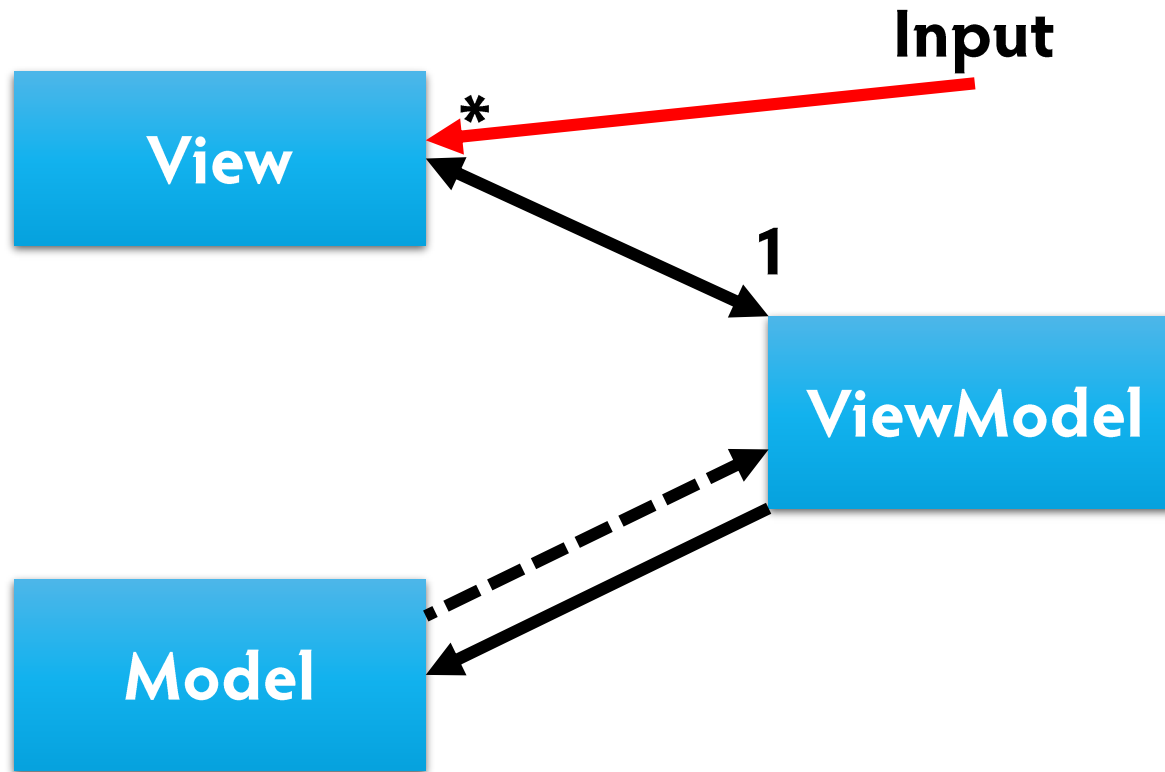
# MVP



# MVP

- The application has three layers
  - Model
    - The representation of the information stored by the application
      - Plain data is augmented with meta data to provide meaning
    - Applications use permanent storing procedures to save data
    - The data access layer is part of the model, most of the cases
  - View
    - Visualize the model in the correct form, which is capable of user interaction
    - Accepts the user interaction on View and send to the Presenter
      - Passive View: only displays the data, all business logic and transformation is in the Presenter
      - Supervisor View: some of the control tasks are here. It simplifies the Presenter by removing the conversions, checks, UI repaint/redraw procedures. Thus in Presenter only the business logic should be implemented.
  - Presenter
    - Middle layer, keeps the application in one piece
    - Business logic, and process control
    - Transforms data and transport between View and Model layer

# MVVM



# MVVM

- This pattern is similar to the MVC pattern
  - Designed for event-driven applications
  - Model
    - The representation of the information stored by the application
      - Plain data is augmented with meta data to provide meaning
    - Application use permanent storing procedures to save data
    - The data access layer is part of the model, most of the cases
  - View
    - Visualize the model in the correct form, which is capable of user interaction
    - Typically it is a UI element
    - Different view for different objective may be exist
  - ViewModel
    - Model of View
    - It can be considered as a special Controller, which converts the information coming from the Model to View, and the commands coming from the View to the Model
    - Public properties, commands and abstract interface are provided
    - It represents the conceptional state of data, instead of the real data which is stored in the model



# Tests, software design

# Software design

- Remember / recall
  - Basics of Software Technology
- Most important
  - Reasoned plan and software
  - Even drawings, diagrams, etc.
- Debugging is not trivial
  - But not impossible as well
- Emulator / simulator is not perfect
  - Some of the functions are missing
  - Varying quality of hardware and implementation
    - The specifications are not followed precisely

# Test – Validation

- Verification and validation – objectives:
  - To find the errors in the system
  - To ascertain about that the system can be used in real situations
- Most widespread validation technique
- The errors themselves have to be found, not their absence
- A test is successful when at least one error is discovered
- Testing defects:
  - The objective is to discover faults and defects of the system
  - Types:
    - Component tests: black box, equivalence-classes, structural tests, path-test
    - Integration tests: „top-down/bottom-up”, interface tests, stress-test
    - Object oriented tests
- Statistical tests
  - Testing the performance and reliability, in real situations (with real user input, and frequency)



# Test types

- Functional tests
  - The program is considered as a black box, the test cases are made upon specification
  - The implementation is not taken into account
  - Tests can be designed in early stage if the development
  - Special knowledge may be required to design some of the tests
- Structural test
  - Tests are designed based on the structure and implementation of the program
  - Equivalence classes can be designed based on the structure and code
  - During test design the source code is analyzed to ensure that all statements are executed at least once
    - All execution path cannot be tested in reality

# Test types

- OOP test
  - Component and integration test can be used in OOP systems
  - Differences
    - The object as components are larger than simple functions
      - White box test can be applied with difficulties
    - Objects are loosely coupled, and the system/subsystem may not have unambiguous bottom/top
    - The source of reused components may be inaccessible this we cannot analyze
- Inspection
  - The objective of software inspection is to find defects
  - Approximately 60% of the errors can be found with the cheaper inspection
  - A single test is capable of revealing a single error, inspection can find multiple errors
  - Inspection and testing cannot substitute each other
  - Inspection should be executed in the early stages of the development, to reduce the cost of tests

# Load tests

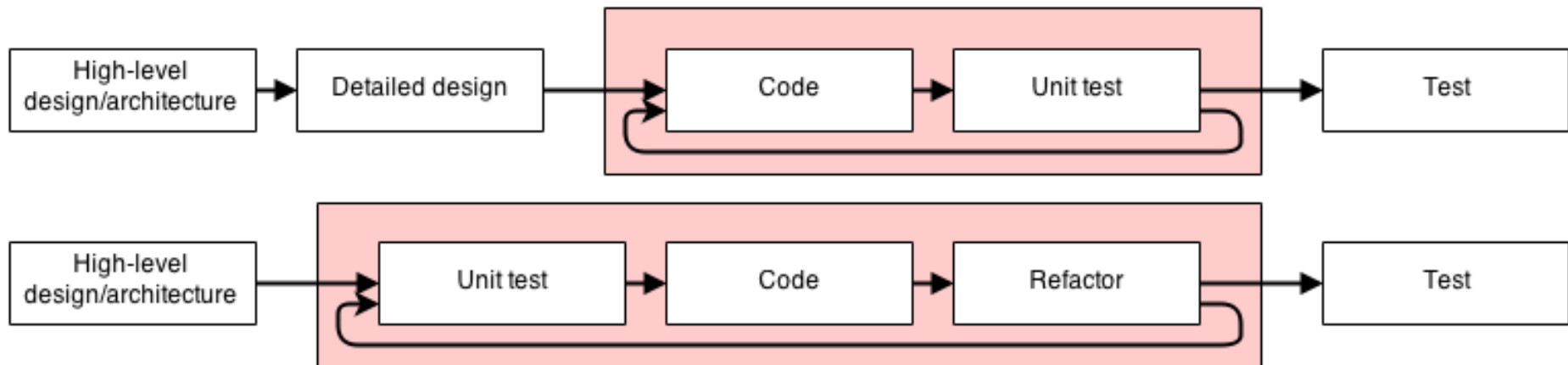
- The systems should be tested with larger load (than designed)
  - The load should be increased gradually, until system failure or performance degradation
- Tasks
  - To test the behavior of the system under extreme conditions
  - Overload must not cause data loss, or complete service failure
  - Such defects can be discovered, which do not happen under normal conditions.
- It is especially important in case of distributed systems
  - Larger load may cause coordination / load distribution problems
  - Thus it may result in increasing and self sustaining overload process

# Test design

- For each program unit tests should be designed
  - Unit tests
  - Component integration tests
- Tests have to be started to design in parallel with software design process

# Software life cycle– TDD

- Agile method very popular



# Extreme Programming tests

- In the Extreme Programming approach,
  - Tests are written before the code itself
  - If code has no automated test case, it is assumed not to work
  - A test framework is used so that automated testing can be done after every small change to the code
    - This may be as often as every 5 or 10 minutes
  - If a bug is found after development, a test is created to keep the bug from coming back
- Consequences
  - Fewer bugs
  - More maintainable code
  - Continuous integration
    - During development, the program always works
    - it may not do everything required, but what it does, it does right

# Two examples

- ```
int max(int a, int b) {  
    if (a > b) {  
        return a;  
    } else {  
        return b;  
    }  
}
```
- ```
void testMax() {
 int x = max(3, 7);
 if (x != 7) {
 System.out.println("max(3, 7) gives " + x);
 }
 x = max(3, -7);
 if (x != 3) {
 System.out.println("max(3, -7)
gives " + x);
 }
}
```
- ```
public static void main(String[] args) {  
    new MyClass().testMax();  
}
```
- ```
@Test
void testMax() {
 assertEquals(7, max(3, 7));
 assertEquals(3, max(3, -7));
}
```

# JUnit

- JUnit is a framework for writing tests
  - JUnit was written by Erich Gamma (of Design Patterns fame) and Kent Beck (creator of XP methodology)
  - JUnit uses Java's reflection capabilities (Java programs can examine their own code)
  - JUnit helps the programmer:
    - define and execute tests and test suites
    - formalize requirements and clarify architecture
    - write and debug code
    - integrate code and always be ready to release a working version
  - JUnit is not included in SDK, but almost all IDEs include it
    - You have to import it to your project



# Terminology

- A test fixture sets up the data (both objects and primitives) that are needed to run tests
  - Example: If you are testing code that updates an employee record, you need an employee record to test it on
- A unit test is a test of a single class
- A test case tests the response of a single method to a particular set of inputs
- A test suite is a collection of test cases
- A test runner is software that runs tests and reports results
- An integration test is a test of how well classes work together
  - JUnit provides some limited support for integration tests

# Test suite

- Obviously you have to test your code to get it working in the first place
  - You can do ad hoc testing (testing whatever occurs to you at the moment), or
  - You can build a test suite (a thorough set of tests that can be run at any time)
- Disadvantages of writing a test suite
  - It's a lot of extra programming
    - True—but use of a good test framework can help quite a bit
  - You don't have time to do all that extra work
    - False—Experiments repeatedly show that test suites reduce debugging time more than the amount spent building the test suite
- Advantages of having a test suite
  - Your program will have many fewer bugs
  - It will be a lot easier to maintain and modify your program
    - This is a huge win for programs that, unlike class assignments, get actual use!

# Assert methods

- Each assert method has parameters like these:  
message, expected-value, actual-value
- Assert methods dealing with floating point numbers get an additional argument, a tolerance
- Each assert method has an equivalent version that does not take a message – however, this use is not recommended because:
  - messages helps documents the tests
  - messages provide additional information when reading failure logs

# More in test classes

- Suppose you want to test a class Counter
- `public class CounterTest`  
    `extends junit.framework.TestCase {`
  - This is the unit test for the Counter class
- `public CounterTest() { } //Default constructor`
- `protected void setUp()`
  - Test fixture creates and initializes instance variables, etc.
- `protected void tearDown()`
  - Releases any system resources used by the test fixture
- `public void testIncrement(), public void testDecrement()`
  - These methods contain tests for the Counter methods `increment()`, `decrement()`, etc.
  - Note capitalization convention

# JUnit tests for Counter

- ```
public class CounterTest extends junit.framework.TestCase {  
    Counter counter1;  
    public CounterTest() { } // default constructor
```
- ```
 protected void setUp() { // creates a (simple) test fixture
 counter1 = new Counter();
 }
```
- ```
    public void testIncrement() {  
        assertTrue(counter1.increment() == 1);  
        assertTrue(counter1.increment() == 2);  
    }
```
- ```
 public void testDecrement() {
 assertTrue(counter1.decrement() == -1);
 }
}
```

# TestSuites

- TestSuites collect a selection of tests to run them as a unit
- Collections automatically use TestSuites, however to specify the order in which tests are run, write your own:  

```
public static Test suite() {
 suite.addTest(new TestBowl("testBowl"));
 suite.addTest(new TestBowl("testAdding"));
 return suite;
 }
```
- Should seldom have to write your own TestSuites as each method in your TestCase should be independent of all others

- Can create TestSuites that test a whole package:

```
public static Test suite() {
 TestSuite suite = new TestSuite();
 suite.addTestSuite(TestBowl.class);
 suite.addTestSuite(TestFruit.class);
 return suite;
}
```

# A simple example

- Suppose that you have a class `Arithmetic` with methods `int multiply(int x, int y)`, and `boolean isPositive(int x)`
- ```
import org.junit.*;
import static org.junit.Assert.*;
public class ArithmeticTest {

    @Test
    public void testMultiply() {
        assertEquals(4, Arithmetic.multiply(2, 2));
        assertEquals(-15, Arithmetic.multiply(3, -5));
    }

    @Test
    public void testIsPositive() {
        assertTrue(Arithmetic.isPositive(5));
        assertFalse(Arithmetic.isPositive(-5));
        assertFalse(Arithmetic.isPositive(0));
    }
}
```

Writing a JUnit test class

- This page is really only for expensive setup, such as when you need to connect to a database to do your testing
 - If you wish, you can declare one method to be executed just once, when the class is first loaded

@BeforeClass

```
public static void setUpClass() throws Exception {  
    // one-time initialization code  
}
```

- If you wish, you can declare one method to be executed just once, to do cleanup after all the tests have been completed

@AfterClass

```
public static void tearDownClass() throws Exception {  
    // one-time cleanup code  
}
```


Special features of @Test

- You can limit how long a method is allowed to take
- This is good protection against infinite loops
- The time limit is specified in milliseconds
- The test fails if the method takes too long
- @Test(timeout=10)
public void greatBig() {
 assertTrue(program.ackerman(5, 5) > 10e12);
}
- Some method calls should throw an exception
- You can specify that a particular exception is expected
- The test will pass if the expected exception is thrown, and fail otherwise
- @Test(expected=IllegalArgumentException.class)
public void factorial() {
 program.factorial(-5);
}

Ignoring a test

- The @Ignore annotation says to not run a test

```
@Ignore("I don't want Dave to know this doesn't work")
@Test
public void add() {
    assertEquals(4, program.sum(2, 2));
}
```
- You shouldn't use @Ignore unless you have a very good reason!

Homework

1. Demonstrate Proxy pattern in C++!
 - Create a smart, reference counting pointer to any object!
 - Use templates
2. Demonstrate adapter pattern in Java
 - Based on a GUI example:
 - Two different methods of specifying size and location of UI elements
3. Create a Java application to demonstrate JUnit capabilities
 - Basic calculations
 - Populating an array



Objective-C

Next week



Pázmány Péter Catholic University
Faculty of Information Technology and Bionics

Basics of Mobile Application Development

Swift I.



Reminder

iOS building blocks

- Different software and software levels
 - Based on OS X (Darwin)
- Development on SWIFT language
 - Brand new language, not extension for Objective-C
 - However its logic can be understand better based on Objective-C
- Development on Objective-C language
 - Based on C
 - Thin layer on the C
 - Fully object oriented
 - Smalltalk messaging model

Cocoa Touch

Media

Core Services

Core OS

Details

- Core OS
 - Mac OS X Kernel
 - XNU
 - TCP/IP
 - Sockets
 - Energy managements
 - File systems
 - Security
- Core Services
 - Network management
 - Collections
 - Preference storage
 - URL handling
 - File accessing
 - Contacts
 - Embedded SQL database manager
 - Core Location
 - To determine the location of the devices (GPS, Cell, etc.)
 - Thread managements
 - CoreMotion
 - To retrieve the accelerometer data

Details

- Media

- Core Audio
- Open AL
 - Open Audio Library
- Sound mixer
- Sound recorder
- Video playback
- Supporting media formats
- 2D acceleration
- Core Animation
- OpenGL ES

- Cocoa Touch

- Multi-touch and gestures
- Camera support
- Localization (multilingual apps)
- Views and view hierarchies
- Accelerometer support
- Web view
- Map kit
- Handling notifications
- Core Motion
 - Provide and process motion related information

iOS development tools

- XCode
 - With iOS SDK
 - An OS X required as operating system (and an Apple computer)
 - Newer iOS requires newer XCode, and newer OS X
 - And newer hardware
- SDK includes a simulator
 - This is not emulator, it is a simulator
 - A simulator calculates the corresponding output for a specific input
 - Thus the iOS systems is not running on the test-environment.



Pázmány Péter Catholic University
Faculty of Information Technology and Bionics

SWIFT



Properties

- History
 - Created in 2010
 - Introduced in 2014
 - Version 2.0 in September 2015
 - Version 5.1 in September 2019
- Influenced by
 - Haskell
 - C#
 - Python
 - Ruby
 - Rust
 - CLU
- Successor of Objective-C

Properties

- Supports different programming paradigms
 - OOP
 - Functional
 - Protocol oriented
- Open
 - It is stated that it is matter of time that it appears on Android and Windows platform

How to try

- REPL
 - Read Eval Print Loop
 - In terminal `xcrun swift`
 - The code is executed immediately
- Playground
 - REPL in the Xcode
 - The Apple's Framework elements can be accessed
- REPL.IT
 - <https://repl.it>

Variables and type

- Simple values (var variable, let constants)

- `var implicitInteger = 70`
- `let implicitDouble = 70.0`
- `let explicitDouble: Double = 70`
- `let appleSummary = "I have \$(apples) apples."`
- `let fruitSummary = "I have \$(apples + oranges) pieces of fruit."`

- Array

- `var shoppingList = ["catfish", "water", "tulips", "blue paint"]`
- `shoppingList[1] = "bottle of water"`
- `shoppingList.append("car")`
- `shoppingList.insert("bus", at: 70)`
- `print(shoppingList)`

Variables and type

- Set

- `let oddDigits: Set = [1, 3, 5, 7, 9]`
- `let evenDigits: Set = [0, 2, 4, 6, 8]`
- `let singleDigitPrimeNumbers: Set = [2, 3, 5, 7]`
- `oddDigits.intersection(evenDigits).sorted()`
- `oddDigits.contains(1) // True`

- Dictionaries

- `var occupations = [`
 `"Malcolm": "Captain",`
 `"Kaylee" : "Mechanic",`
 `]`
- `occupations["Jayne"] = "Public Relations"`
- `let emptyArray = [String]()`
- `let emptyDictionary = [String: Float]()`

- Tuple

- `let k = (70, 80)`
- `k.0 // 70`

Optionals

- A variable can store a value or a `nil` value
- To do so you have to use optionals
 - `var optionalString: String? = "Hello"`
 - `println(optionalString == nil)`
 - `var optionalName: String? = "John Appleseed"`
 - `var greeting = "Hello!"`
 - `if let name = optionalName {
 greeting = "Hello, \(name)"
}`
 - To extract value if it is not nil: `optionalName!`

Control statements

- For loop

- ```
var firstForLoop = 0
for i in 0..<4 {
 firstForLoop += i
}
```
- ```
var secondForLoop = 0
for var i = 0; i < 4; ++i {
    secondForLoop += i
}
```

- The `.. is used to iterate the i between 0 and 3`
- To iterate between 0 and 4 you have to use
 - `i in 0...4`

Control statements

- For each (for in)

```
let individualScores = [75, 43, 103, 87, 12]
var teamScore = 0
for score in individualScores {
    if score > 50 {
        teamScore += 3
    } else {
        teamScore += 1
    }
}
```

Control statements

- Further options

```
• let interestingNumbers = [  
    "Prime": [2, 3, 5, 7, 11, 13],  
    "Fibonacci": [1, 1, 2, 3, 5, 8],  
    "Square": [1, 4, 9, 16, 25]  
]  
  
• var largest = 0  
  
• for (kind, numbers) in interestingNumbers {  
    for number in numbers {  
        if number > largest {  
            largest = number  
        }  
    }  
}
```

Control statements

- Further options
 - ```
let base = 3
let power = 10
var answer = 1
for _ in 1...power {
 answer *= base
}
```

# Control statements

- While loop

```
var n = 2
while n < 100 {
 n = n * 2
}
```

- Do while loop

```
var m = 2
repeat {
 m = m * 2
} while m < 10
```

# Control statements

- Multiple branches

```
import Foundation
```

```
let vegetable = "red pepper"
```

```
let vegetableComment: String
```

```
switch vegetable {
```

```
 case "celery":
```

```
 vegetableComment = "Add some raisins and make ants on a log"
```

```
 case "cucumber", "watercress":
```

```
 vegetableComment = "That would make a good tea sandwich."
```

```
 case let x where x.hasSuffix("pepper"):
```

```
 vegetableComment = "Is it a spicy \(x)?"
```

```
 default:
```

```
 vegetableComment = "Everything tastes good in soup."
```

```
}
```

# Functions

- Example

```
func greet(name: String, day: String) -> String {
 return "Hello \ \(name), today is \ \(day)."
}
greet(name: "Bob", day: "Tuesday")
```

- Details

- Parameters name and type, and the return type
- The syntax is similar to mathematics



# Functions

- Further example

- ```
func calculateStatistics(scores: [Int]) -> (min: Int, max: Int, sum: Int) {  
    var min = scores[0]  
    var max = scores[0]  
    var sum = 0  
    for score in scores {  
        if score > max {  
            max = score  
        } else if score < min {  
            min = score  
        }  
        sum += score  
    }  
    return (min, max, sum)  
}  
  
let statistics = calculateStatistics(scores: [5, 3, 100, 3, 9])
```

Functions

- Variable parameter number

```
func sumOf(numbers: Int ...) -> Int {  
    var sum = 0  
    for number in numbers {  
        sum += number  
    }  
    return sum  
}
```

```
sumOf(numbers: 42, 597, 12) // 651
```

Functions

- Optional tuple return types

```
func minMax(array: [Int]) -> (min: Int, max: Int)? {  
    if array.isEmpty { return nil }  
    var currentMin = array[0]  
    var currentMax = array[0]  
    for value in array[1..<array.count] {  
        if value < currentMin {  
            currentMin = value  
        } else if value > currentMax {  
            currentMax = value  
        }  
    }  
    return (currentMin, currentMax)  
}
```

Function parameters

Default parameter value

```
func someFunction(parameterWithDefault: Int = 42) {  
    // Function body  
}
```

The formal parameter is constant

Function parameters

You can define inout parameters, to change its value inside the function

```
func someFunction(parameter: inout Int) {  
    parameter *= 2  
}
```

```
var param = 10  
someFunction(parameter: &param)  
print(param)
```

Function as type

- Following example returns a function as result

```
func makeIncrementer() -> ((Int) -> Int) {  
    func addOne(number: Int) -> Int {  
        return 1 + number  
    }  
    return addOne  
}  
var increment = makeIncrementer()  
print(increment(7))
```

Function as type

- Similarly you can pass a function as parameter as well:

```
func hasAnyMatches(list: [Int],  
                    condition: (Int) -> Bool) -> Bool {  
    for item in list {  
        if condition(item) {  
            return true  
        }  
    }  
    return false  
}  
  
func lessThanTen(number: Int) -> Bool {  
    return number < 10  
}  
  
var numbers = [20, 19, 7, 12]  
print(hasAnyMatches(list: numbers, condition: lessThanTen))
```

Homework (Deadline: 10/15/19 10.15 am)

- Write a SWIFT code
 - Have a dictionary (key-value pairs)
 - Pass the dictionary to different processing functions
 - Find and return the
 - Largest value
 - Smallest key
 - Remove a specific key
 - Add a filter to the functions
 - The filter is passed as a function
 - Some string pattern search
 - The statistics (etc.) should consider only the values/keys matching the filter
 - Function returns true



Swift II.

Next week



Pázmány Péter Catholic University
Faculty of Information Technology and Bionics

Basics of Mobile Application Development

Swift II.

Multiple branches – revisited

Intervals

```
let approximateCount = 62
let countedThings = "moons orbiting Saturn"
let naturalCount: String
switch approximateCount {
case 0:
    naturalCount = "no"
case 1..<5:
    naturalCount = "a few"
case 5..<12:
    naturalCount = "several"
case 12..<100:
    naturalCount = "dozens of"
case 100..<1000:
    naturalCount = "hundreds of"
default:
    naturalCount = "many"
}
print("There are \"(naturalCount) \"(countedThings).")
```

Multiple branches – revisited

Tuples

```
let somePoint = (1, 1)
switch somePoint {
case (0, 0):
    print("\(somePoint) is at the origin")
case (_, 0):
    print("\(somePoint) is on the x-axis")
case (0, _):
    print("\(somePoint) is on the y-axis")
case (-2...2, -2...2):
    print("\(somePoint) is inside the box")
default:
    print("\(somePoint) is outside of the box")
}
```

Control Transfer Statements

- Control transfer statements change the order in which your code is executed, by transferring control from one piece of code to another.
- Swift has five control transfer statements:
 - `continue`
 - `break`
 - `fallthrough`
 - `return`
 - `throw`

Continue

- The continue statement tells a loop to stop what it is doing and start again at the beginning of the next iteration through the loop

```
let puzzleInput = "great minds think alike"
var puzzleOutput = ""
let charactersToRemove: [Character] = ["a", "e", "i", "o",
"u", " "]
for character in puzzleInput {
    if charactersToRemove.contains(character) {
        continue
    } else {
        puzzleOutput.append(character)
    }
}
print(puzzleOutput)
```

Break

- Break in a Loop Statement
 - When used inside a loop statement, break ends the loop's execution immediately and transfers control to the code after the loop's closing brace
- Break in a Switch Statement
 - When used inside a switch statement, break causes the switch statement to end its execution immediately and to transfer control to the code after the switch statement's closing brace

Fallthrough

- To enable the C-style fallthrough behavior in a switch

```
let integerToDescribe = 5
var description = "The number \$(integerToDescribe) is"
switch integerToDescribe {
case 2, 3, 5, 7, 11, 13, 17, 19:
    description += " a prime number, and also"
    fallthrough
default:
    description += " an integer."
}
print(description)
```


Function parameter argument label

- Each parameter has a name and argument label
 - By default, the argument label is the same as the parameter name

```
func greet(name: String, day: String) -> String {  
    return "Hello \ \(name), today is \ \(day)."
}
```
 - When a function is invoked the argument label has to be specified as it was seen last week

```
greet(name: "Bob", day: "Tuesday")
```
 - You can also specify you own name

```
func greet(name s1: String, day s2: String) -> String
```
 - It can be invoked the same way

```
greet(name: "Bob", day: "Tuesday")
```
 - You also can omit the label:

```
func greet(_ s1: String, _ s2: String) -> String
```
 - Then

```
greet("Bob", "Tuesday")
```

Enum

- Enum type

```
enum Rank: Int {  
    case Ace = 1  
    case Two, Three, Four, Five, Six, Seven,  
    Eight, Nine, Ten  
    case Jack, Queen, King  
}  
let ace = Rank.Ace  
let aceRawValue = ace.rawValue
```

Enum in switch

- A function for the previous enum

```
func simpleDescription() -> String {  
    switch self {  
        case .Ace:  
            return "ace"  
        case .Jack:  
            return "jack"  
        case .Queen:  
            return "queen"  
        case .King:  
            return "king"  
        default:  
            return String(self.rawValue)  
    }  
}
```

Another example

```
enum Suit {  
    case Spades, Hearts, Diamonds, Clubs  
    func simpleDescription() -> String {  
        switch self {  
            case .Spades:  
                return "spades"  
            case .Hearts:  
                return "hearts"  
            case .Diamonds:  
                return "diamonds"  
            case .Clubs:  
                return "clubs"  
        }  
    }  
}
```

Further options

- You can add additional values to the instance of an enum

```
enum ServerResponse {  
    case Result(String, String)  
    case Error(String)  
}  
let success = ServerResponse.Result("6:00 am", "8:09 pm")  
let failure = ServerResponse.Error("Out of cheese.")  
switch success {  
case let .Result(sunrise, sunset):  
    let serverResponse = "Time1 \ \(sunrise), time2 \ \(sunset)."  
case let .Error(error):  
    let serverResponse = "Failure... \ \(error)"  
}
```

Struct

```
struct Card {  
    var rank: Rank  
    var suit: Suit  
    func simpleDescription() -> String {  
        return "The \((rank.simpleDescription()) of  
                \((suit.simpleDescription())"  
    }  
}  
  
let threeOfSpades = Card(rank: .Three, suit: .Spades)  
let threeOfSpadesDescription =  
threeOfSpades.simpleDescription()
```

Class and struct

- Both classes and structs can have
 - Properties to store data
 - Get and set functions
 - Methods to implements functions
 - Initializer, to set the initial state
 - Indexer
 - Extensions to add new capabilities to the default functionalities
 - The also can implement (match to) protocols, to provide standard functions
- Classes
 - There is inheritance between classes
 - Type conversion in runtime is possible
 - You can deinitialize classes to free up resources
 - By using reference counting the life cycle of the classes is handled automatically

Differences

- The struct and enum are by value types while the classes are reference types
 - It means that in case of parameter passing the first ones are copied
 - In case of the latter one, the reference is copied thus the instance is not duplicated
- The built-in String, Array and Dictionary are implemented as structs

Properties – fields

Example

```
struct FixedLengthRange {  
    var firstValue: Int  
    let length: Int  
}  
  
var rangeOfThreeItems =  
FixedLengthRange(firstValue: 0, length: 3)  
rangeOfThreeItems.firstValue = 6  
  
!!! Error  
let rangeOfThreeItems =  
FixedLengthRange(firstValue: 0, length: 3)  
rangeOfThreeItems.firstValue = 6
```

Properties – fields

Calculated, manipulated value

```
struct Rect {  
    var origin = Point()  
    var size = Size()  
    var center: Point {  
        get {  
            let centerX = origin.x + (size.width / 2)  
            let centerY = origin.y + (size.height / 2)  
            return Point(x: centerX, y: centerY)  
        }  
        set(newCenter) {  
            origin.x = newCenter.x - (size.width / 2)  
            origin.y = newCenter.y - (size.height / 2)  
        }  
    }  
}  
  
struct Point {  
    var x = 0.0, y = 0.0  
}  
  
struct Size {  
    var width = 0.0, height = 0.0  
}
```

Observing a property

```
class StepCounter {  
    var totalSteps: Int = 0 {  
        willSet(newTotalSteps) {  
            print("About to set totalSteps to \$(newTotalSteps)")  
        }  
        didSet {  
            if totalSteps > oldValue {  
                print("Added \$(totalSteps - oldValue) steps")  
            }  
        }  
    }  
}
```

Properties – fields

- Class variables are defined with static keyword
- If there is no set function defined then the property is read only

```
class Counter {  
    static var maxCount = 10  
    var count = 0  
    func incrementCount() {  
        count += 1  
    }  
    static func incrementMaxCount(value : Int) {  
        maxCount += value  
    }  
}
```

```
Counter.incrementMaxCount(value: 5) // 15
```

Changing the state of a class

```
class Counter {  
    var count = 0  
    func increment() {  
        count += 1  
    }  
    func incrementBy(amount: Int) {  
        count += amount  
    }  
    func reset() {  
        count = 0  
    }  
}
```

Changing the state of a struct

```
struct Counter {  
    var count = 0  
    mutating func increment() {  
        count += 1  
    }  
}
```

- In case of struct the self variable is immutable

Initialization – Example

```
struct Color {  
    let red, green, blue: Double  
    init(red: Double, green: Double, blue: Double) {  
        self.red    = red  
        self.green   = green  
        self.blue    = blue  
    }  
    init(white: Double) {  
        red    = white  
        green  = white  
        blue   = white  
    }  
}
```

Inheritance, overriding

```
class Counter {  
    var count = 0  
    func incrementCount() {  
        count += 1  
    }  
}  
  
class ChildCounter : Counter{  
    override func incrementCount() {  
        count += 2  
    }  
}  
  
var c = ChildCounter()  
c.incrementCount()  
print(c.count)
```


Inheritance, overriding

- Overriding a class function (observe the class keyword)

```
class Counter {  
    static var maxCount = 10  
    var count = 0  
    func incrementCount() {  
        ++count  
    }  
    class func incrementMaxCount(value : Int) {  
        maxCount += value  
    }  
}  
Counter.incrementMaxCount(5) // 15
```

Inheritance, overriding

- Overriding a class function (observe the class keyword)

```
Counter.incrementMaxCount(5) // 15
```

```
class SuperCounter : Counter {  
    override class  
        func incrementMaxCount(value : Int) {  
            maxCount += 2*value  
        }  
}
```

```
SuperCounter.incrementMaxCount(5) // 25  
Counter.maxCount
```

Preventing the overriding

- Obviously
 - `final func`
 - `final class func`
 - `final var`
 - `final subscript`
- And
 - `final class`
- Without these modifiers, you can override
 - Functions
 - Properties
 - Get and Set functions
 - Indexers

Extension – protocol

- Adding a new function to any struct, enum
 - Possibilities
 - New calculated (derived) property
 - New class/instance function
 - New initializer
 - New indexer
 - New nested type
 - Existing type can be prepared to implement a new protocol
 - Keyword: `extension`
- Protocol
 - It is the well-known conception of interface

Examples

```
protocol Animal {  
    func name() -> String  
}  
  
class Dog : Animal {  
    func name() -> String {  
        return "Jack"  
    }  
}  
  
let favorite : Animal = Dog()  
favourite.name() // "Jack"
```

Examples

```
class Greyhound : Dog {  
    override func name() -> String {  
        return "Bruno"  
    }  
}  
  
let favourite2 : Animal = Greyhound()  
favourite2.name() // "Bruno"
```

Extension of a protocol

```
extension Animal {  
    func doubleName() -> String {  
        return name() + " " + name()  
    }  
}
```

```
let favourite2 : Greyhound = Greyhound()  
favourite2.doubleName() // "Bruno Bruno"  
favourite.doubleName()  // "Jack Jack"
```

Controlling the access

- The visibility of the members can be controlled with the common keywords
 - `public` – the public interface of the module, can be accessed from anywhere)
 - `internal` – can be accessed anywhere in the module (public in the module for interoperability)
 - `private` – can be accessed in the environment where it is defined

Serizalization of optionals

```
class One {  
    var number: Int?  
}  
class Simple {  
    var oneOrNot: One?  
}  
let optAndSimple: Simple? = Simple()  
optAndSimple!.oneOrNot = One()  
optAndSimple!.oneOrNot!.number = 5  
  
if let c = optAndSimple?.oneOrNot?.number {  
    print("The number is \(c)")  
} else {  
    print("No number")  
}
```

Homework – Deadline 10/22/2019 10.15 am

- You have to create a demonstration of SWIFT object oriented capabilities
- Details
 - Create a class to represent any cards
 - Content, initialization
 - Comparison
 - Create a class to represent playing cards
 - Suit, rank
 - Use inheritance
 - Use enum
 - Create a deck for cards and playing cards as well
- Test your code



Objective-C

Next week



Basics of Mobile Application Development

Objective-C

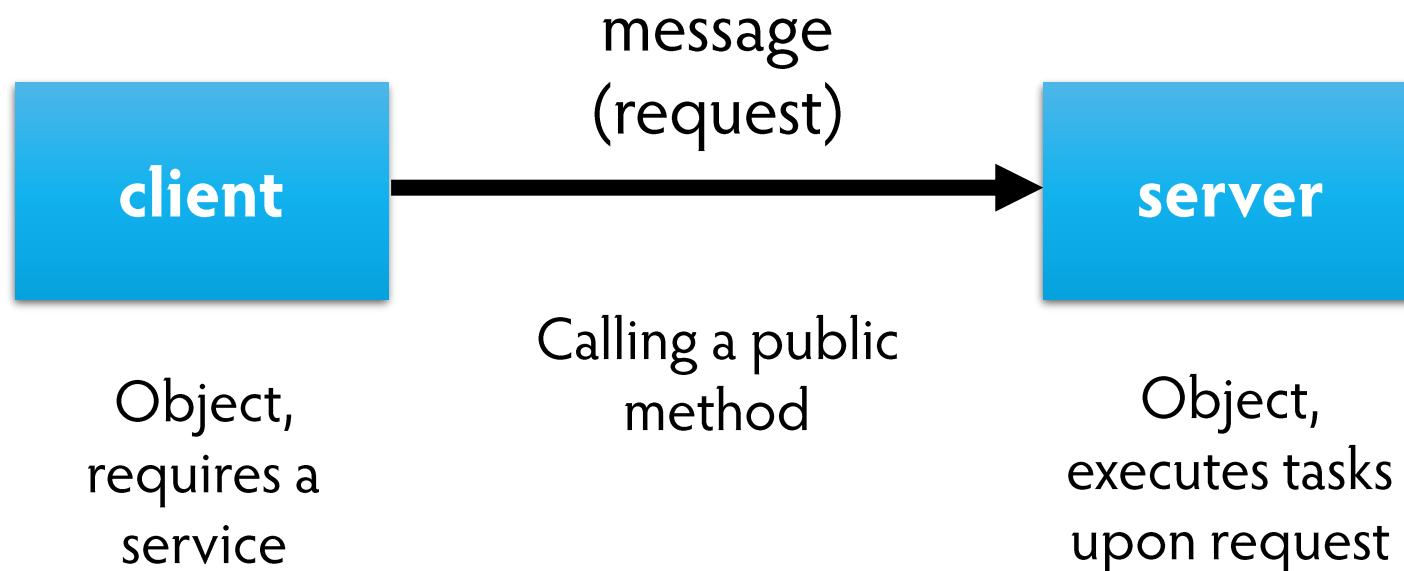


Reminder

OOP – keywords

- Object
 - Represents of entities of the real world
- Class of objects
 - Group of similar objects
 - Behavior
 - Structure
 - Template to create objects
- Method
 - A function (procedure) which manipulates the state of an object
- Field
 - A variable defining a property of an object
- Messaging
 - Interaction of objects
 - Interfaces are defined to facilitate the communication of objects
- Abstraction
 - Grouping classes
- Hierarchy
 - Design and implementation tool

Client sends a message





Objective-C

Basic properties

- Extension of C language
 - This is not C++
 - Thin layer on the C, which is processed by the preprocessor
 - New syntactic elements to create classes and methods
 - Smalltalk-style
 - Fully object oriented
 - The C variables, functions are the same
 - All C code can be compiled with the Objective-C compiler
 - The iOS framework was Objective-C based originally
 - There are existing Objective-C based codes, libraries
- Short history
 - Objective-C was developed by the beginning of 1980s
 - NeXT bought the license
 - Apple acquired NeXT

Data types

- C primitives
 - Without explicit type: `void`
 - Integers: (unsigned) `short`, `char`, `int`, `long`, `long long`
 - Fix size integers: `int8_t`, `uint16_t`, ...
 - Floating-point number: `float`, `double`, `long double`
- Objective-C primitives
 - Logical: `BOOL` (two values: `YES` and `NO`)
 - Base type of the objects: `id`
 - Data type for instances of meta-classes: `Class`
 - Type for storing selectors (to store functions): `SEL`
- Types can be used as usual, examples:
 - `short aShort = 1234;`
 - `char aChar = 'A';`
 - `BOOL isGood = YES;`

Data types

- Basic objects
 - Objects: `NSObject`
 - Superclass most of the objects
 - Numbers: `NSNumber`
 - Immutable – the same reasons as in Java
 - Text: `NSString`
 - Immutable
 - Fixedpoint numbers: `NSDecimalNumber`
 - Immutable
 - Collections:
 - Set (immutable): `NSSet`
 - Array (immutable): `NSArray`
 - Key-Value pairs (immutable): `NSDictionary`
 - Date: `NSDate`
- Each above can be accessed through pointers
 - The instantiation and lifetime will be discussed

Classes

- The concept of the classes and objects are the same
 - However some keywords are used in other way
- Class declaration is separated into:
 - .h file – header, which contains the public interface of class/object
 - .m file – implementation, which contains the private implementation
- On the following slides the .h file is on the left side and the .m file is on the right side
- Remember! New keywords starts with the @ symbol, as the preprocessor has to find them

Creating class

Card.H file

- A Card class will be created, which is subclass of the NSObject.
- The superclass must be imported.

```
#import <Foundation/Foundation.h>

@interface Card : NSObject
// Public declarations

@end
```

Card.M file

- Implementation of the same class. (You do not have to specify the superclass again)
- The .h file must be imported.

```
#import "Card.h"

@implementation Card
// Implementation

@end
```

.h import

- Previously an element of the framework has been imported
- To import the entire framework
 - `@import Foundation;`
- To import anything else
 - `#import "Superclass"`

@interface and @end

- Between the two keywords you can specify the interface if the class
 - The fields and the methods can be specified
 - In the .h file the public, in the .m file the private members
 - There is no other visibility level
- In case of .m file
- `#import "Card.h"`

```
@interface Card()  
// Private declarations  
@end
```

```
@implementation Card
```

```
@end
```

@interface and @end

- Fields can be part of classes, which can be considered as the properties of the class/object
 - You can declare by using the `@property` keyword
 - The type and variable name must be specified
 - The declarations of get and set are also there
 - Members can only be accessed through methods
 - Public and private environment as well
 - Declaration is in the interface
 - Example
 - `@property (strong) NSString *contents;`
 - In this example a pointer refers to an `NSString`
 - If the object is a property, it must be accessed by using pointers
 - This brings the problem of memory management

@property

- A property can be **strong** or **weak**
 - **strong**: The object that is referred by the property, exists while at least one strong pointer refers to that specific object. (The number of reference is greater than zero.)
 - If you set it to `nil` the number of reference is decreased.
 - **weak**: If there is no strong pointer to that instance, then the object can be destroyed and the memory can be freed up.
 - The weak pointer is set to `nil` in that case.
- A property can be **nonatomic** as well
 - Then the access is not thread safe
 - In the other case, the compiler creates locks, and through of them the parallel access is controlled
 - Currently you can used **nonatomic**

@synthesize

- Behind the property there is a variable, which is declared by the compiler, along with the get and set functions
 - Its name is the name of the property, with an `_` before the name
 - You can override this behavior by using the `@synthesize` keyword
 - Previous example can be continued: In the `@implementation` part of the `.h` file:
 - `@synthesize contents = _contentsvariable;`
- Of course, you can write your own get and set messages
- The code that is created automatically is something like this:
 - `@synthesize contents = _contents`
 - ```
- (NSString *)contents
{
 return _contents;
}
```
  - ```
- (void)setContents:(NSString *)contents
{
    _contents = contents;
}
```

Further options

- A property can not only be an object
 - `@property (nonatomic) BOOL chosen;`
 - `@property (nonatomic) BOOL matched;`
 - Here there is no meaning to use strong/weak options, as they are not stored in the heap of the memory.
 - Thus they exist till the object exists
 - Arbitrary C type can be used, even structs
- You can specify the name of the get/set message
- Previous example
 - `-(BOOL)chosen`
 - Instead of the previous:
 - `@property (nonatomic, getter=isChosen) BOOL chosen;`
 - `@property (nonatomic, getter=isMatched) BOOL matched;`
 - Then
 - `-(BOOL)isChosen`
 - The readability of the code is better
- A property also can be **readonly** as well
 - And several others, which are not important at this point

Functions – Messages

- Instead of calling methods/functions, the message sending semantics comes into foreground
 - C++ style approach (traditional)
 - `foo->bar (parameter);`
 - Objective-C approach
 - `[foo bar:parameter];`
- It is determined in runtime whether the target object can or cannot process the request
 - Thus the type checking happens in runtime not in compilation time
 - Always expect NIL as response
- Additional information
 - The different parameters are defined through the name of the message
 - Traditionally
 - `-(type)method:(type)param1 :(type)param2;`
 - Objective-C
 - `-(type)method:(type)param1 andParam2:(type)param2;`

Overload!?

- We would like to have two messages with different parameter type
 - `-(int)doIt:(int)param1 :(int)param2;`
 - `-(int)doIt:(int)param1 :(NSString*)param2;`
- This is not allowed
- But if you include the name (purpose) of the parameter into the name of the message
 - `-(int)doIt:(int)param1 withSomeInt:(int)param2;`
 - `-(int)doIt:(int)param1 withSomeString:(NSString*)param2;`
- In that case the two messages have different names, so it is not overloading
 - Overload is not supported by Objective-C
 - But you can mimic, by using the id type
 - And the implementation decides what to do with the actual parameter
- In previous case, the two messages both have two parameters
 - Neither one is optional

Card example

Card.H file

```
#import <Foundation/Foundation.h>

@interface Card : NSObject

@property (strong) NSString
*contents;

@property (nonatomic,
getter=isChosen) BOOL chosen;

@property (nonatomic,
getter=isMatched) BOOL matched;

-(int)match:(Card *)card;

@end
```

Card.M file

```
• #import "Card.h"

@interface Card()
// Private declarations
@end

@implementation Card

-(int)match:(Card *)card
{
    int score = 0;

    // We calculate the score

    return score;
}

@end
```

New message

- ```
-(int)match:(Card *)card
{
 int score = 0;

 if ([card.contents isEqualToString:self.contents]) {
 score = 1;
 }

 return score;
}
```
- Observe
  - You send the message as previously mentioned
  - Instead of `this` there is `self`
  - As everything is an object, you can use `.` to access members
  - The name of the `isEqualToString`
  - The `self.contents` is the get message, similarly to `card.contents`

# Comparison

- The == operator compares the value in case of primitives and objects (pointers) as well
  - Unsurprisingly, the memory addresses are compared
- In case of objects you must specify a message, which can compare the objects based on their properties
- NSString:
  - isEqualToString
- NSNumber:
  - isEqualToNumber
- Etc.



# Extend the message – NSArray

- The signature of the message is extended
  - `-(int)match:(NSArray *)othercards;`
- Then the implementation will be
  - ```
-(int)match:(NSArray *)othercards
{
    int score = 0;

    for (Card *card in othercards) {
        if ([card.contents isEqualToString:self.contents]) {
            score = 1;
        }
    }

    return score;
}
```
- You can observe the for-each loop
 - The syntax of the for loop is the well known one

Using the class

Deck.H

```
• #import <Foundation/Foundation.h>
  #import "Card.h"

@interface Deck : NSObject

-(void)addCard:(Card *)card
atTop(BOOL)atTop;

-(void)addCard:(Card *)card

-(Card *)drawRandomCard;

@end
```

Deck.M

```
• #import "Deck.h"

@interface Deck()
@end

@implementation Deck

-(void)addCard:(Card *)card
atTop(BOOL)atTop
{
    // TODO
}

-(void)addCard:(Card *)card
{
    [self addCard:card atTop:NO];
}

-(Card *)drawRandomCard
{
    // TODO
}

@end
```

In previous code

- There are two versions of addCard message
 - If you want to delegate, then you can send the other message
- There is no data structure to store the card data
 - And there is no code to manage the data
- NSArray
 - It is immutable, thus it is not feasible
 - However there is the NSMutableArray type
 - We will send messages to the array
 - Indexing => sending a message

Using the code

- `#import "Deck.h"`

```
@interface Deck()  
@property (strong, nonatomic) NSMutableArray *cards;  
@end
```

```
@implementation Deck
```

```
-(void)addCard:(Card *)card atTop(BOOL)atTop  
{  
    if (aTop) {  
        [self.cards insertObject:card atIndex:0];  
    } else {  
        [self.cards addObject:card];  
    }  
}
```

```
...
```

```
@end
```

Using the code

- `@property (strong, nonatomic) NSMutableArray *cards;`
- This line creates the property and the variable of the property
- However the object is not initialized, so we have to it, and also, we have to manage the variable
 - Currently we have a problem, as in `addCard` we access to a `NIL` pointer
- The automatically generated get function:
 - `-(NSMutableArray *)cards { return _cards; }`
- It has to be replaced:
 - ```
-(NSMutableArray *) cards {
 if (!_cards) _cards = [[NSMutableArray alloc] init];
 return _cards;
}
```
- And we are now arriving to the important question of initialization

# Initialization of an object

- The memory for the object (of a pointer) has to be allocated and the object also has to be initialized
  - Previously, we used the new operator and we called a constructor
  - In Objective-C there are no such things, then we must send two different messages
  - Technically the two messages can be separated, but we should not do that
    - Also it is not forbidden to return NIL after initialization
- Initialization with literals
  - NSString: @"Hi guys";
  - NSNumber: @42;
  - NSArray: @[@"One", @"Two", @"Three"];
  - You do not have to deal with the lifecycle
- Initialization of an object
  - `[[NSMutableArray alloc] init]`

# drawRandomCard

```
-(Card *)drawRandomCard
{
 Card *randomCard = nil;
 if ([self.cards count]) {
 unsigned index = arc4random() % [self.cards count];
 randomCard = self.cards[index];
 [self.cards removeObjectAtIndex:index];
 }
 return randomCard;
}
```

- `cards[index]` is a message as well!

# Create a subclass

## PlayingCard.h

- `#import "Card.h"`

```
@interface PlayingCard : Card

@property (strong, nonatomic)
NSString *suit;

@property (nonatomic) NSUInteger
rank;

@end
```

## PlayingCard.m

- `#import "PlayingCard.h"`

```
@implementation PlayingCard

- (NSString *)contents
{
 return
 [NSString stringWithFormat:@"%d%@",
 self.rank, self.suit];
}

@end
```

- The get function of the contents is overridden here
- The variable is declared only once, in the superclass
- The string is not allocated, but created by formatting



# Continue

- ```
#import "PlayingCard.h"
@implementation PlayingCard

- (NSString *)contents
{
    NSArray *rankStrings = @[@"?", @"A", @"2", @"3", ..., @"10", @"J", @"Q", @"K"];
    return [rankStrings[self.rank] stringByAppendingString:self.suit];
}

@end
```
- Here the [] and @[] statements are translated to sending messages
- We can create our own get and set messages
- ```
- (NSString *)suit
{
 return _suit ? _suit : @"?";
}
```

# Lets continue

- Creating a set message to set the suit of the cards
  - However we face a problem, as neither the get and set message is generated automatically
  - Thus the property variable will not be synthesized
  - We have to do it manually
  - The property variable must not be accessed directly outside of the set/get messages
- `@synthesize suit = _suit;`
- The code
  - Remember, you can send a message to any object
- ```
(void)setSuit:(NSString *)suit
{
    if ([@"♥", @"♦", @"♠", @"♣" containsObject:suit]) {
        _suit = suit;
    }
}
```

Class members

- Previous members was instance members
 - To create class members you have to used the + symbol
 - `+ (NSArray *)validSuits`

```
{  
    return @[@"♥",@"♦",@"♠",@"♣"];  
}
```
- In this case you have to send the message to the class
 - `-(void)setSuit:(NSString *)suit`

```
{  
    if ([[PlayingCard validSuits]  
containsObject:suit]) {  
        _suit = suit;  
    }  
}
```
- You can create public and private class members as well

Current state

PlayingCard.h

```
• #import "Card.h"

@interface PlayingCard : Card

@property (strong, nonatomic) NSString *suit;

@property (nonatomic) NSUInteger rank;

+ (NSArray *)validSuits;
+ (NSArray *)rankStrings;
+ (NSUInteger)maxRank;

@end
```

PlayingCard.m

```
• #import "PlayingCard.h"
@implementation PlayingCard
@synthesize suit = _suit;
- (NSString *)contents
{
    NSArray *rankStrings = [PlayingCard rankStrings];
    return [rankStrings[self.rank]
            stringByAppendingString:self.suit];
}
+ (NSArray *)validSuits
{
    return @[@"♥", @"♦", @"♠", @"♣"];
}
+ (NSArray *)rankStrings
{
    return @[@"?", @"A", @"2", @"3", ..., @"10", @"J", @"Q", @"K"];
}
- (void)setSuit:(NSString *)suit
{
    if ([[PlayingCard validSuits] containsObject:suit]) {
        _suit = suit;
    }
}
- (NSString *)suit
{
    return _suit ? _suit : @"?";
}
+ (NSUInteger)maxRank { return [[self rankStrings] count]-1; }
@end
```

PlayingCardDeck

- This will contain the PlayingCards
 - Based on the Deck class
 - There is no extension in the interface part
 - Existing cards will be inserted during the initialization
- `init`
 - Unusual – compared to the well-known constructors
 - There is a return type – the type of the instance (`instancetype`)
 - The created instance is assigned to the `self` variable
 - The `init`, or any alternative have to be called immediately after the `alloc` call
 - Even if they can be separated technically

New class

PlayingCardDeck.h

- `#import "Deck.h"`

`@interface PlayingCardDeck : Deck`

`@end`

PlayingCardDeck.m

- `#import "PlayingCardDeck.h"`

`@implementation PlayingCardDeck`

`- (instancetype)init`
`{`
 `self = [super init];`
 `if (self) {`

 `}`
 `return self;`
`}`

`@end`
- Note
 - There is a return!
 - Superclass is initialized first
 - It can be resulted in NIL

PlayingCardDeck.m

```
• #import "PlayingCardDeck.h",  
  #import "PlayingCard.h"  
  
@implementation PlayingCardDeck  
  
- (instancetype)init  
{  
    self = [super init];  
    if (self) {  
        for (NSString *suit in [PlayingCard validSuits]) {  
            for (NSUInteger rank = 1; rank <= [PlayingCard maxRank]; rank++) {  
                PlayingCard *card = [[PlayingCard alloc] init];  
                card.rank = rank;  
                card.suit = suit;  
                [self addCard:card];  
            }  
        }  
    }  
    return self;  
}  
  
@end
```

Question

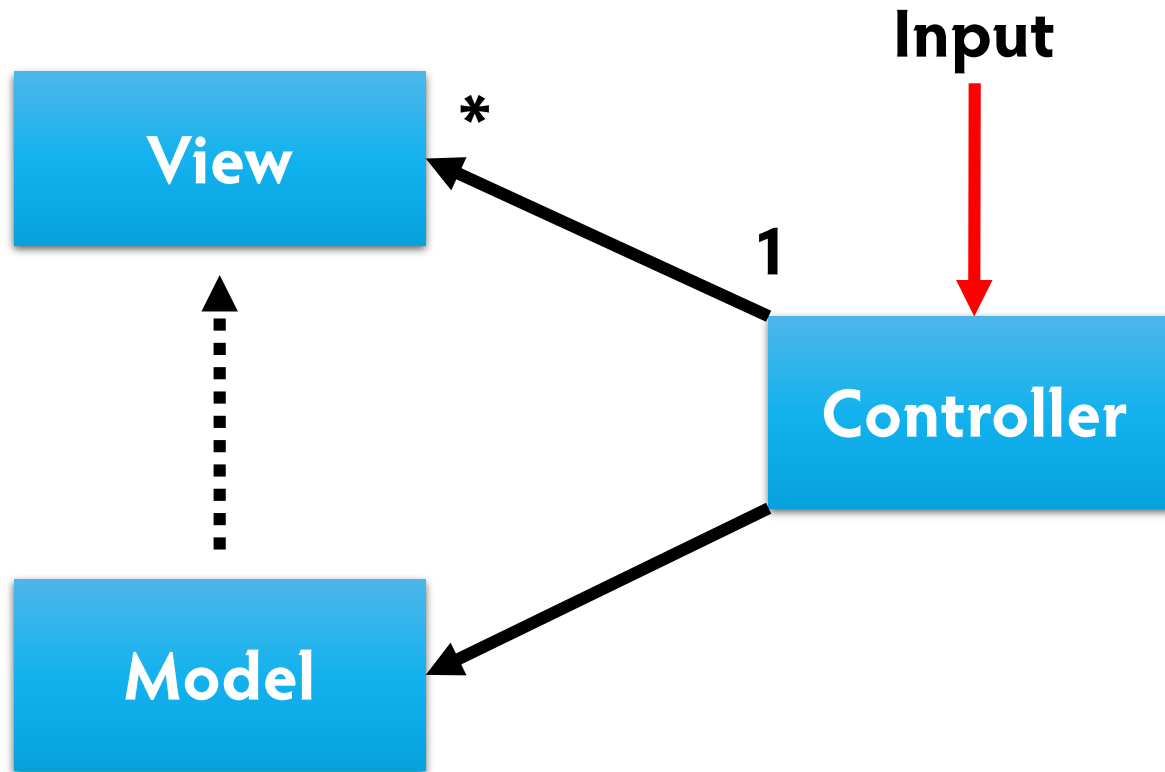
- What does the next line do?

```
cardA.contents = @[cardB.contents,cardC.contents][[cardB match:@[cardC]] ? 1 : 0]
```




MVC

MVC



MVC

- The application has three layers
 - Model
 - The representation of the information stored by the application
 - Plain data is augmented with meta data to provide meaning
 - Many application uses permanent storing procedure to save data
 - The data access layer is part of the model, most of the cases

MVC

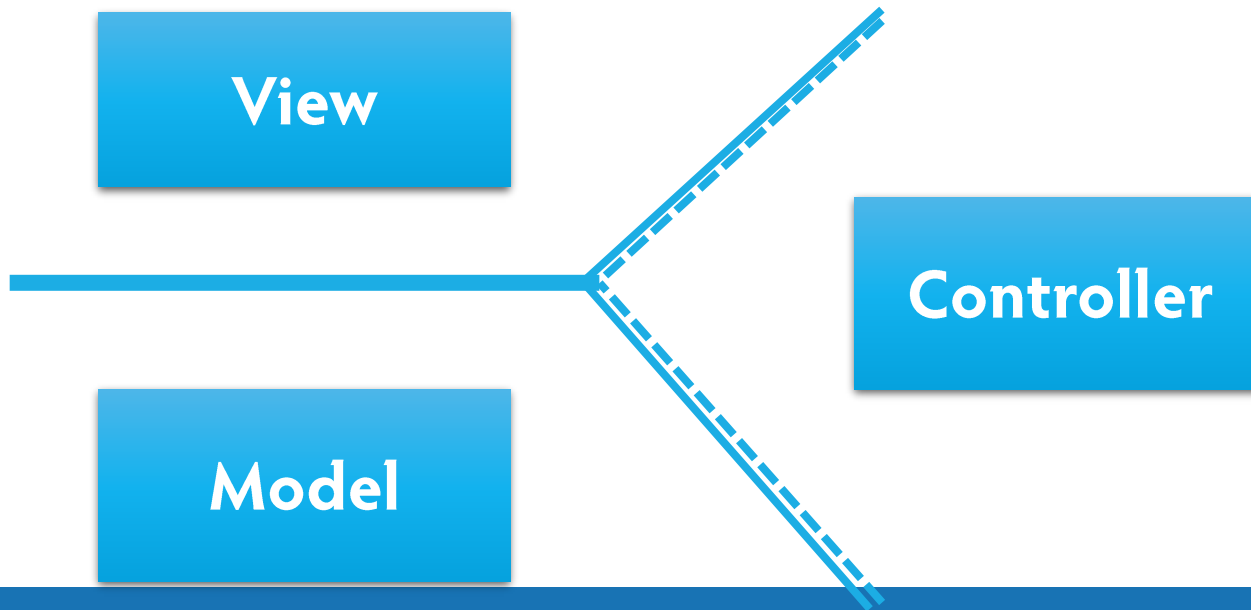
- The application has three layers
 - View
 - Visualize the model in the correct form, which is capable of user interaction
 - Typically it is a UI element
 - Different view for different objective may be exist

MVC

- The application has three layers
 - Controller
 - Events (mostly user interactions) are processed and appropriate response is generated
 - May change the model

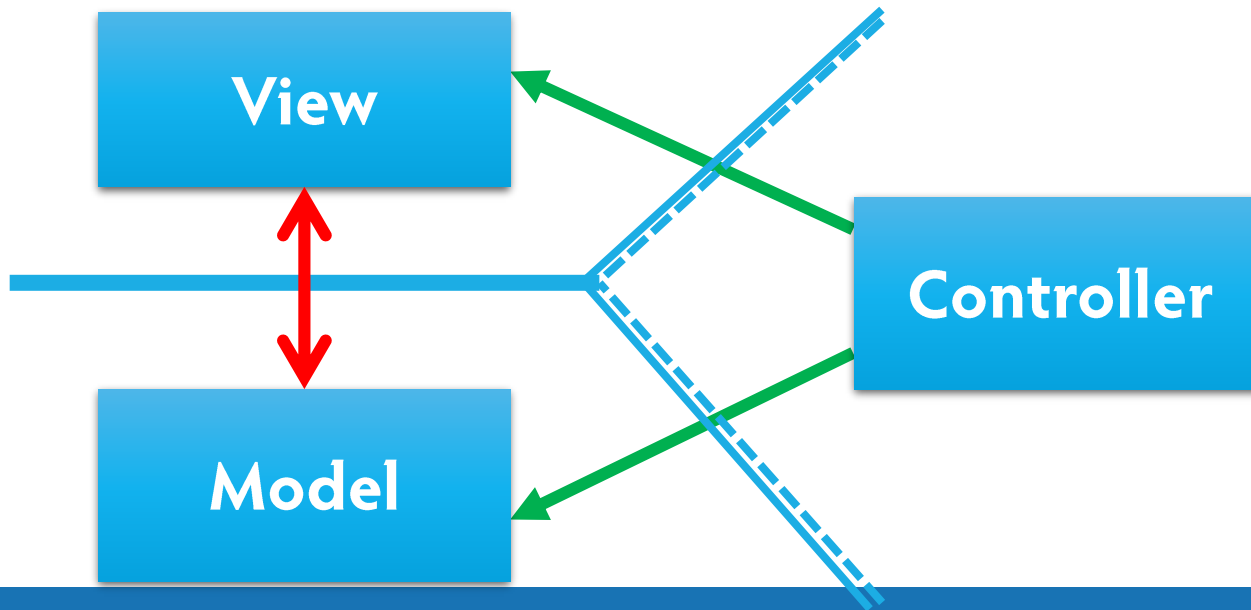
MVC

- Communication between the components



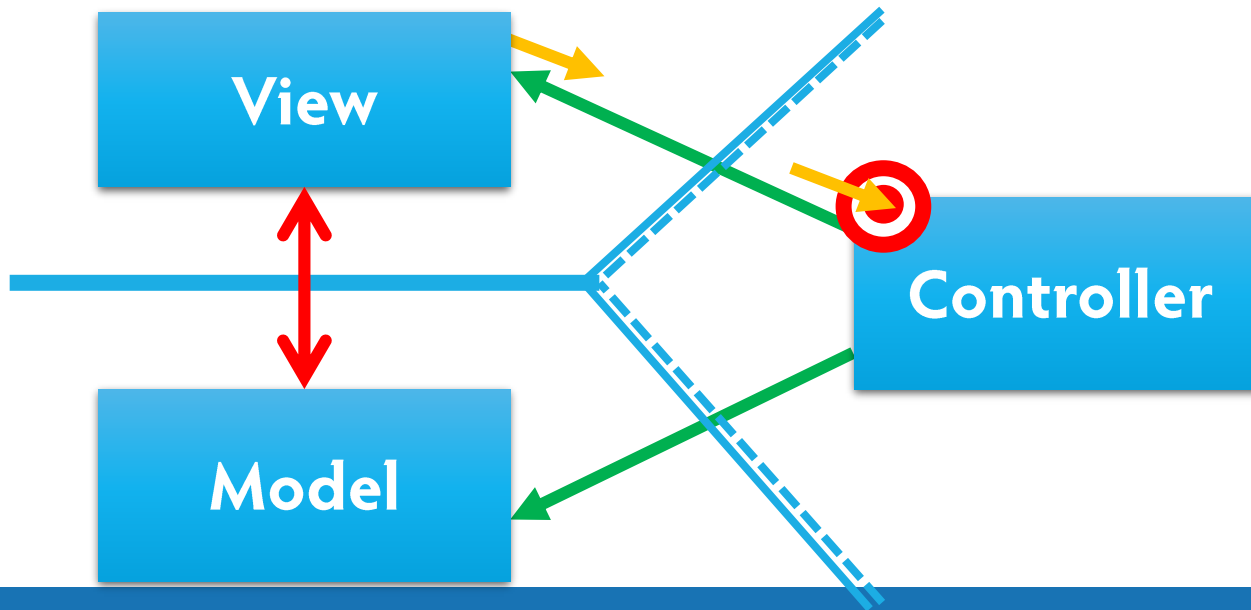
MVC

- Communication between the components
 - Controller communications with View and Model
 - View and Model cannot access each other directly



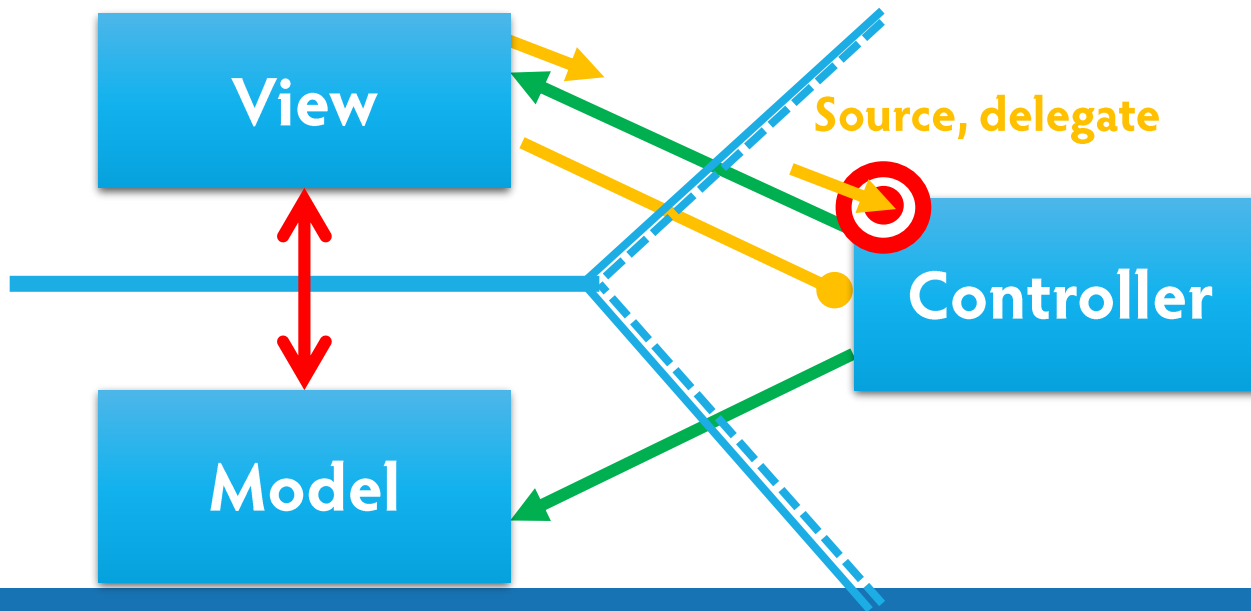
MVC

- However View notifies the controller indirectly
 - Controllers have to specify Targets
 - You can invoke Actions for specific Targets



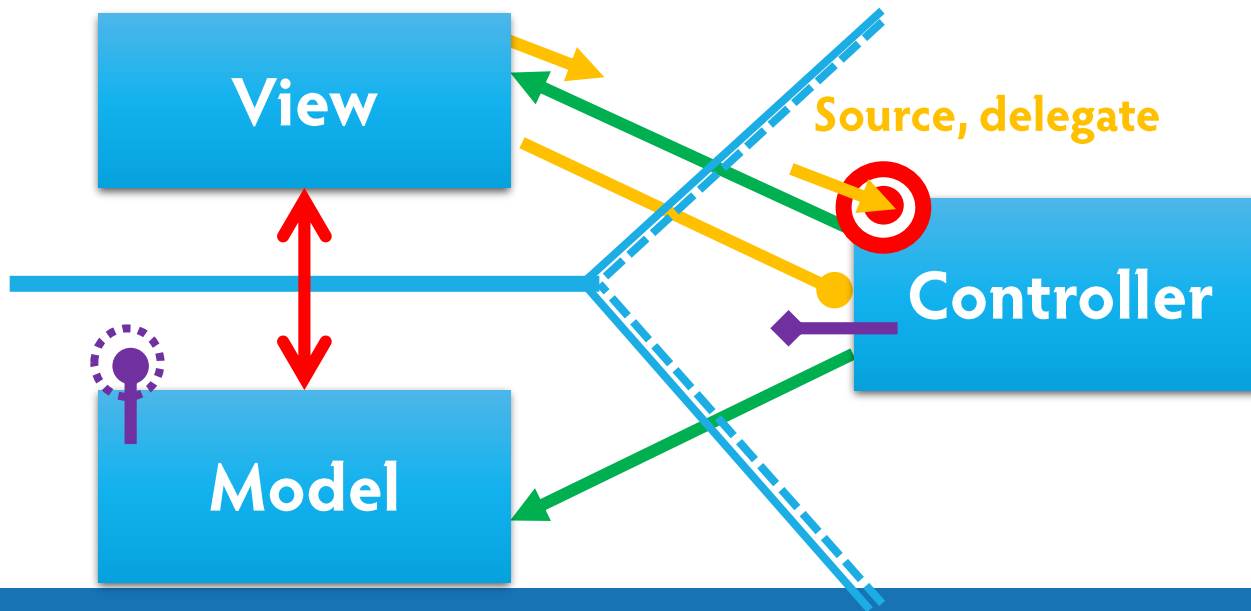
MVC

- In other cases the View has to be synchronized
 - Delegates have to be created
 - View have to know the structure of the data
 - Controller has to be act as a source of data
 - Data is transformed by the Controller

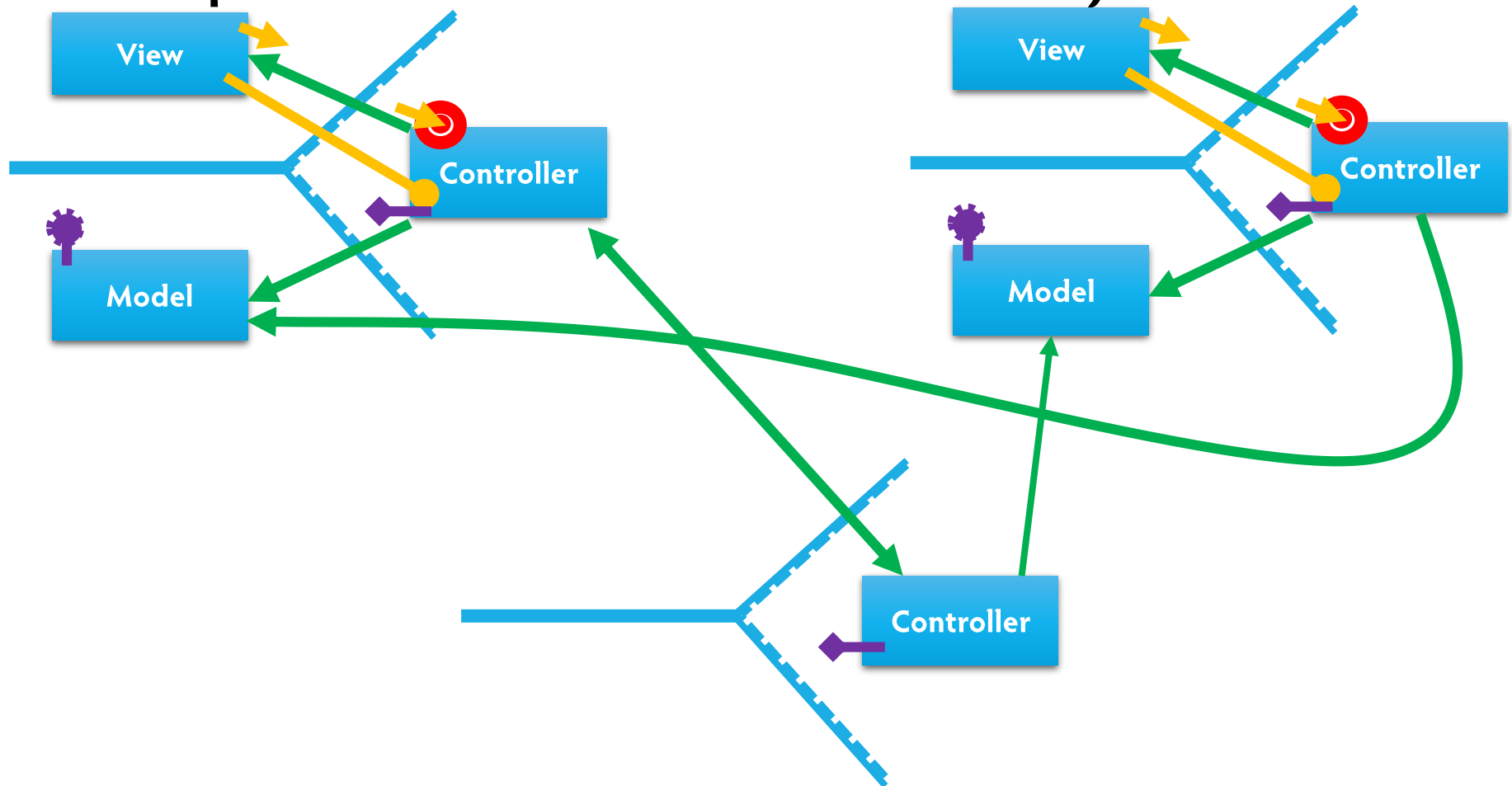


MVC

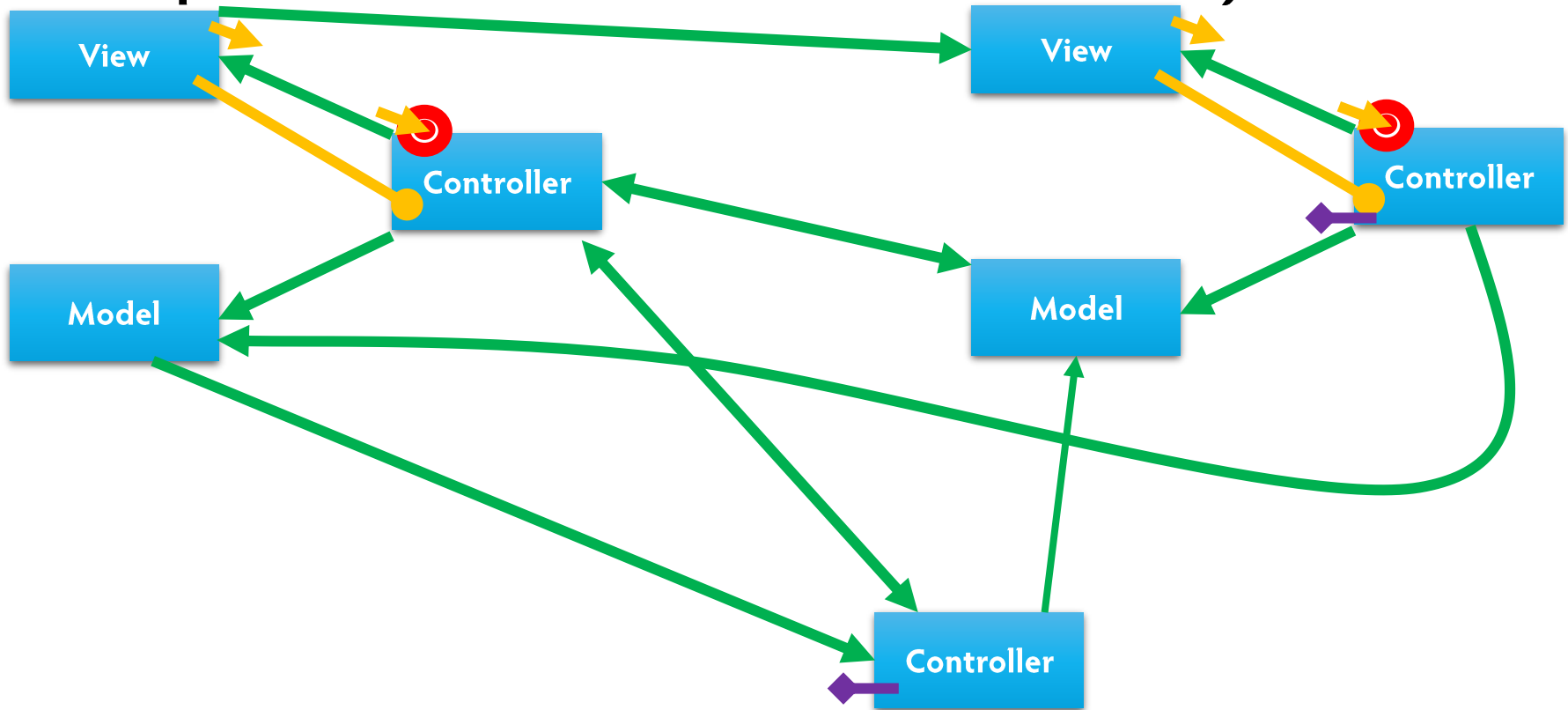
- Model also have to notify the Controller
 - No direct communication allowed
 - Broadcast messages are sent
 - Controllers, can act on



Multiple MVCs – rules are obeyed



Multiple MVCs – rules are disobeyed



Homework – Deadline 11/05 10.15 am

- Create a brief demonstration application
 - Have a Storage class, with internal variable to store values in an array
 - Create functions to retrieve
 - Value at index
 - Most frequent item
 - Smallest/largest item
 - Create a main to test your class, with messages

GNUStep setup

- On Windows 10
 - Install linux subsystem <https://docs.microsoft.com/en-us/windows/wsl/install-win10>
 - Continue with next block

On Linux

- Install gcc and extensions:
 - `sudo apt install gcc gobjc++ gnustep gnustep-devel gnustep-make`
- Write your code and compile

```
gcc -MMD -MP -DGNUSTEP -DGNUSTEP_BASE_LIBRARY=1 -DGNU_GUI_LIBRARY=1 -DGNU_RUNTIME=1 -  
DGNUSTEP_BASE_LIBRARY=1 -fno-strict-aliasing -fexceptions -fobjc-exceptions -D_NATIVE_OBJC_EXCEPTIONS -  
pthread -fPIC -Wall -DGSWARN -DGSDIAGNOSE -Wno-import -g -O2 -fgnu-runtime -fconstant-string-  
class=NSConstantString -I. -I /usr/include/GNUstep -o a.out main.m -lobjc -lgustep-base
```



Xcode and Android Studio

After the break



Pázmány Péter Catholic University
Faculty of Information Technology and Bionics

Basics of Mobile Application Development

Xcode Demo – Android Studio

First App - XCode

×



Welcome to Xcode



Get started with a playground

Explore new ideas quickly and easily.



Create a new Xcode project

Create an app for iPhone, iPad, Mac, Apple Watch or Apple TV.



Check out an existing project

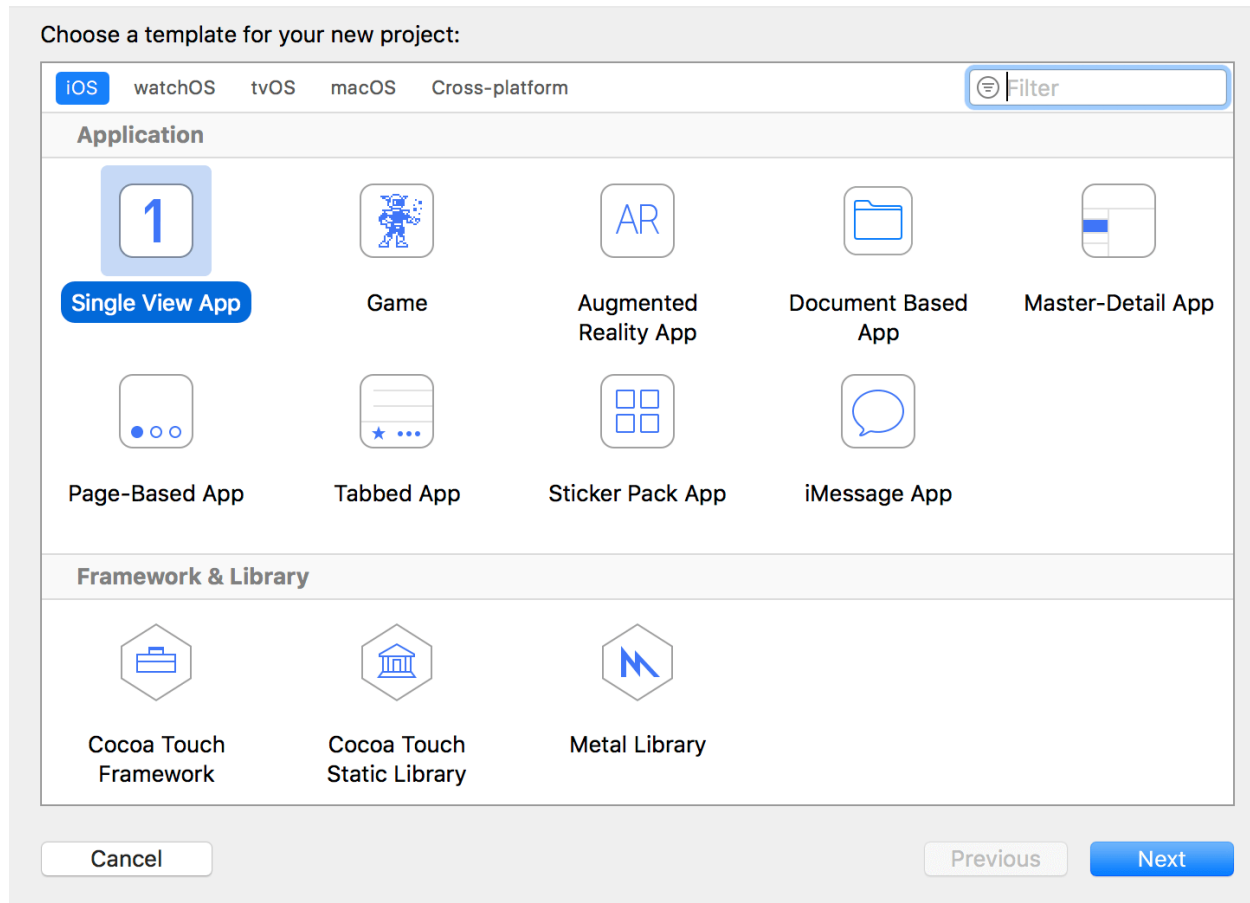
Start working on something from an SCM repository.

☒ Show this window when Xcode launches

No Recent Projects

Open another project...

First App - XCode



First App - XCode

Choose options for your new project:

Product Name: HelloWorld

Team: None

Organization Name:

Organization Identifier:

Bundle Identifier:

Language: Swift

☐ Use Core Data

☐ Include Unit Tests

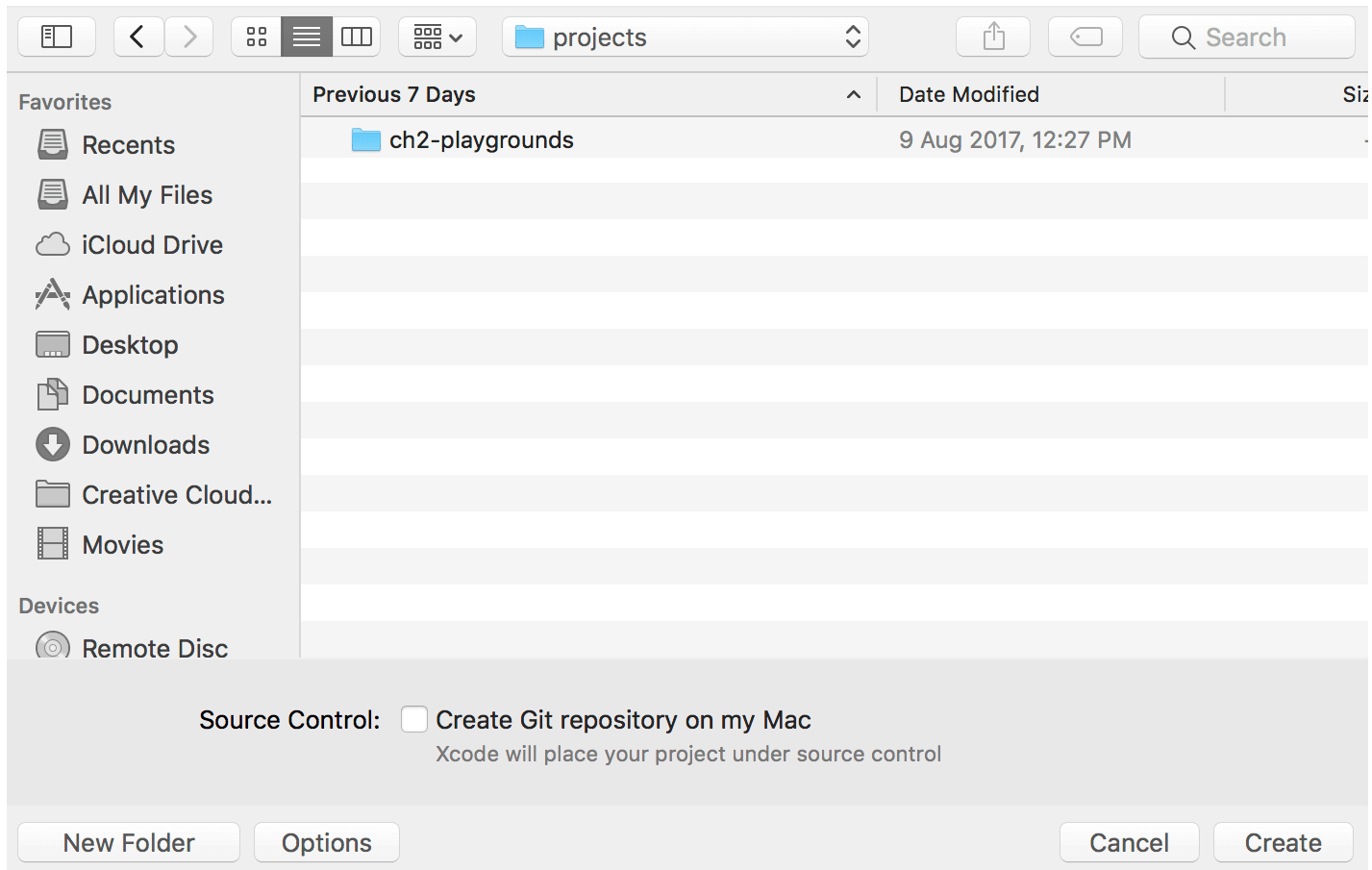
☐ Include UI Tests

Cancel Previous Next

Options

- **Product Name: HelloWorld**
 - This is the name of your app.
- **Organization Name: MAD**
 - It is the name of your organization.
 - If you are not building the app for your organization, use your name as the organization name.
- **Organization Identifier: mad.itk.ppke.hu**
 - It is actually the domain name written the other way round.
 - If you have a domain, you can use your own domain name.
- **Bundle Identifier**
 - It is a unique identifier of your app, which is used during app submission.
 - You do not need to fill in this option. Xcode automatically generates it for you.
- **Language: Swift**
 - Xcode supports both Objective-C and Swift for app development.
- **Use Core Data: [unchecked]**
 - You do not need Core Data for this simple project.
- **Include Unit Tests: [unchecked]**
 - You do not need unit tests for this simple project.
- **Include UI Tests: [unchecked]**
 - You do not need UI tests for this simple project.

First App - XCode





The screenshot displays the Xcode IDE interface for a project named 'HelloWorld'. The top status bar indicates 'HelloWorld: Ready' and 'Today at 12:34 PM'. The left sidebar shows the project's file structure, including 'AppDelegate.swift', 'ViewController.swift', 'Main.storyboard', 'Assets.xcassets', 'LaunchScreen.storyboard', 'Info.plist', and 'Products'. The main workspace is divided into two panes. The left pane shows the 'PROJECT' and 'TARGETS' sections, with 'HelloWorld' selected under 'TARGETS'. The right pane shows the 'Identity and Type' section, which includes fields for 'Name' (HelloWorld), 'Location' (Absolute), 'Full Path' (/Users/simon/Documents/AppCoda/Books/iOS 11 and Swift/projects/HelloWorld/HelloWorld.xcodeproj), 'Project Format' (Xcode 8.0-compatible), 'Organization', and 'Class Prefix'. Below this, the 'Text Settings' section shows 'Indent Using' (Spaces), 'Widths' (4), 'Tab' (4), 'Indent' (4), and a checked 'Wrap lines' option. The bottom of the right pane displays 'No Matches'.

PROJECT

- ▼ HelloWorld
 - AppDelegate.swift
 - ViewController.swift
 - Main.storyboard
 - Assets.xcassets
 - LaunchScreen.storyboard
 - Info.plist
 - Products

TARGETS

- ▼ HelloWorld

General Capabilities Resource Tags Info Build Settings Build Phases Build Rules

Identity

Display Name: HelloWorld

Bundle Identifier:

Version: 1.0

Build: 1

Signing

☒ Automatically manage signing
Xcode will create and update profiles, app IDs, and certificates.

Team: None

Provisioning Profile: Xcode Managed Profile

Signing Certificate: iOS Developer

Status: ❗ Signing for "HelloWorld" requires a development team.
Select a development team in the project editor.

Deployment Info

Deployment Target: 11.0

Devices: Universal

Main Interface: Main

Device Orientation: ☒ Portrait
☐ Upside Down
☒ Landscape Left
☒ Landscape Right

Status Bar Style: Default

☐ Hide status bar
☐ Requires full screen

App Icons and Launch Images

App Icons Source: AppIcon

Launch Images Source: Use Asset Catalog...

Launch Screen File: LaunchScreen

Embedded Binaries

Identity and Type

Name: HelloWorld

Location: Absolute

Full Path: /Users/simon/Documents/AppCoda/Books/iOS 11 and Swift/projects/HelloWorld/HelloWorld.xcodeproj

Project Document

Project Format: Xcode 8.0-compatible

Organization:

Class Prefix:

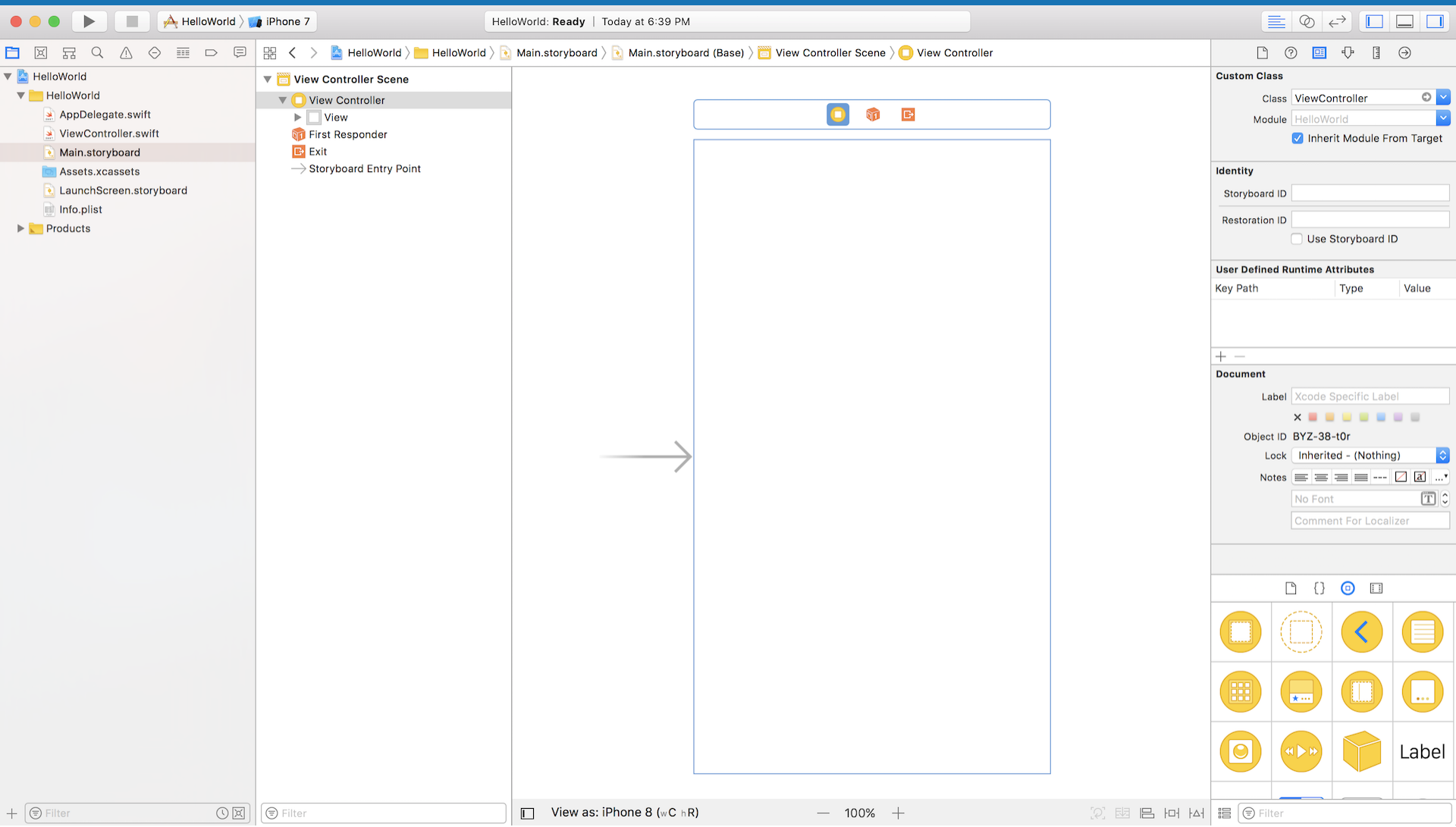
Text Settings

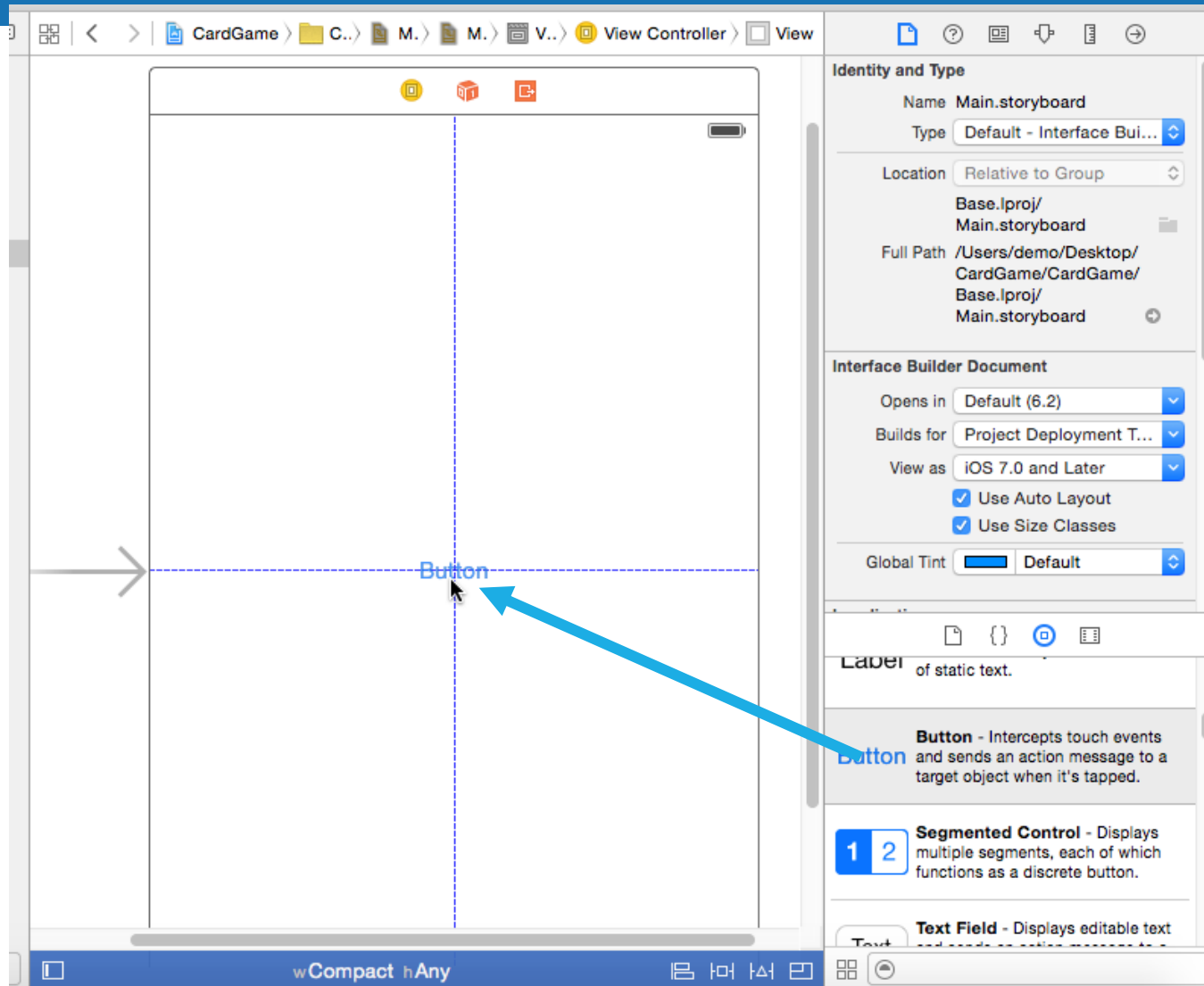
Indent Using: Spaces

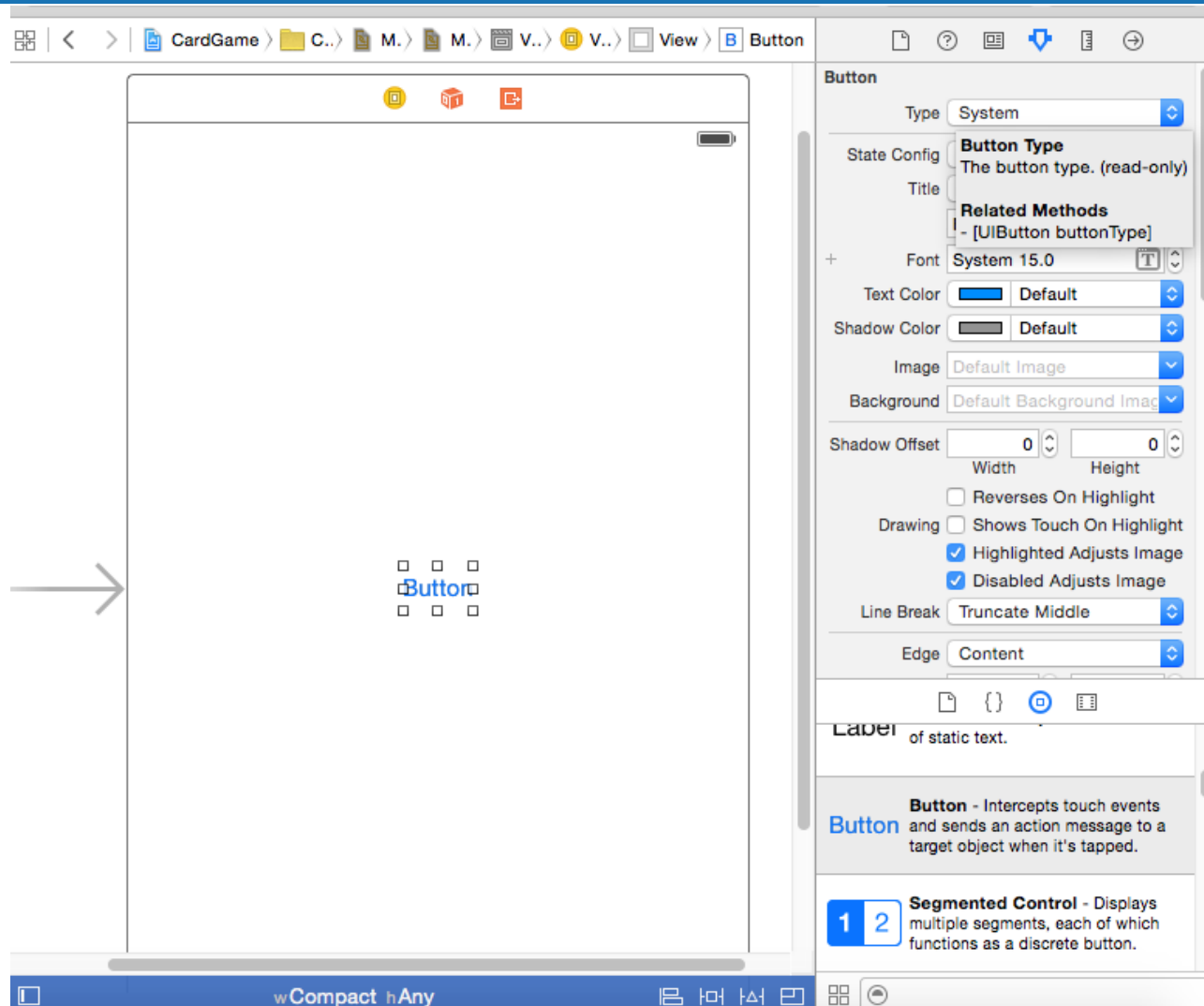
Widths: 4 Tab 4 Indent

☒ Wrap lines

No Matches







Code

- Existing code belongs to the lifecycle management of the application
 - viewDidLoad
 - didReceiveMemoryWarning
- Write a code which changes the properties of the button
 - To do so drag the button to the code editor
 - And hold the CTRL button
 - Then the connection between the storyboard and the code is made
 - Xcode indicates that

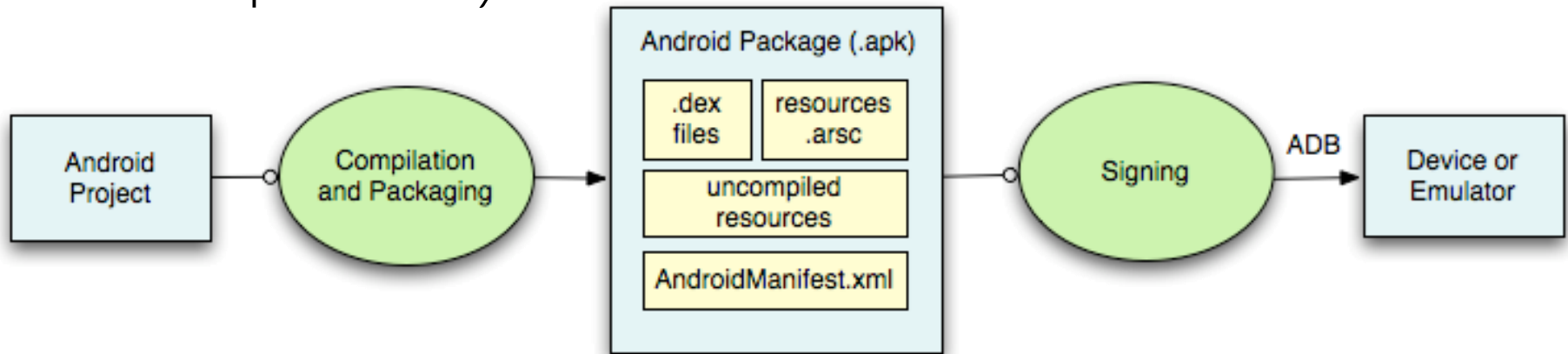


Pázmány Péter Catholic University
Faculty of Information Technology and Bionics

Android

Android project cycle

- Android applications are developed in Java language (most of the cases, so there are other possibilities)
- Source code is compiled to byte code, which is transformed to Dalvik Executable, which can be interpreted by Android VM (ART)
- Resources (images, layouts, etc.) are compiled
- All of above is packaged to the APK file, with the AndroidManifest.xml
- APK is signed digitally, and then can be installed on device or emulator
- Previous steps are done by the IDE



Android development tools

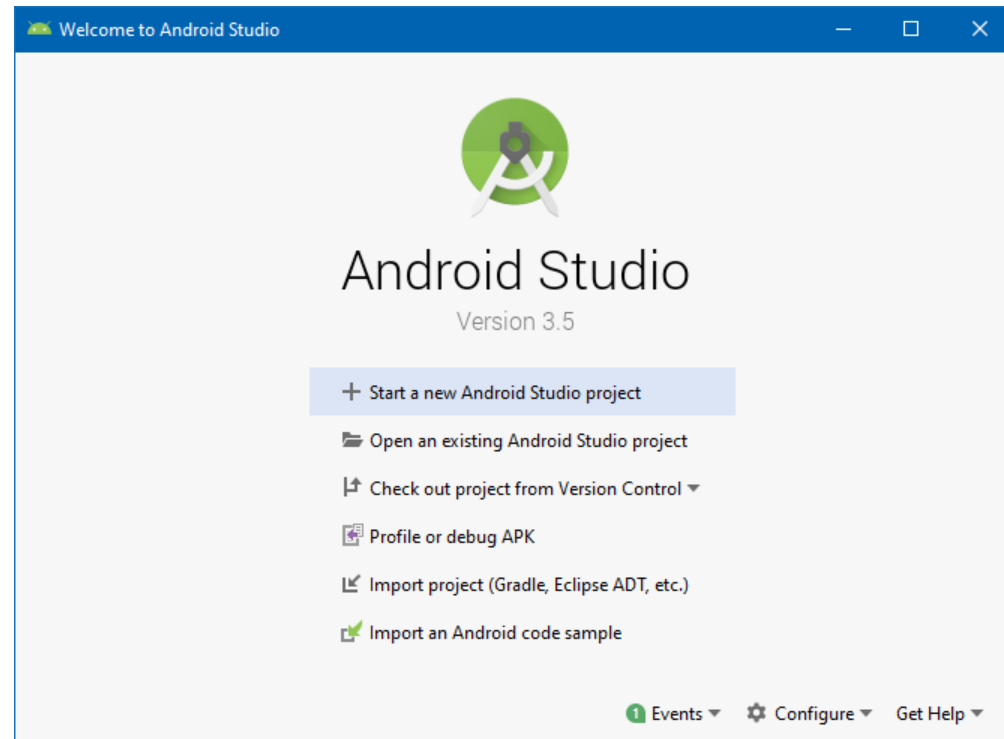
- Android Studio
 - Java SDK
- Android SDK
 - Compiler and program libraries
 - Emulator
 - This is an emulator, where executes the entire Android operating system over your OS on your device
- Android NDK
 - For native (C/C++) libraries
 - Currently you do not need it
- As a result we can develop Android software on any of the main desktop platforms
 - Even on Android: AIDE

Android Debug Bridge (ADB)

- For communication between the PC and device
- Client-server architecture
 - Client
 - Command line application, running on the developer's computer
 - IDE's hide it
 - Server
 - Manages the communication between the client and the daemon
 - Daemon thread
 - Background thread on the device or emulator
- Steps
 - Starting client
 - Server listens on 5037 TCP port
 - Server sets up the link between the client and daemon
 - TCP ports between 5555-5585
 - Two ports for each connection
 - Even for console, Odd for ADB
 - Emulator 1, console: 5554
Emulator 1, adb: 5555
Emulator 2, console: 5556
Emulator 2, adb: 5557 ...

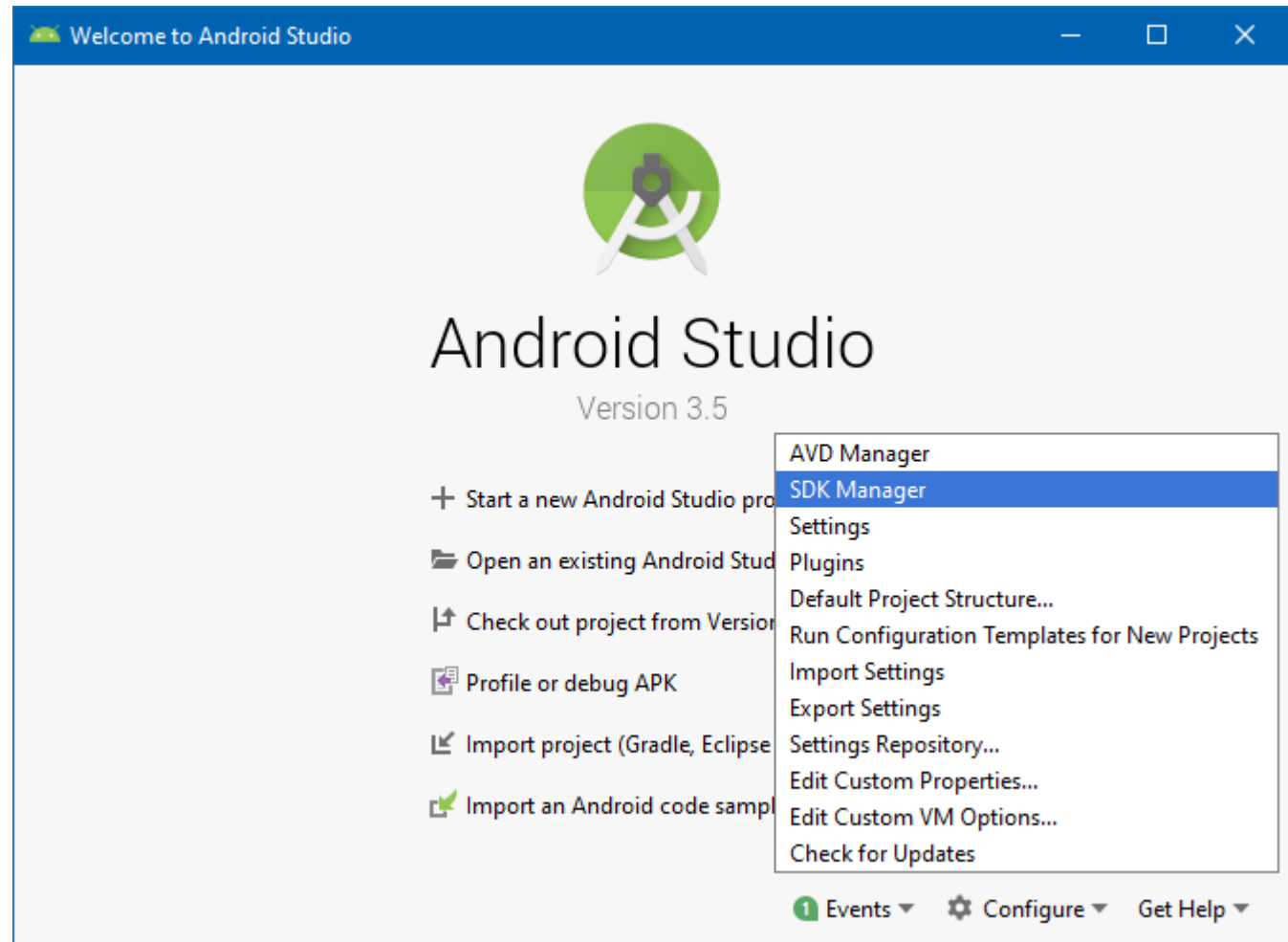
Obtaining development environment

- Java SDK
 - Install as usual
- Android Studio
 - <http://developer.android.com/sdk/index.html>
- Start

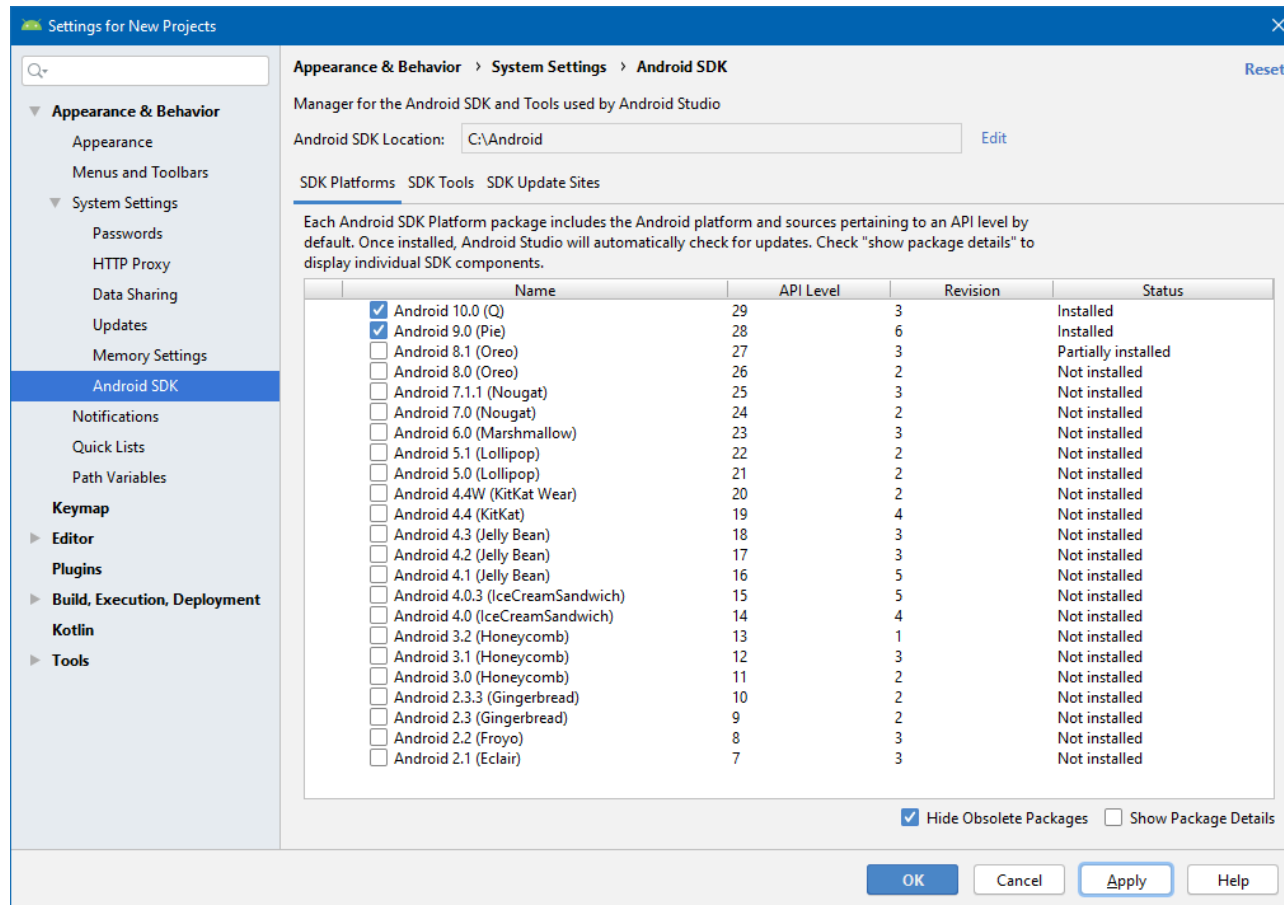


Settings

- SDK settings



SDK

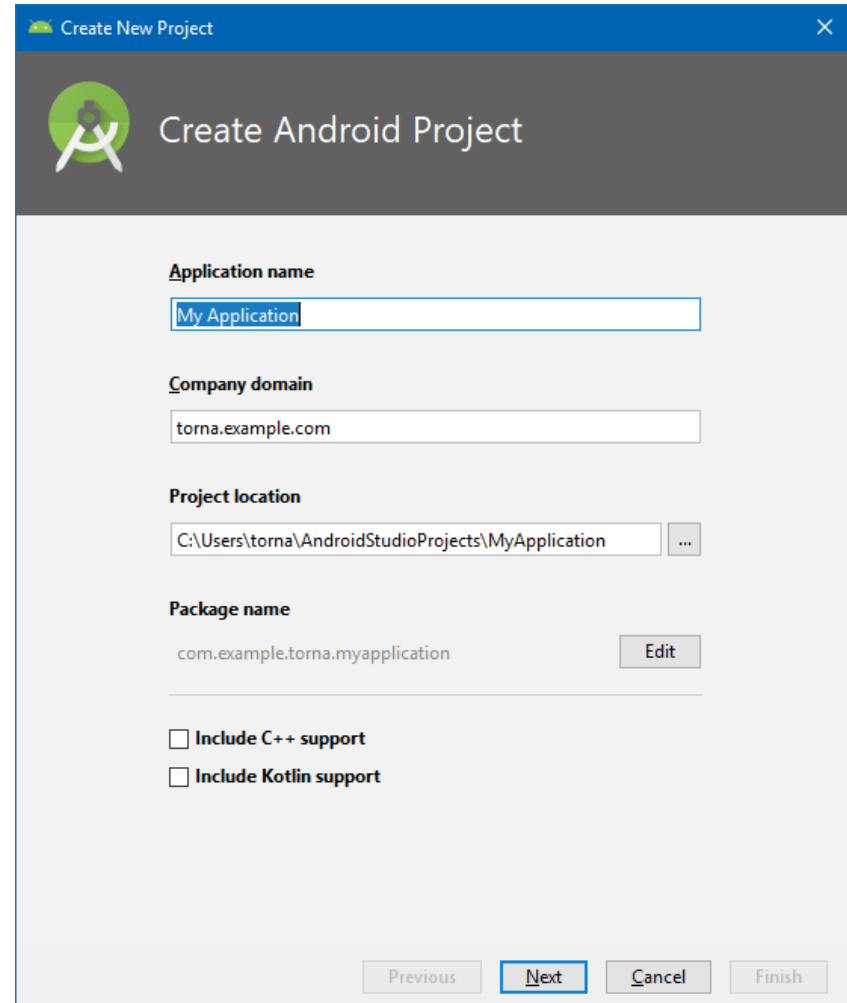


Setup

- You should install (some of them are already installed)
 - „Tools” folder
 - Android SDK Tools
 - Android SDK Platform-tools
 - Android SDK Build-tools
 - „Android O” (10.0) folder (older versions are also supported)
 - SDK Platform
 - Intel x86 Atom System Image
 - Sources for Android SDK
 - „Extras” folder
 - Android Support Library
 - Google Play Services
 - Google USB Driver (Windows)
 - Intel x86 Emulator Accelerator (HAXM Installer) – if you have appropriate Intel CPU

Hello Android

- Let's create a new Android app
- On welcome screen choose „Start new Android Studio project”
- Set the name of the project
- Set the company domain
 - Package name is generated
 - Package name should be unique
 - mad.itk.ppke.hu



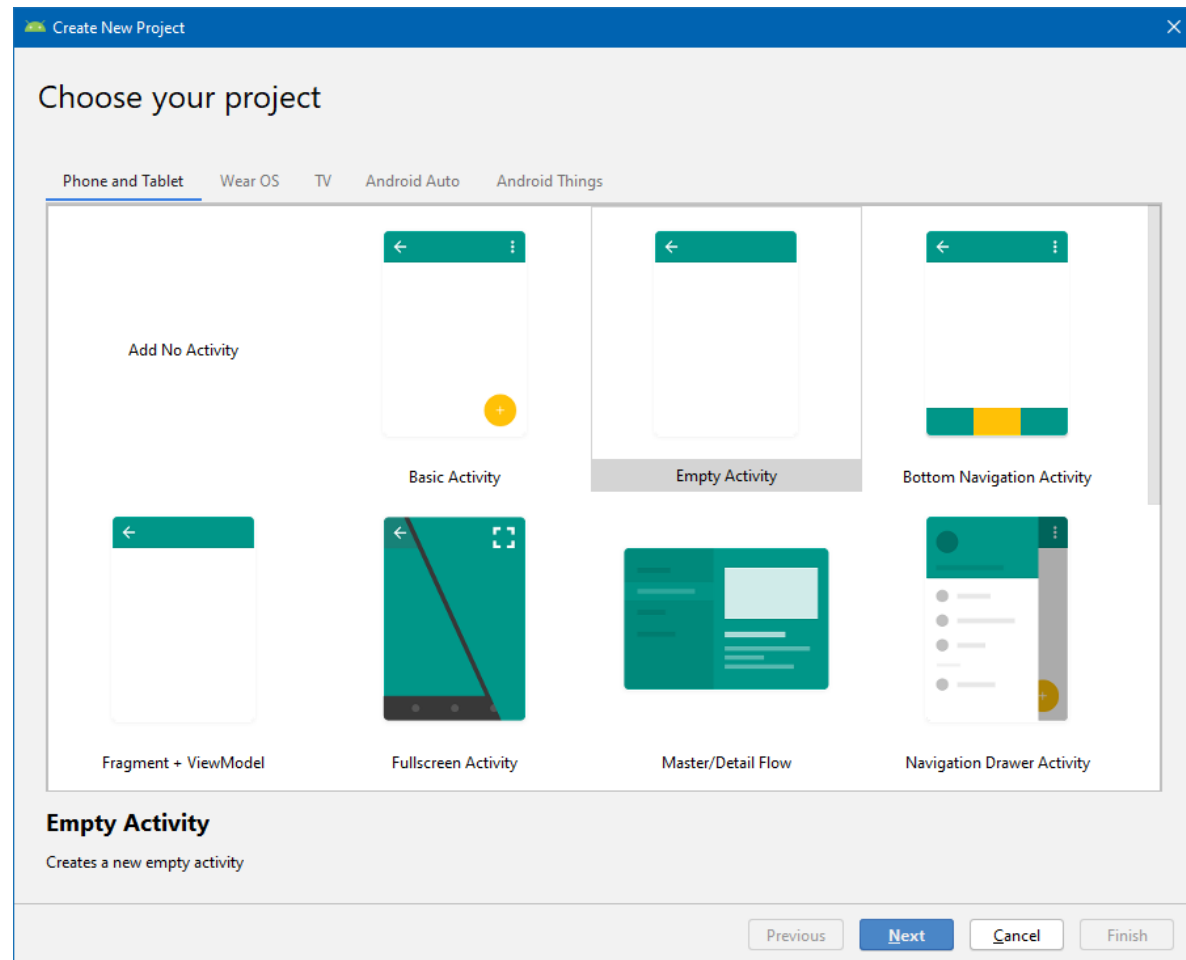
The screenshot shows the 'Create New Project' dialog box in Android Studio. The dialog has a title bar with the text 'Create New Project' and a close button. Below the title bar is a header section with the Android logo and the text 'Create Android Project'. The main area contains several input fields and checkboxes:

- Application name:** A text field containing 'My Application'.
- Company domain:** A text field containing 'torna.example.com'.
- Project location:** A text field containing 'C:\Users\torna\AndroidStudioProjects\MyApplication' with a browse button (three dots) to its right.
- Package name:** A text field containing 'com.example.torna.myapplication' with an 'Edit' button to its right.
- Include C++ support:** A checkbox that is currently unchecked.
- Include Kotlin support:** A checkbox that is currently unchecked.

At the bottom of the dialog are four buttons: 'Previous', 'Next' (which is highlighted with a blue border), 'Cancel', and 'Finish'.

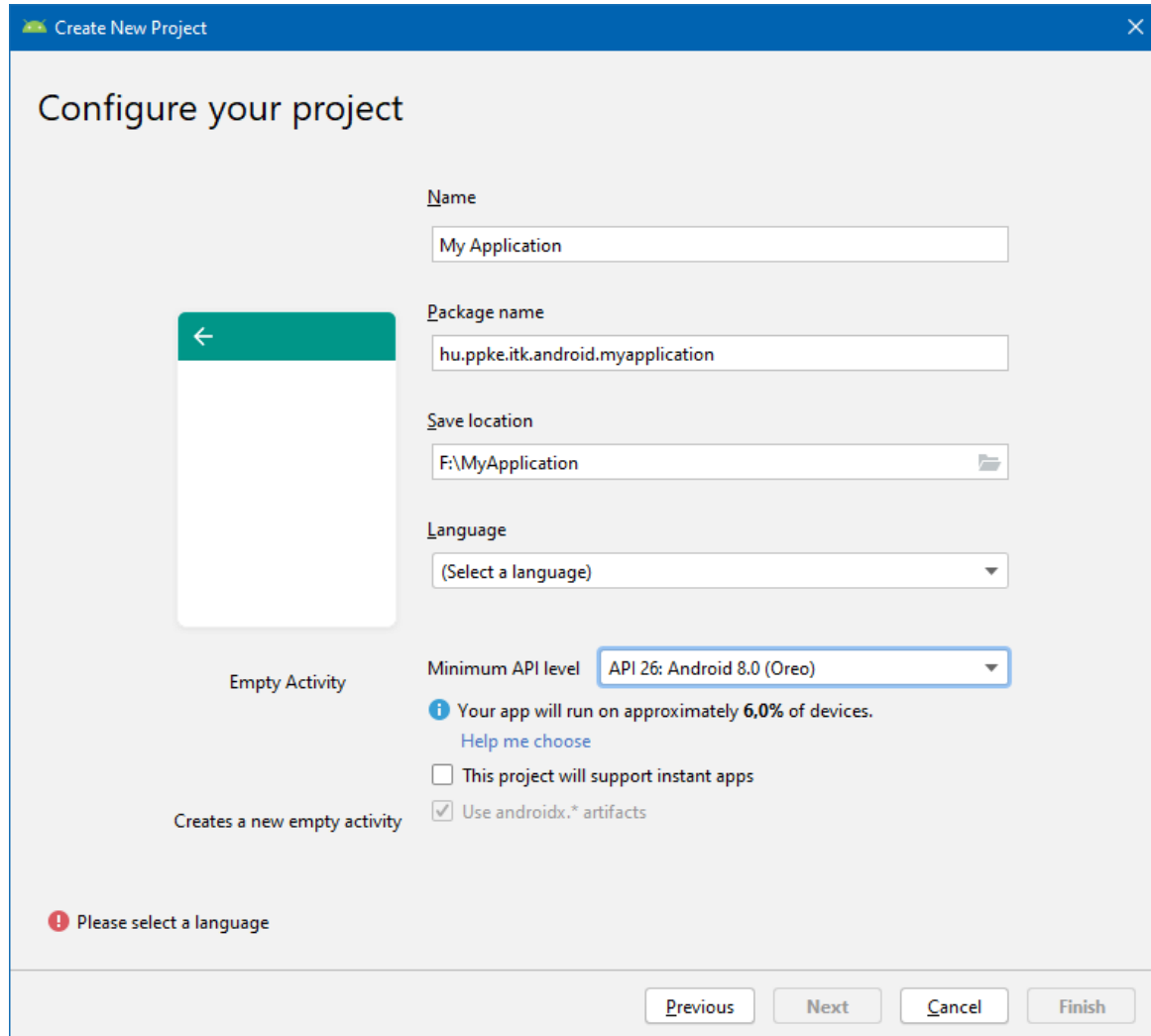
Hello Android

- Let's create a new Android app



Hello Android

- Minimum SDK
- Language!



The screenshot shows the 'Create New Project' dialog in Android Studio. The title bar says 'Create New Project'. The main heading is 'Configure your project'. On the left, there is a preview of an 'Empty Activity' with a green header bar containing a white back arrow. Below the preview, it says 'Empty Activity' and 'Creates a new empty activity'. On the right, there are several input fields and checkboxes:

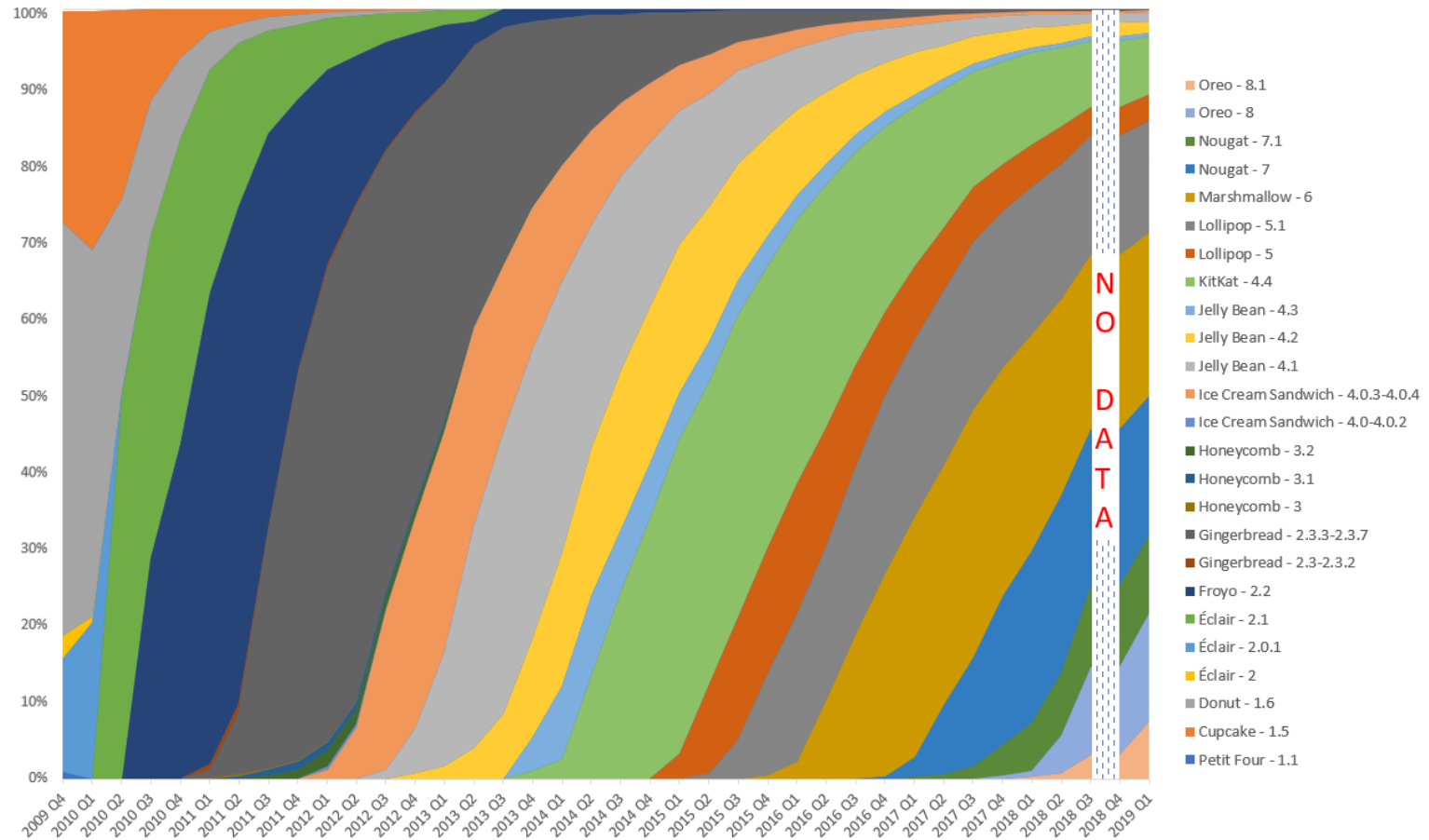
- Name:** My Application
- Package name:** hu.ppke.itk.android.myapplication
- Save location:** F:\MyApplication
- Language:** (Select a language) [dropdown arrow]
- Minimum API level:** API 26: Android 8.0 (Oreo) [dropdown arrow]
- Information:** Your app will run on approximately 6,0% of devices. [Help me choose](#)
- Checkboxes:**
 - ☐ This project will support instant apps
 - ☒ Use androidx.* artifacts

At the bottom left, there is a red warning icon and the text 'Please select a language'. At the bottom right, there are four buttons: 'Previous', 'Next', 'Cancel', and 'Finish'.

Hello Android

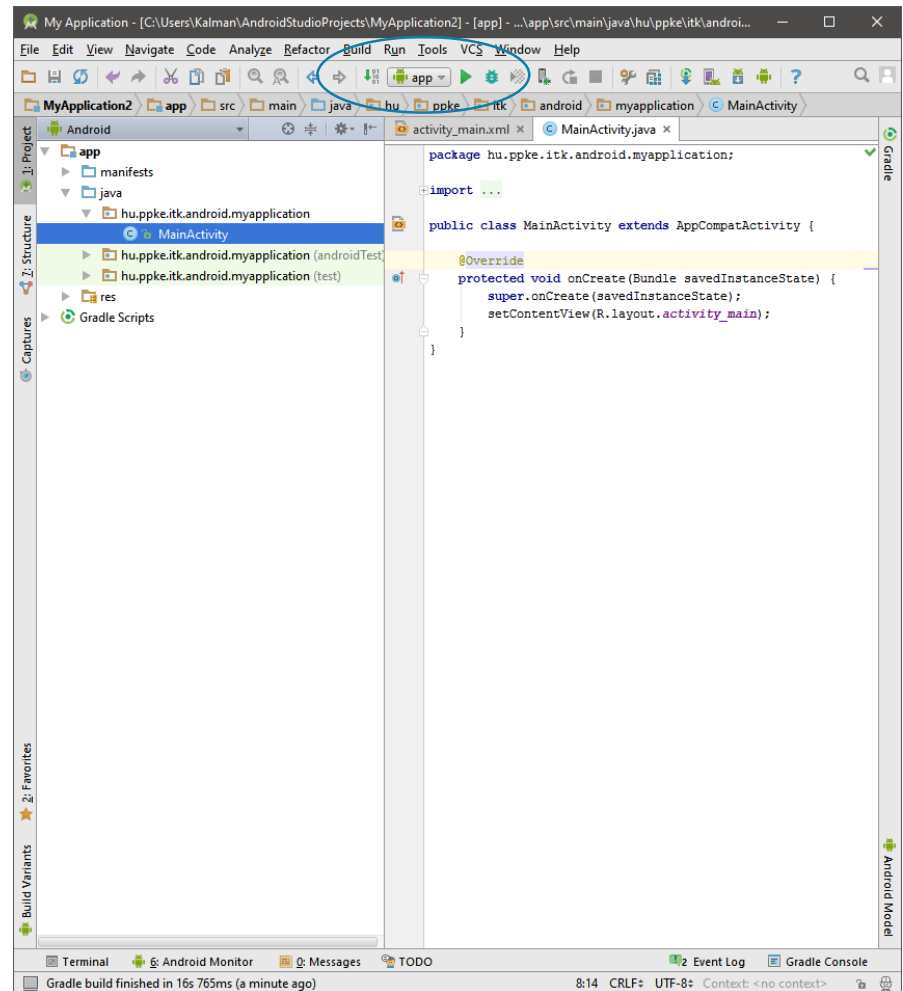
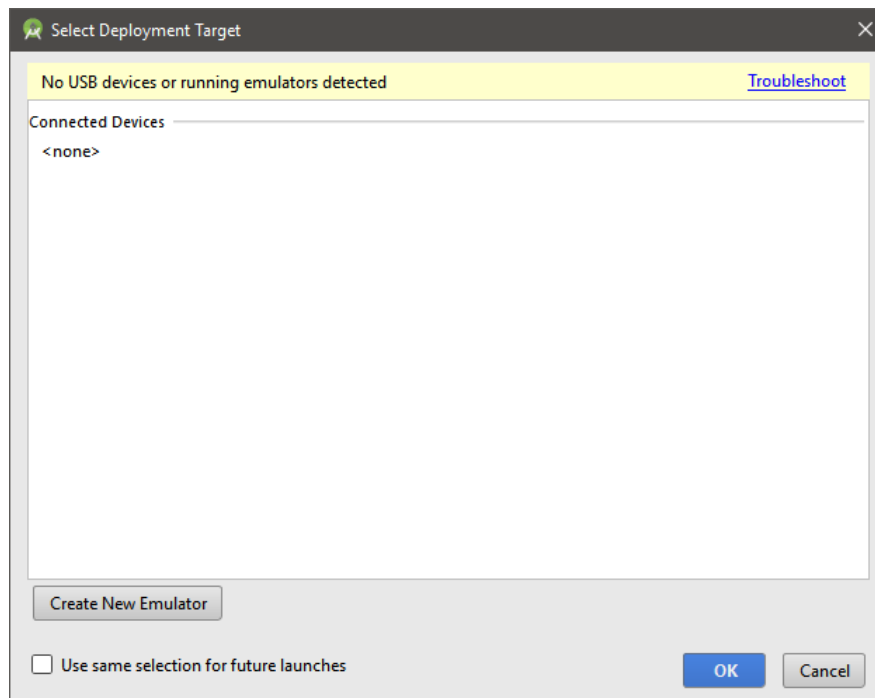
- Minimum SDK version: the oldest Android version, which is supported by the app
 - If it is too low, many of new API components cannot be used
 - If it is too high, only a few device will be supported
- Target SDK version: which capabilities wanted to be utilized
 - You should choose the latest one
- Compile with: which used for compilation
 - You should choose the latest one as well

Spread of different versions



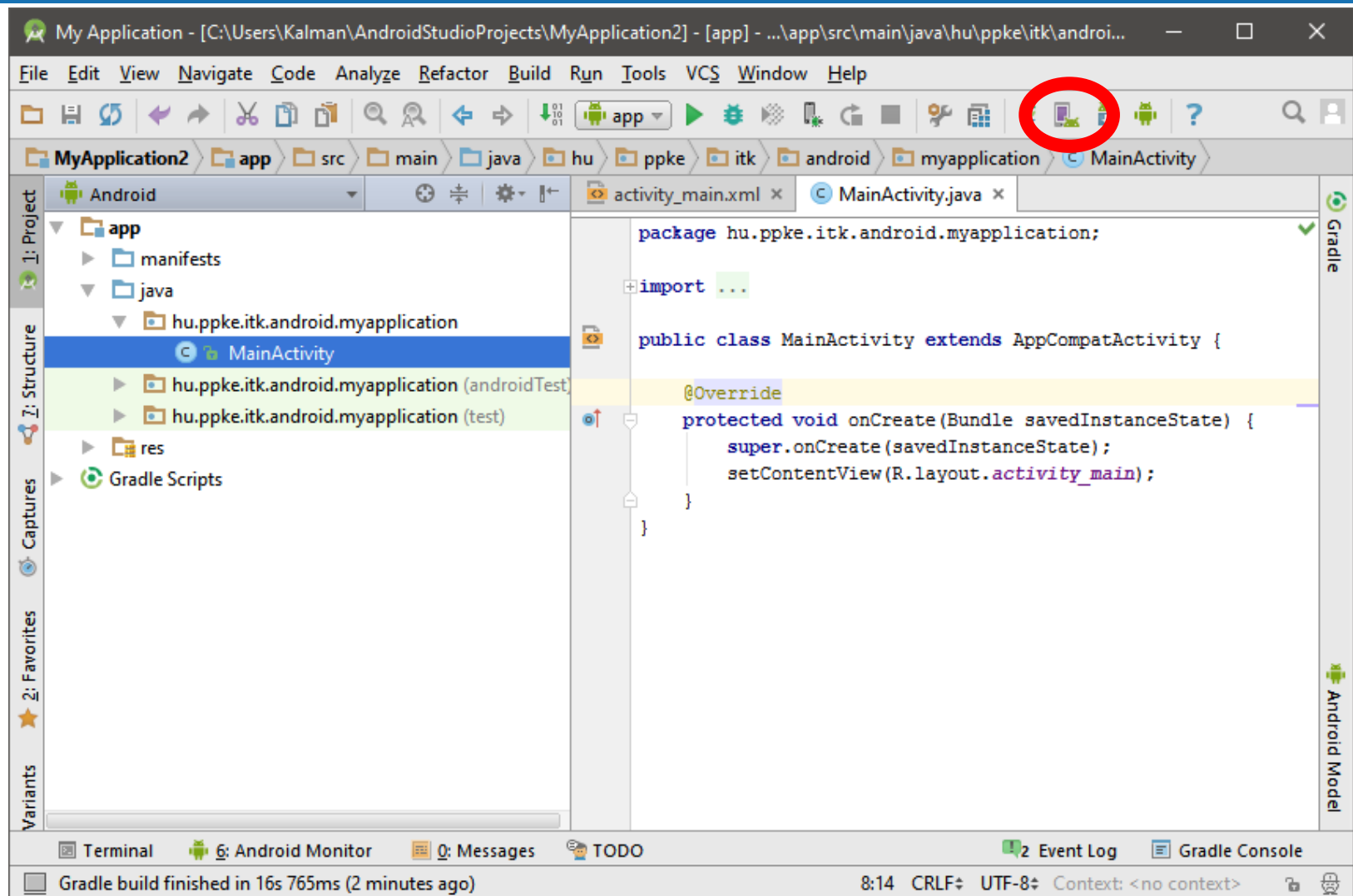
Hello Android

- Compile, install and execute the application



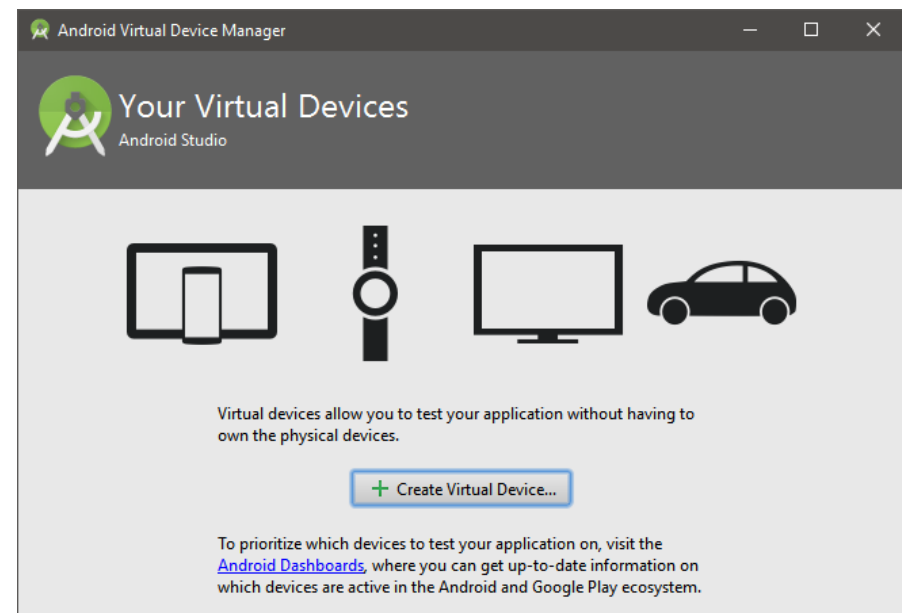
What is required?

- Emulator
 - A mobile phone can be emulated
 - Realistic, even location with GPS coordinates can be simulated
 - And AVD have to be configured, and we have to select the device to be started
 - AVD (Android Virtual Device)
 - Virtual devices can be created and its properties can be set
 - Screen size, SD size, etc.
- Device
 - Physical device



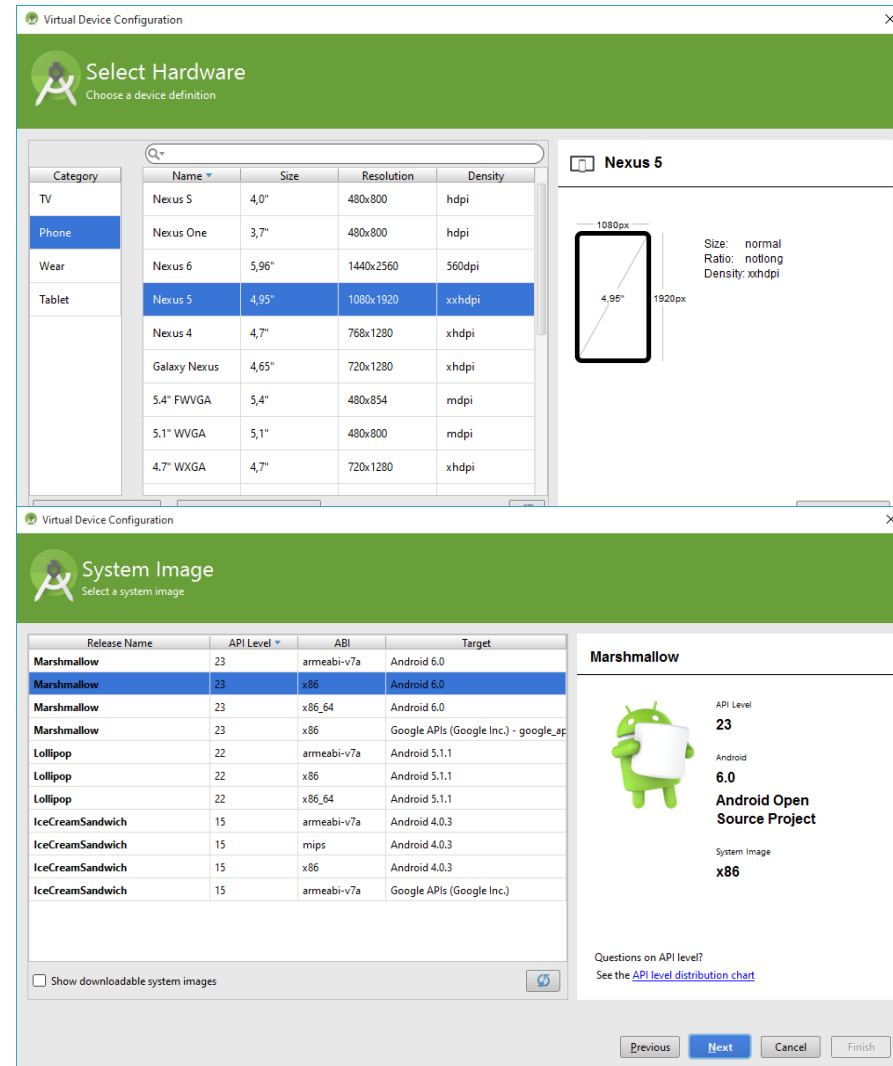
Setting an emulator

- AVD Manger
 - A virtual device can be
 - created
 - deleted
 - started
 - modified
- Each device has an „disk” image, which is used by the emulator
 - Thus we have persistent storage

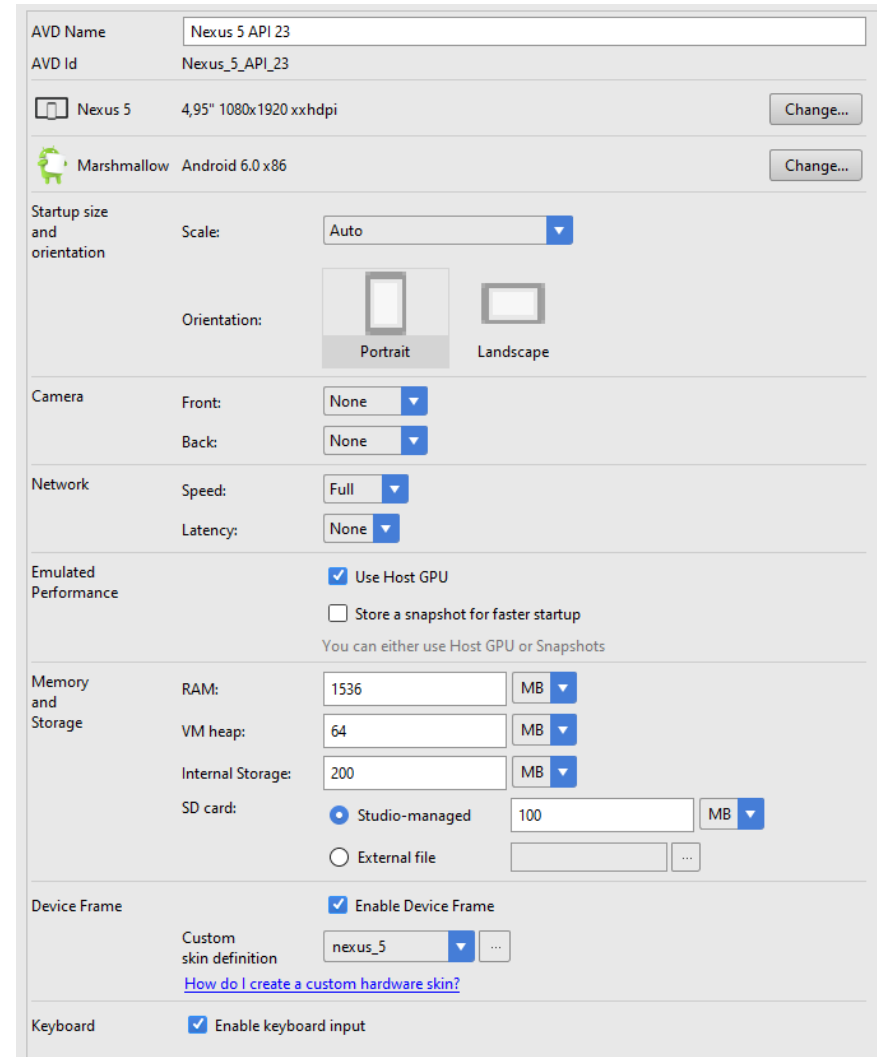


Setting an emulator


- Create a new one
 - Device: Pixel
 - Set the previously downloaded Android version
 - 10.0
 - Architecture x86
 - With Google API






Advanced settings



AVD Name: Nexus 5 API 23
AVD Id: Nexus_5_API_23

 Nexus 5 4,95" 1080x1920 xxhdpi Change...

 Marshmallow Android 6.0 x86 Change...

Startup size and orientation
Scale: Auto ▼
Orientation:  Portrait  Landscape

Camera
Front: None ▼
Back: None ▼

Network
Speed: Full ▼
Latency: None ▼

Emulated Performance
☒ Use Host GPU
☐ Store a snapshot for faster startup
You can either use Host GPU or Snapshots

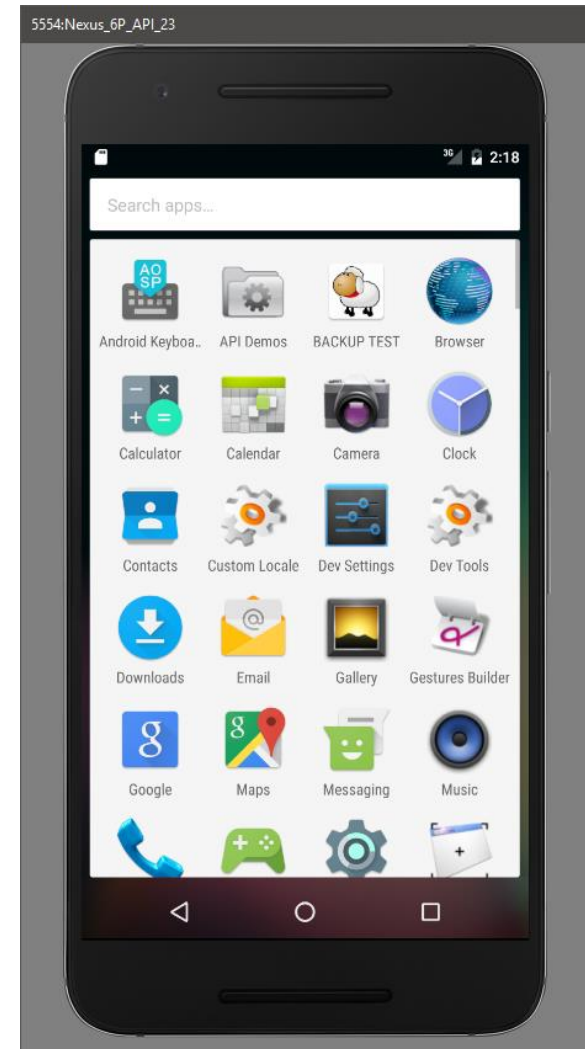
Memory and Storage
RAM: 1536 MB ▼
VM heap: 64 MB ▼
Internal Storage: 200 MB ▼
SD card: ☒ Studio-managed 100 MB ▼
☐ External file ...

Device Frame
☒ Enable Device Frame
Custom skin definition: nexus_5 ▼ ...
[How do I create a custom hardware skin?](#)

Keyboard
☒ Enable keyboard input

Emulator settings

- Optional: if you have an appropriate Intel processor, which supports virtualization, install HAXM
 - As previously discussed
 - Enable it in BIOS / EFI
 - It would really speed up the emulator
- Emulator can be started by pressing the green „Play” button



Emulator

- On the emulator image there is a complete system
 - Most of things are working
 - Events can be simulated
 - Network type
 - Incoming call
 - Incoming SMS
 - GPS coordinate
- Android device monitor
 - Tools | Android | ADM

Events

Extended controls

Location

Cellular

Battery

Phone

Directional pad

Fingerprint

Settings

Help

GPS data point

☒ Decimal

☐ Sexagesimal

Latitude
37.422

Longitude
-122.084

Altitude (meters)
0.0

SEND

GPS data playback

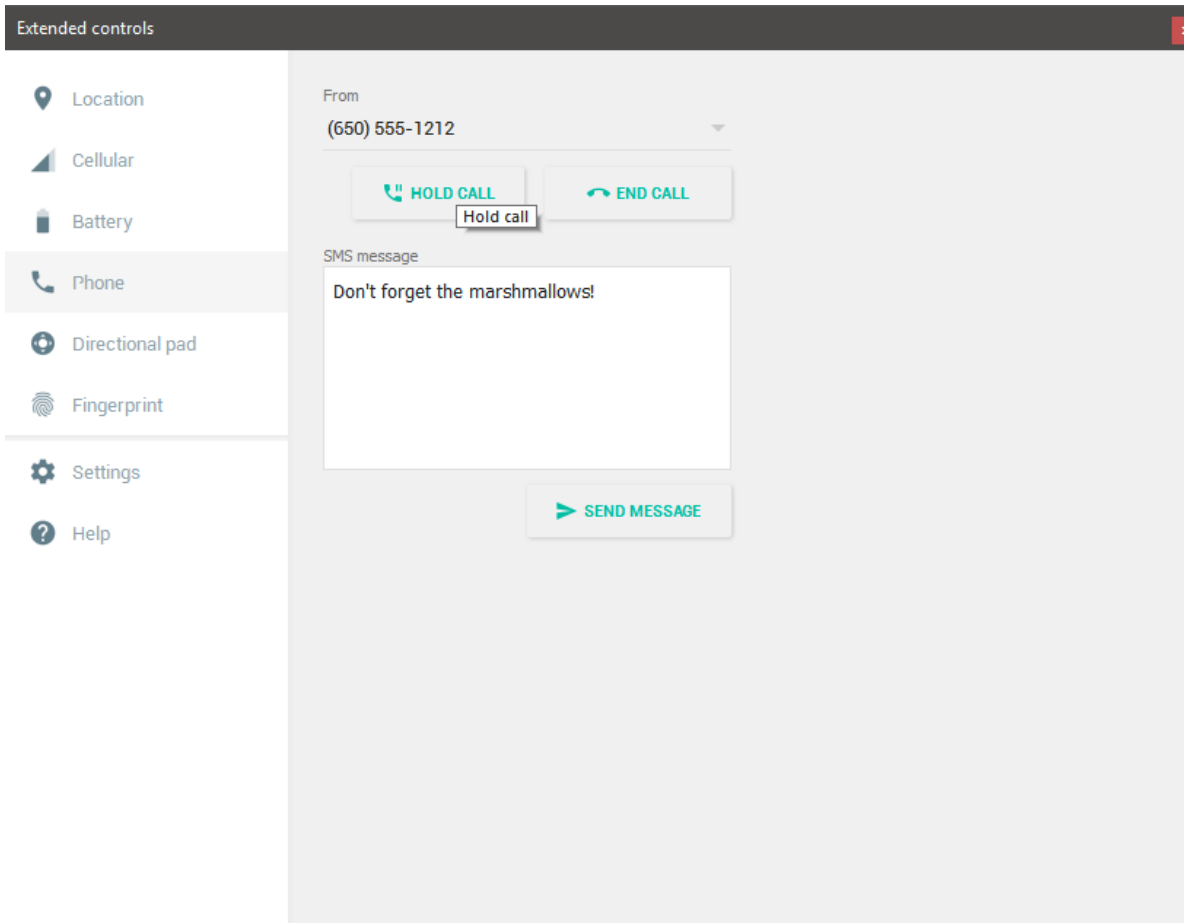
Delay (sec)	Latitude	Longitude	Elevation	Name	Description
-------------	----------	-----------	-----------	------	-------------

▶

Speed 1X

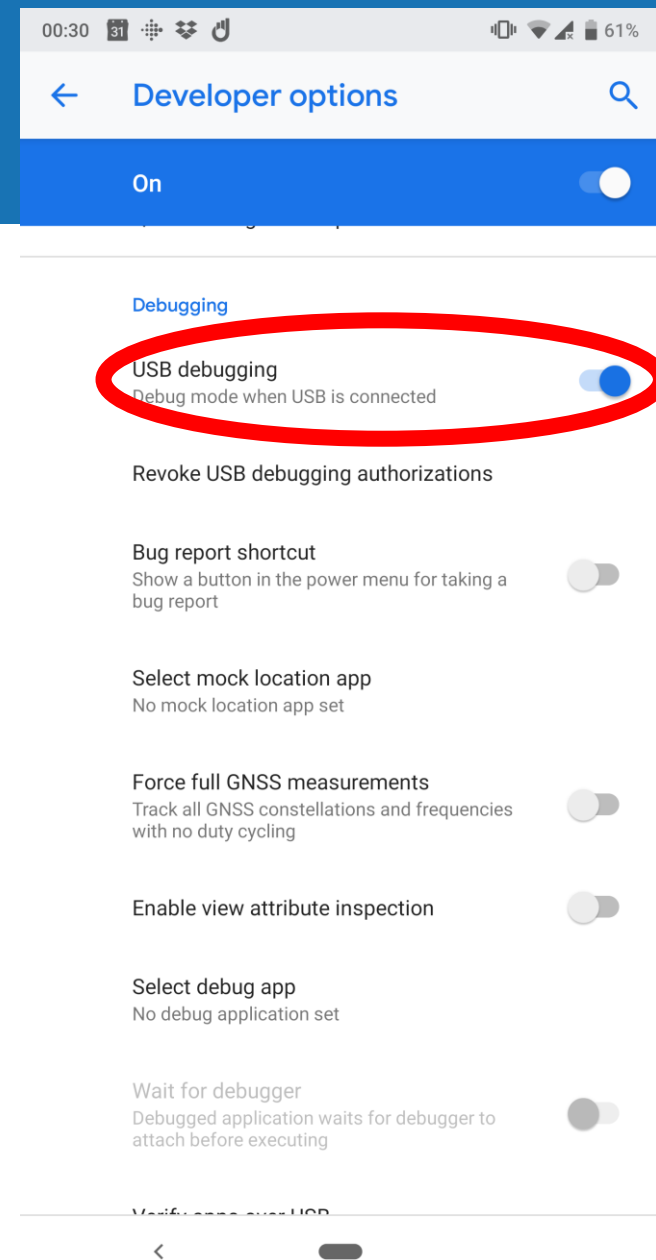
LOAD GPX/KML

Emulator



Physical device

- Enabling developer mode
 - Settings | About Phone | Build number
 - Press it eight times
 - Then enable USB debugging
 - Settings / Developer settings
- Set up drivers on Windows
 - There is an universal driver available (Google USB driver)
 - It does not work with all devices
 - Linux and MacOS almost always recognize devices



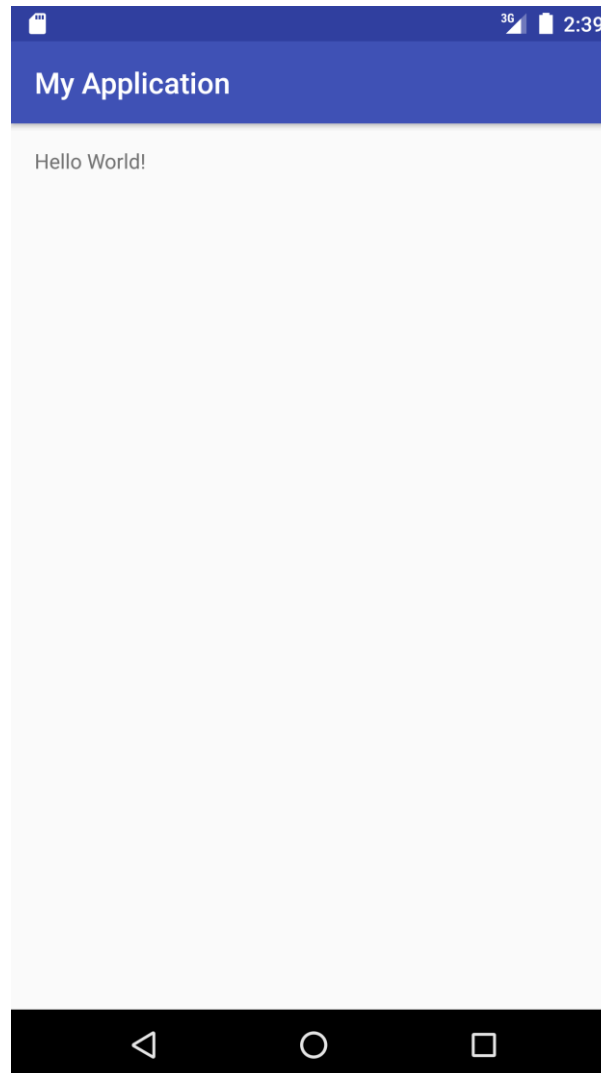
Physical device

- You have to accept the certificate of the PC
 - A pop-up windows appears to do so
- Once everything is done then a notification is sent
- In Android Studio, the device is listed as online devices
 - You can used only online devices
 - If there is a problem you should pull out and plugin
 - Its a miracle
 - TO check devices, type in command line
 - adb devices
 - ADB is located at
 - <ANDROID SDK>\platform-tools\adb

Result

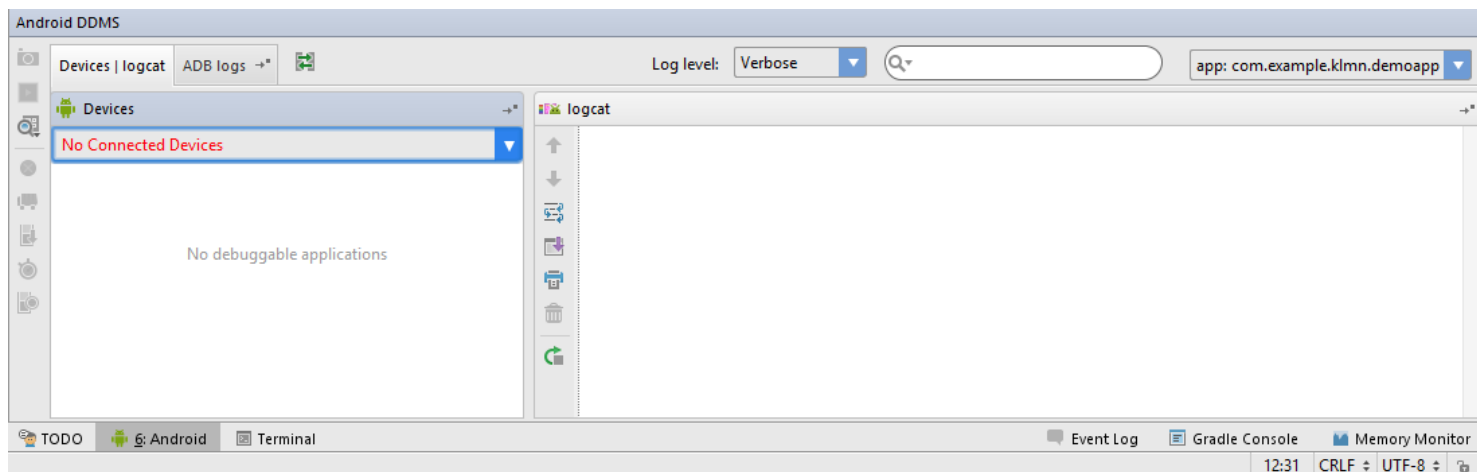


Result



LogCat

- Informative messages can be sent to the console of the PC
- Use the static functions of the `android.util.Log` class
 - `Log.i("MainActivity", "Hello logging!");` // information log
 - First parameter: label – you may want to write the classname here
 - Second parameter: message
- In Android Studio press `Alt + 6` to open the console



Tools

```
Parancssor
C:\Android>cd tools

C:\Android\tools>ls
'ls' is not recognized as an internal or external command,
operable program or batch file.

C:\Android\tools>dir
Volume in drive C is Rendszer
Volume Serial Number is 4AEE-91C7

Directory of C:\Android\tools

2018. 09. 12. 23:19 <DIR>      .
2018. 09. 12. 23:19 <DIR>      ..
2018. 09. 12. 23:19          5 778 android.bat
2018. 09. 12. 23:19 <DIR>      bin
2018. 09. 12. 23:19        636 928 emulator-check.exe
2018. 09. 12. 23:19        809 984 emulator.exe
2018. 09. 12. 23:19 <DIR>      lib
2018. 09. 12. 23:19        239 821 mksdcard.exe
2018. 09. 12. 23:19          947 monitor.bat
2018. 09. 12. 23:19        829 319 NOTICE.txt
2018. 09. 12. 23:19         17 372 package.xml
2018. 09. 12. 23:19 <DIR>      proguard
2018. 09. 12. 23:19          138 source.properties
2018. 09. 12. 23:19 <DIR>      support
                        8 File(s)        2 540 287 bytes
                        6 Dir(s)   185 144 782 848 bytes free

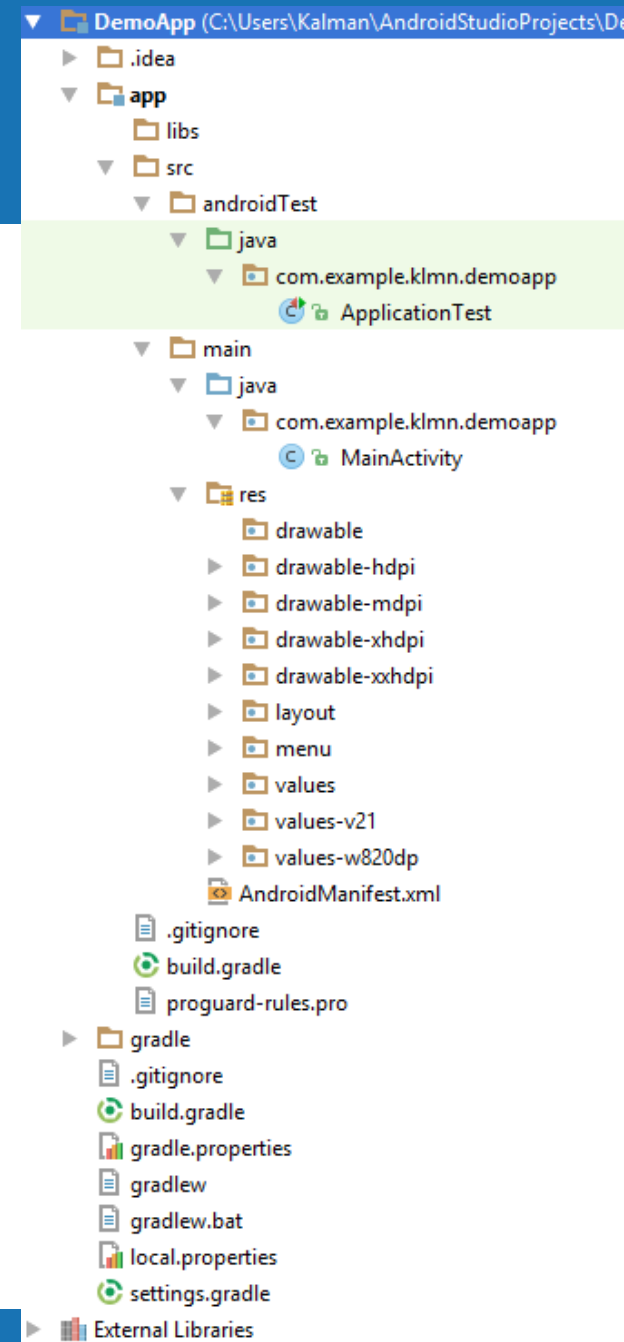
C:\Android\tools>
```

Tools

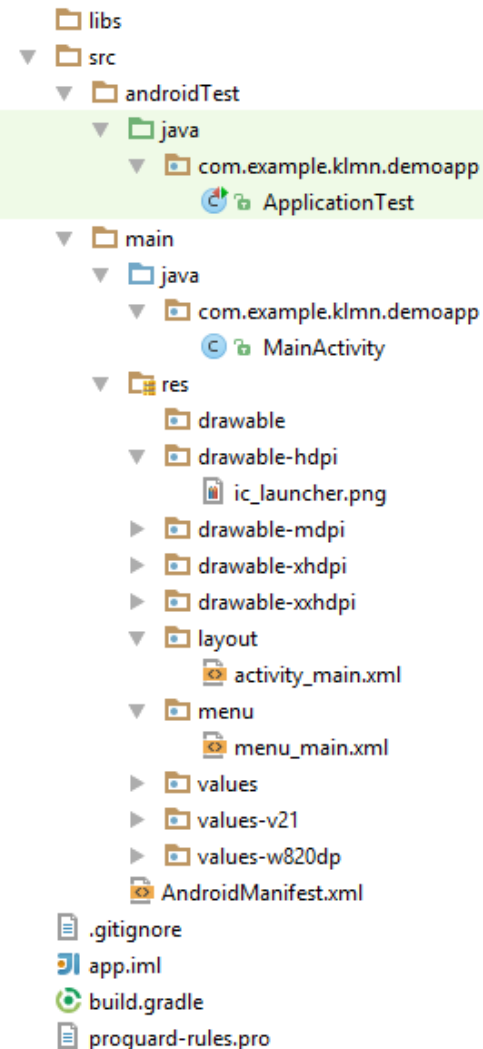
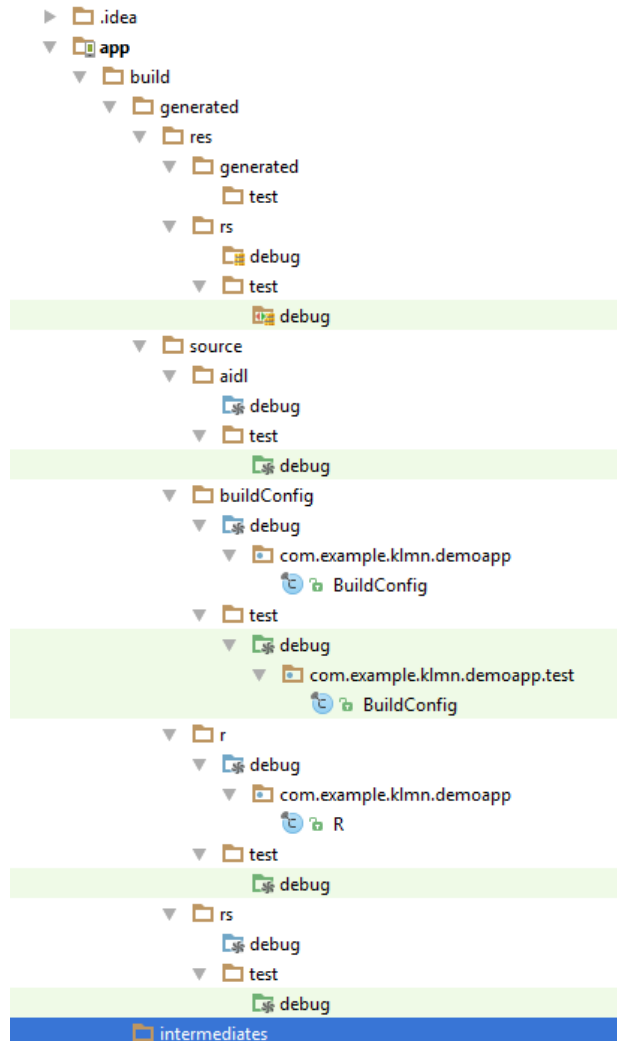
```
Parancssor - adb shell
file_contexts
firmware
fsg
fstab.shamu
init
init.environ.rc
init.mmi.touch.sh
init.rc
init.shamu.diag.rc
init.shamu.power.rc
init.shamu.rc
init.shamu.usb.rc
init.trace.rc
init.usb.rc
init.zygote32.rc
mnt
oem
persist
proc
property_contexts
res
root
sbin
sdcard
seapp_contexts
selinux_version
sepolicy
service_contexts
storage
sys
system
tombstones
ueventd.rc
ueventd.shamu.rc
vendor
verity_key
shell@shamu:/ $
```


Project structure

- .idea
 - IntelliJ IDEA settings
- app
 - Files of Android applications
- build
 - Files generated during build
- gradle
 - Location of gradle wrapper
- build.gradle
 - Project settings for Gradle building
- gradle.properties
 - Project settings for Gradle
- gradlew or gradlew.bat
 - OS specific gradle settings
- local.properties
 - Local computer specific settings
- .iml
 - IntelliJ IDEA module information
- settings.gradle
 - Gradle tool parameters



▼ DemoApp (C:\Users\Kalman\AndroidStudioProjects\DemoApp)



Project structure

- build
 - Files generated after build process – flavor and version specific
 - Several builds for different API, etc.
- libs
 - User defined libraries
- src
 - androidTest
 - For Junit tests
 - main/java/ ...
 - Java source codes
 - main/jni
 - Android NDK/JNI source codes
 - main/assets
 - Most of the cases it is empty
 - Files are put into the APK file, raw resources

Project structure

- src/main/res
 - anim
 - Animations encoded in XML
 - drawable (xdpi, hdpi, mdpi, ldpi)
 - Images (.jpg .png or .xml)
 - layout - *.xml
 - To describe UI layouts
 - raw
 - Resources: mp3, mp4, avi, CVS, etc.
 - values – strings.xml
 - Texts used in the application
 - Used for localization

Project structure – AndroidManifest

- src/main/AndroidManifest.xml
 - All important information about the application
 - Components
 - Hardware requirements
 - Android version compatibilities
 - Permissions
 - Java package name
 - The libraries that the application must be linked
- <http://developer.android.com/guide/topics/manifest/manifest-intro.html>

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.hello"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="15"
        android:targetSdkVersion="19" />

    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.CAMERA" />

    <uses-feature android:name="android.hardware.camera" />
    <uses-feature android:name="android.hardware.camera.autofocus" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".SettingsActivity" android:screenOrientation="portrait" />
        <activity android:name=".NewsActivity" android:screenOrientation="portrait" />
    </application>
</manifest>
```

<!-- Application version -->

<!-- Android 4.0 and above -->

<!-- can access internet -->
<!-- write on external storage (SD card) -->
<!-- use camera -->

<!-- requires camera -->
<!-- requires autofouces -->

<!-- icon, name and theme for app -->

<!-- main Activity (later) -->

build.gradle

```
apply plugin: 'com.android.application'
```

```
android {  
    compileSdkVersion 24  
    buildToolsVersion "24.0.3"  
  
    defaultConfig {  
        applicationId "hu.ppke.itk.mad"  
        minSdkVersion 20  
        targetSdkVersion 24  
        versionCode 1  
        versionName "1.0"  
    }  
    buildTypes {  
        release {  
            minifyEnabled false  
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'  
        }  
    }  
}
```

← Used SDK version

← Package name

← Minimum SDK version needed

← Version of the application

```
dependencies {  
    implementation fileTree(dir: 'libs', include: ['*.jar'])  
    testImplementation 'junit:junit:4.12'  
    implementation 'com.android.support:appcompat-v7:24.2.1'  
}
```

← Used libraries



Android

Next week



Pázmány Péter Catholic University
Faculty of Information Technology and Bionics

Basics of Mobile Application Development

Android Basics

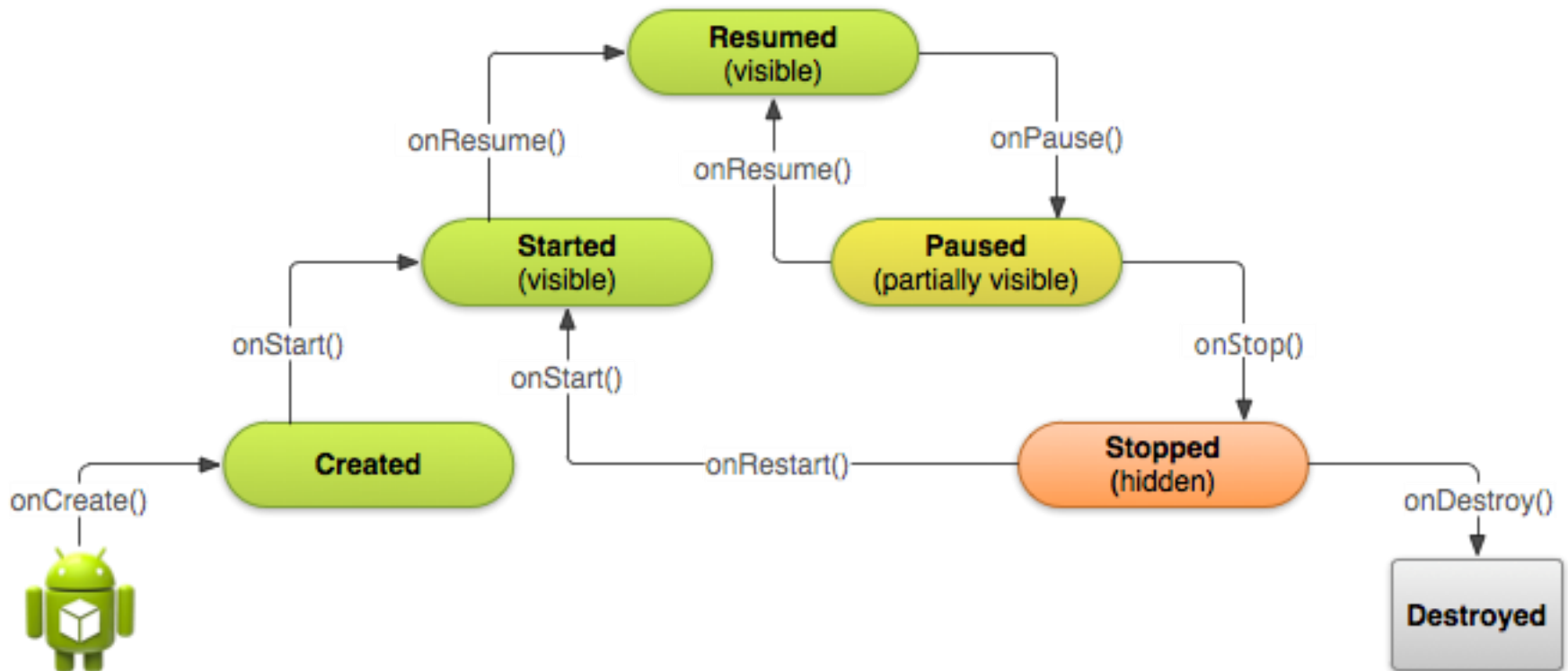
Activity

- There is no main() method!
- „An activity is a single, focused thing that the user can do. Almost all activities interact with the user, so the Activity class takes care of creating a window for you in which you can place your UI with setContentView(View)”
 - So Activities represents a page/window where the user can interact with the application.
 - Only one activity can be active at the time!
 - User can navigate between the Activities with system buttons
 - Back button destroys the current activity and returns to the previous one
 - Home button stops the current activity and returns to the launcher activity
 - Overview button stops(/pauses) current activity and can switch between recent activities

Activity

- Purposes
 - Communicate with the user
 - Handle GUI elements
 - Execute tasks
- An application can have multiple activities
- All activity is derived from `android.app.Activity` class

Activity life cycle



Activity life cycle – methods

- We are informed about the status changes of **Activity** with several different callback functions
 - We have to override these methods, and these methods are called by the system
 - Then we can execute tasks when events occur
- The life cycle functions are:
 - **onCreate**: when **Activity** starts newly (first start, or after disposal)
 - You may set the GUI and variables here
 - **onStart**: when the **Activity** is visible for the user
 - **onResume**: the **Activity** is in focus, now we can start working
 - **onPause**: when **Activity** is partially visible
 - Due to other **Activity**, or **Dialog**, ...
 - In case of multi windows system (Android 7.x) when this is the inactive **Activity**
 - You may want to save the necessary information (state)
 - This have to be quick, as it blocks any other **Activity**.
 - If the **Activity** is being destroys this is the only function which execution is guaranteed!

Activity life cycle – methods

- **onStop**: when the **Activity** is invisible
 - It is totally invisible due to another **Activity**, or any other reason
 - Incoming call
 - Screen lock
- **onDestroy**: when **finish()** is called, or memory is needed
 - The **Activity** is destroyed (killed, deleted, ...)
 - If the memory is needed instantly then this call may be discarded.
 - Do not save data here, only set the affected variables to **null**
- In all life cycle callback method you have to call the superclass' same method
 - Example: **super.onCreate**
 - The Android system check it
 - Runtime Exception is thrown if you violate this rule

Screen layouts

- You can define the screens two ways
 - Static method
 - Creating .xml files in the res/layout folder
 - Dynamic method
 - In the java source code
 - Creating new instances of View elements
- The layout defines the positions, sizes of elements in the screen
- A layout class is derived from the View class!

Attributes of GUI elements

- `layout_width` and `layout_height`
 - Specify the width and height of the view element or layout
 - It is required to specify
 - Runtime exception is thrown if it is missing
 - The actual size is calculated (based on this value and other elements)
- Possible values
 - `wrap_content` – as the content requires
 - `match_parent` – the size of this element is specified by the parent
 - fix size – the unit is `dp`, which is the devices independent pixel
- `id`: optional (you have to specify if you wish to access it from `Activity`)
- `gravity`: the view is aligned
 - `left`, `right`, `bottom`
 - `center` – vertical and horizontal
 - `horizontal`, `vertical`
 - You can mix: `android:gravity="center|bottom"`

Attributes of GUI elements

- `layout_weight="2"`
 - The „importance” of the element can be set
 - More important element can „push” aside the other elements
 - There are three views but the middle should be larger
- `visibility`:
 - visible – you can see it, visible
 - invisible – cannot be seen, but its size is considered
 - gone – cannot be seen, and no space is occupied
- `padding`
 - Space between the elements
- `background`
 - Could be a color or drawing
- There are attributes which are depending on the actual class of the parent `ViewGroup`
 - For example: the column of a table can be interpreted only in a table

GUI elements

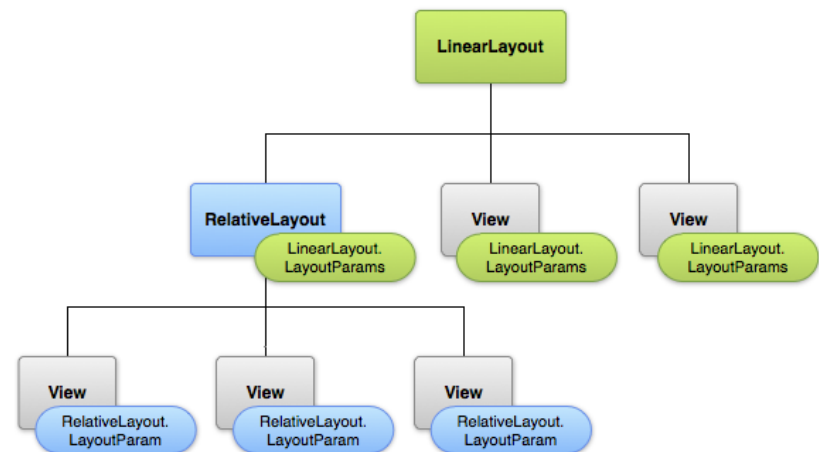
- Layouts
 - Linear Layout
 - Relative Layout
 - Constraint Layout
 - Coordinator Layout
 - RecyclerView
 - Frame Layout
 - Web View
- Widgets and other Views
 - Text View
 - Edit Text
 - Auto Complete Text View
 - Button
 - Image View
 - Scroll View
 - View Pager
 - Map View
 - etc

GUI structure

- The GUI is built from Widgets which are **View** and **ViewGroup** elements arranged in a tree structure
 - The ViewGroup is extended from the View class also
 - The **ViewGroup** is a special **View**, which can have children, so it can contain other elements
- It is possible to define own Views or View groups, but there are a lot of predefined ones.
 - If you need to create an own view extend from the proper class

View hierarchy

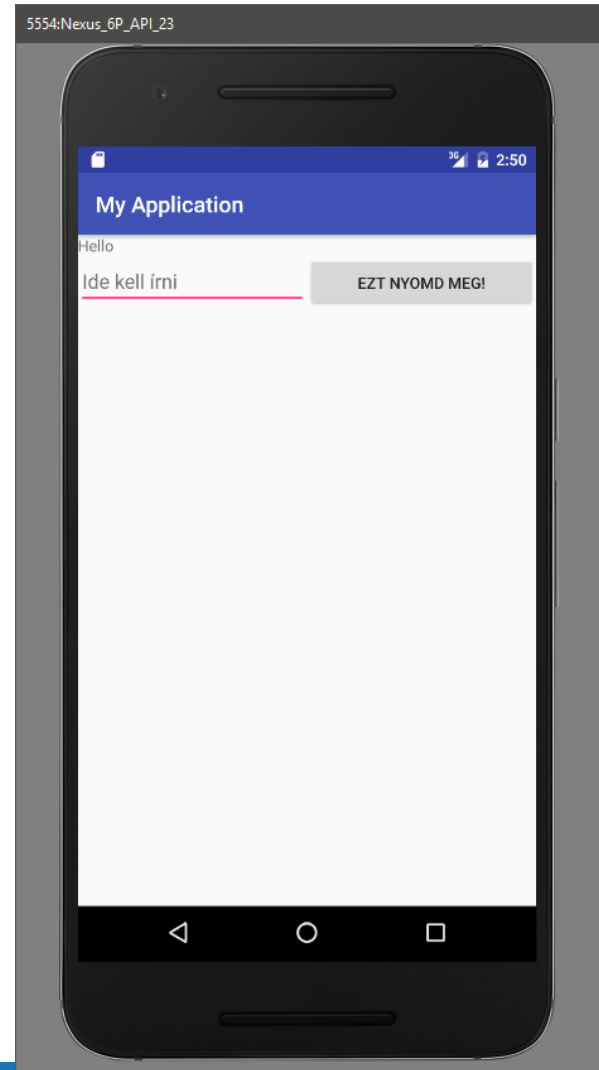
- There is one root element
- Set the root element with the setContentView function of the Activity class.
 - In the onCreate() function
- Every ViewGroup responsible for the drawing of it's children
- Views are drawn on the top of root.
- We can add child to a ViewGroup dynamically with the addView(View) function



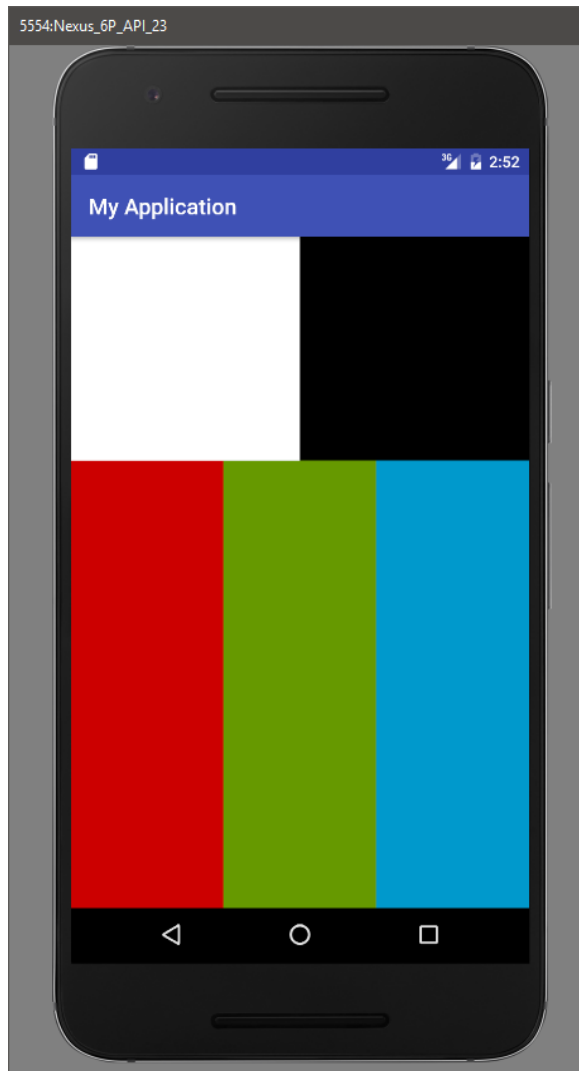
Inflation

- The hierarchy can be derived in xml files as well
 - In that case the parameter of the `setContentView` is not a `View`, but an `int`
 - This is an id for the layout file
 - The id and the xml are connected in the `R.java` file
 - The connection is automatically created
 - First, the system creates the view hierarchy based on the layout
 - Then it calls the `setContentView(View)` function
 - Example:
 - Last week's hello worlds
 - `setContentView(R.layout.activity_main);`

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text= "Hello" />
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal" >
        <EditText
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:hint="Ide kell írni" />
        <Button
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="Ezt nyomd meg!" />
    </LinearLayout>
</LinearLayout>
```



Example



Widgets

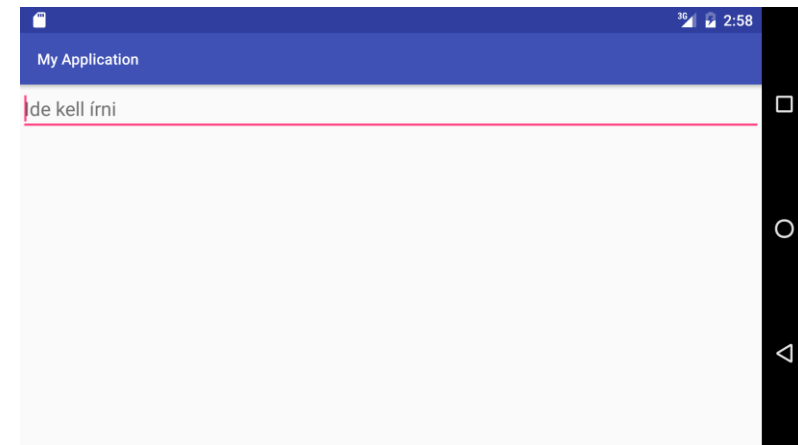
- **TextView**

- To display text
- Main attributes
 - `text` – text given
 - `textColor` – color of text
 - `textSize` – size of the text
 - `typeface` – font of the text

Widgets

• EditText

- Derived from `TextView`
- To input text
- The keyboard is shown automatically when this view gains the focus
- Important attributes
 - `inputType` – text, number, email address, etc.
 - `hint` – hint is shown before any text is added



Widgets

• Button

- Derived from `TextView`
- Differs only the default background
- Represents a button which can be pressed



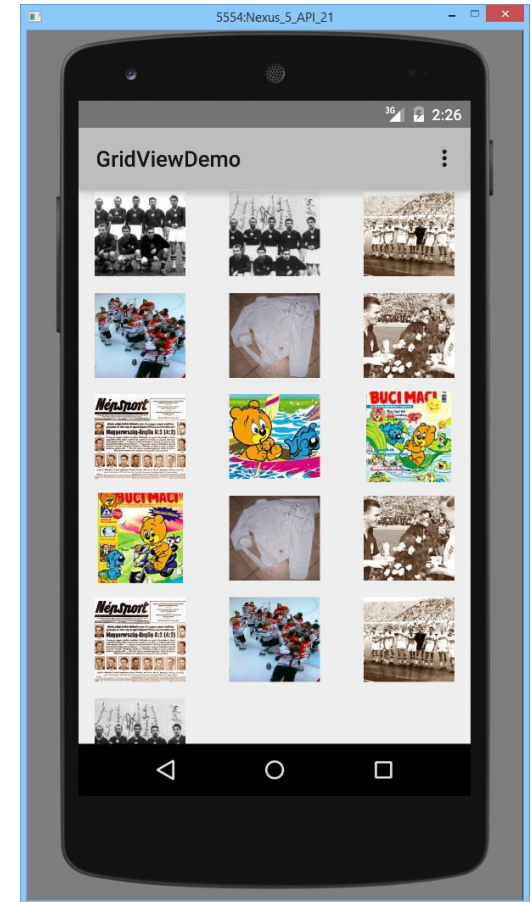
Widgets

- **ImageView**

- To display a picture
 - Set the `src`, not the background
- Important attributes
 - `scaleType`: how the image is scaled if the aspect ration of the `View` differs from the image
- `src`: the image to be displayed

GridView

- Views in a grid (matrix)
- A list adapter has to be defined
 - Adapter to get the actual View
 - For optimal usage of the resources
 - A View is instantiated if and only if it is about to be displayed or visible
 - In lists and similar Views this technique is used
- Larger content handled automatically



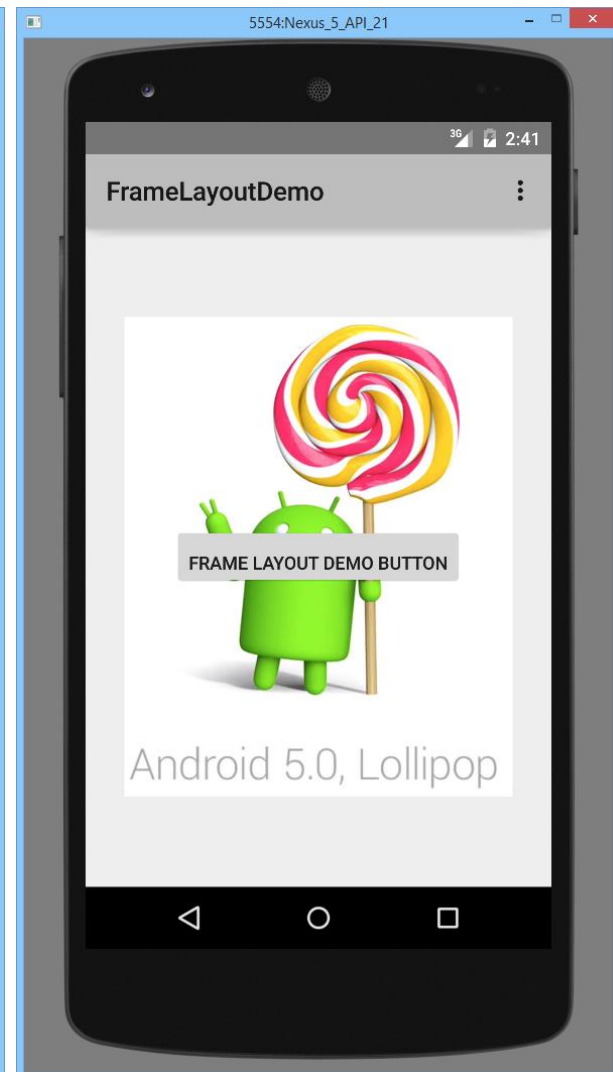
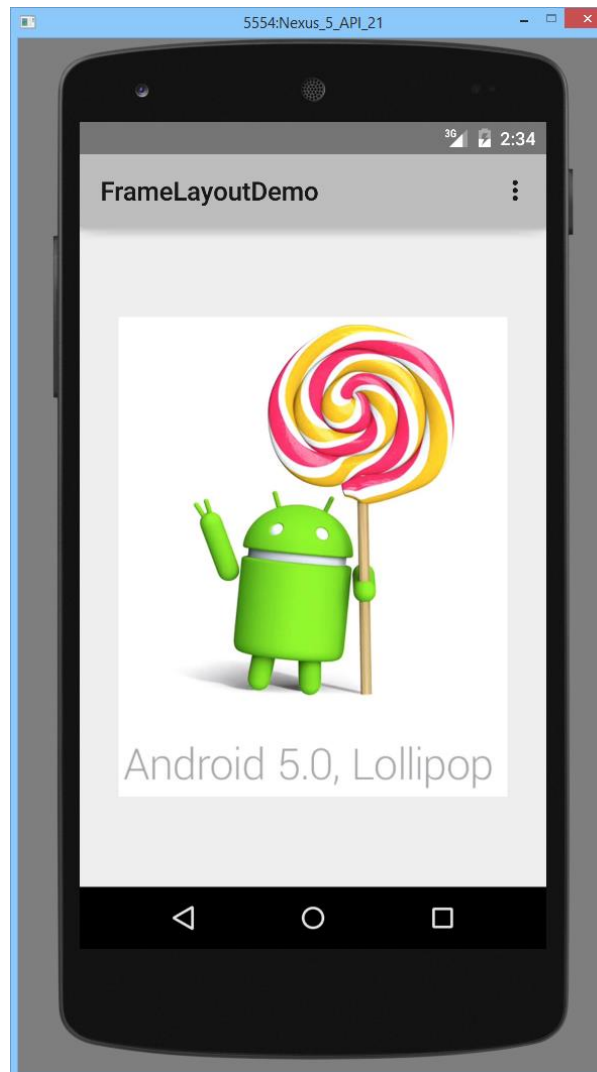
WebView

- Built-in web browser
 - To display web content

FrameLayout

- To display a single element
 - When you put more than one element the overlap each other
 - However you can specify
 - Gravity
 - Padding
 - Etc.

Example



TableLayout

- Table-like arrangement
- TableRow
 - These are the rows of the tables
 - The columns are created from the elements of the rows (horizontal layout)
- TableLayout
 - Contains TableRow elements, under each other
- TableRow
 - Contains View elements, next to each other
- The number of the columns
 - Is defined by the maximal number of Views of all TableRows
 - This maximal value is used for all rows

TableLayout

- Width of the column
 - The widest View of the column
- TableRows are always
 - `layout_width = MATCH_PARENT`
 - `layout_height = arbitrary` chosen

TableLayout

- What is the parent of the TableRow?
 - LinearLayout
- Example
 - `android:stretchColumns`: to fill the screen:
 - `android:stretchColumns="0"` , 0. column
 - `android:stretchColumns="1, 2"` , 1. and 2.
 - `android:stretchColumns="*"` , all

Defining a listener

- The most basic method is to implement the Listener by the Activity

```
public class MainActivity extends Activity implements View.OnClickListener {  
    @Override  
    public void onClick(View v) {  
        // runs on Click event  
    }  
}
```

- Thus the instances passed to the View is the instance of the current Activity
 - this

Adding a listener

- The Listener can be set in the XML
 - It is not recommended as in case the code is changed you have to modify the XML as well
 - If you forget it you will get runtime exception
- The listener also can be set in the code
 - By calling the `set<ListenerName>(listenerInstance)` function
- You should use the `Activity.findViewById` method to retrieve the instance of a view by id
 - The parameter is the id of the View which is set in the XML (Example: `R.id.myButton`)
 - This method iterates recursively the View hierarchy, and returns with the first occurrence
- It returns a View
 - Thus the type must be casted explicitly
 - `Button button = (Button) findViewById(R.id.myButton);`
 - Remember `ClassCastException` is thrown if the type mismatches
- Finally: `button.setOnClickListener(this);`

Listener – anonymous class

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    Button button1 = (Button) findViewById(R.id.button1);
    button1.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            Log.i("MainActivity", "button1 pressed");
        }
    });

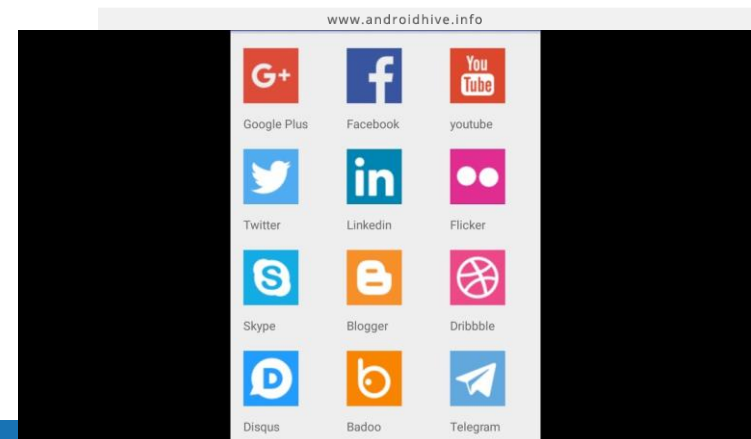
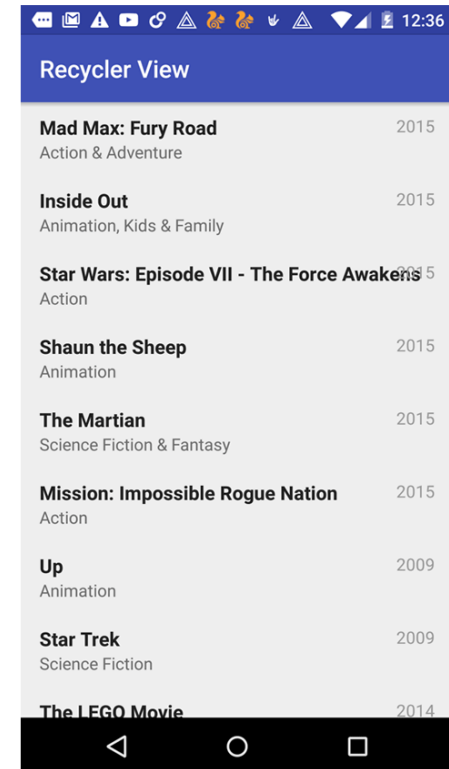
    final String string = "hello";

    Button button2 = (Button) findViewById(R.id.button2);
    button2.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            Log.i("MainActivity", string);
        }
    });
}
```

RecyclerView

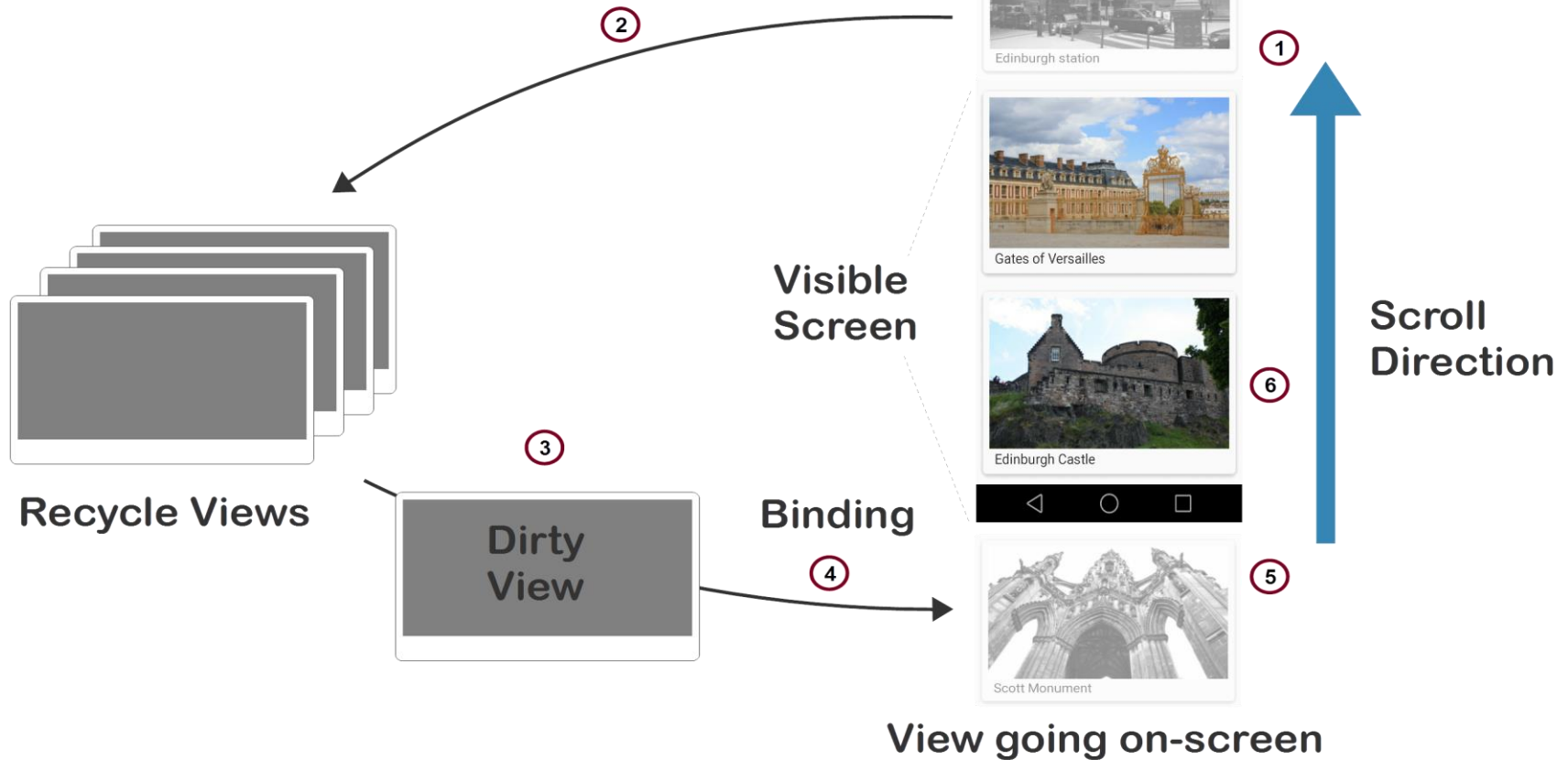
- Viewgroup
 - Renders a group of views in a similar way
 - For example lists, grids of views
- View inflating is a computationally hard task
 - RecyclerView reuses the views which are not visible
 - -> faster scrolling
 - -> Less memory usage

Android RecyclerView Example



RecyclerView - recycling

Scrap View



Components of RecyclerView

- RecyclerView class
 - This class needs to be placed on the layout and inflated with the other views if it have a LayoutManager and an Adapter defined
- LayoutManager
 - Positions the views inside a RecyclerView
 - Determines when to reuse the item views
- RecyclerView.Adapter
 - Fills the items with data
 - Needs a helper ViewHolder class
 - This stores the Views for one item
 - It inflates the proper ViewHolder
 - It binds the data to the given ViewHolder to display it
- ItemAnimator
 - It can animate the items like swipe or click animations

Usage of RecyclerView

1. Add RecyclerView support library to the gradle build file
2. Define a model class to use as the data source
3. Add a RecyclerView to your activity to display the items
4. Create a custom row layout XML file to visualize the item
5. Create a RecyclerView.Adapter and ViewHolder to render the item
6. Bind the adapter to the data source to populate the RecyclerView

More from RecyclerView

- RecyclerView.Adapter types
 - [LinearLayoutManager](#) shows items in a vertical or horizontal scrolling list
 - [GridLayoutManager](#) shows items in a grid
 - [StaggeredGridLayoutManager](#) shows items in a staggered grid
 - Custom LayoutManager can be defined if you extend [LayoutManager](#) class
- Configuration
 - Optimization (if the items won't change)
 - `recyclerView.setHasFixedSize(true);`
- Decoration
 - Add divider or other decoration for items with [ItemDecoration](#)
- Animators
 - You can add animations for add, move, delete or more complex animations with [ItemAnimators](#).

Handling touch events

- Interaction with the items (and every other view in Android) are according to the Observer pattern.
- Multiple ways. For simple click event you can use
 - Use a special ItemDecorator
 - Create an OnClickListener instance for every item
 - Make the ViewHolders implement OnClickListener (maybe the best)
- For any touch event
 - Use ItemTouchListener
- And so on...

User events, interaction

- The UI event handling is like the Observer pattern
- You have to implement an interface, which dedicated method will be called in case of the event occurs
 - In the Observer design pattern that was the notify/update function
 - In Android this function name indicates the type of the event
 - `onClick`
 - `onLongClick`
- After the class is ready, you have to instantiate
- And this instance have to be passed to the View
 - The View is the subject
 - You have to call the `set<ListenerName>` method
 - Similar to the `registerObserver`



Pázmány Péter Catholic University
Faculty of Information Technology and Bionics

Resources

res/anim - Animations

- Basic animations can be described
 - Sliding picture, animated buttons, ...

```
<set
  android:ordering=["together" | "sequentially"]>

  <objectAnimator
    android:propertyName="string"
    android:duration="int"
    android:valueFrom="float | int | color"
    android:valueTo="float | int | color"
    android:startOffset="int"
    android:repeatCount="int"
    android:repeatMode=["repeat" | "reverse"]
    android:valueType=["intType" | "floatType"]/>

  <animator
    android:duration="int"
    android:valueFrom="float | int | color"
    android:valueTo="float | int | color"
    android:startOffset="int"
    android:repeatCount="int"
    android:repeatMode=["repeat" | "reverse"]
    android:valueType=["intType" | "floatType"]/>

  <set>
  </set>
</set>
```

res/values

- Values

- To implement application with multi lingual support

- values/string.xml values-fr/string.xml

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<resources>
```

```
    <string name="string_name" >text_string</string>
```

```
</resources>
```

- The strings can be referred from

- layout.xml-s
- Activity-s

Styles

- To define custom styles
 - Rounded button
 - Custom text with coloring, format, etc. ...
- It can be applied on a single View as well as on entire Activities

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <style
    name="style_name"
    parent="@[package:]style/style_to_inherit">
    <item
      name="[package:]style_property_name"
      >style_value</item>
    </style>
  </resources>
```

- `<EditText style="@style/numbers" .../>`

res/layout

- To define GUI layouts

- ```
<?xml version="1.0" encoding="utf-8"?>
<ViewGroup xmlns:android="http://schemas.android.com/apk/res/android"
 android:id="@+[package:]id/resource_name"
 android:layout_height=["dimension" | "match_parent" | "wrap_content"]
 android:layout_width=["dimension" | "match_parent" | "wrap_content"]
 [ViewGroup-specific attributes] >
 <View
 android:id="@+[package:]id/resource_name"
 android:layout_height=["dimension" | "match_parent" | "wrap_content"]
 android:layout_width=["dimension" | "match_parent" | "wrap_content"]
 [View-specific attributes] >
 <requestFocus/>
 </View>
 <ViewGroup >
 <View />
 </ViewGroup>
 <include layout="@layout/layout_resource"/>
</ViewGroup>
```

# res/color

- Example

- To define colors for a button
- Depends on the state (normal state, pressed, released)
- ```
<?xml version="1.0" encoding="utf-8"?>  
<selector xmlns:android="http://schemas.android.com/apk/res/android" >  
    <item  
        android:color="hex_color"  
        android:state_pressed=["true" | "false"]  
        android:state_focused=["true" | "false"]  
        android:state_selected=["true" | "false"]  
        android:state_checkable=["true" | "false"]  
        android:state_checked=["true" | "false"]  
        android:state_enabled=["true" | "false"]  
        android:state_window_focused=["true" | "false"] />  
</selector>
```

res/color

- Similarly different images also can be defined for buttons
 - Depending on actual state
 - If we are not working on colors, the res/drawable should be used
- An XML have to be defined
- `android:text_color= "@color/color_name"`

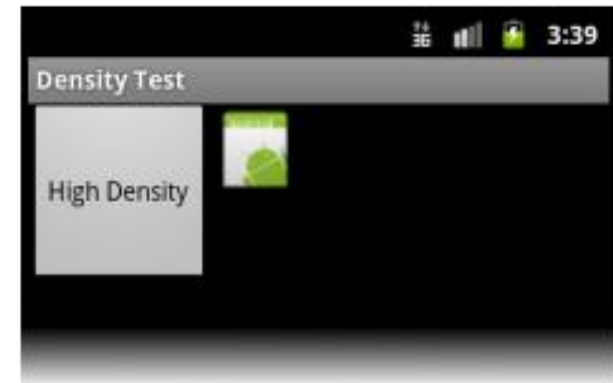
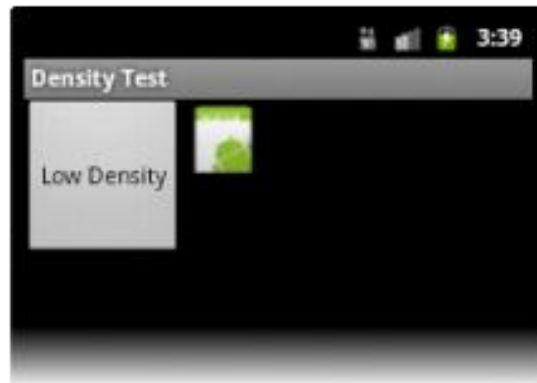
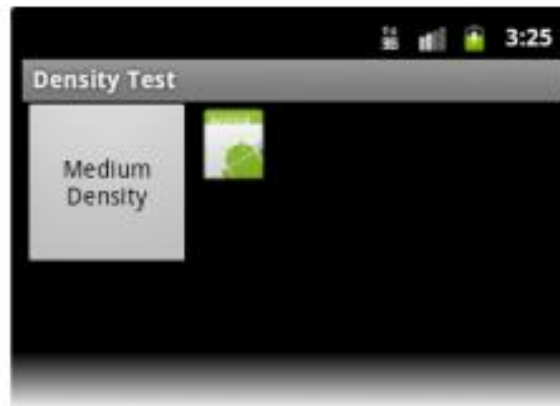
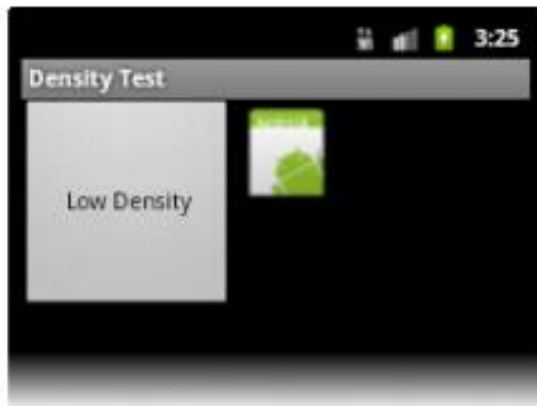


Screen sizes

Definitions

- Screen size: the physical size of the screen
- Orientation
 - Landscape or portrait
- Number of pixels
- Dpi
 - dots per inch
 - Defines the screen density as well
- Dp or Dip
 - Device independent pixel (or density independent pixel)
 - Virtual pixel
 - We can achieve that the physical size of the objects are the same on different devices
 - $1 \text{ dp} \sim 0,16\text{mm} \pm 0,02\text{mm}$ on screen
 - You have to use it in GUI XML
- Resolution
 - Number of pixels on the screen

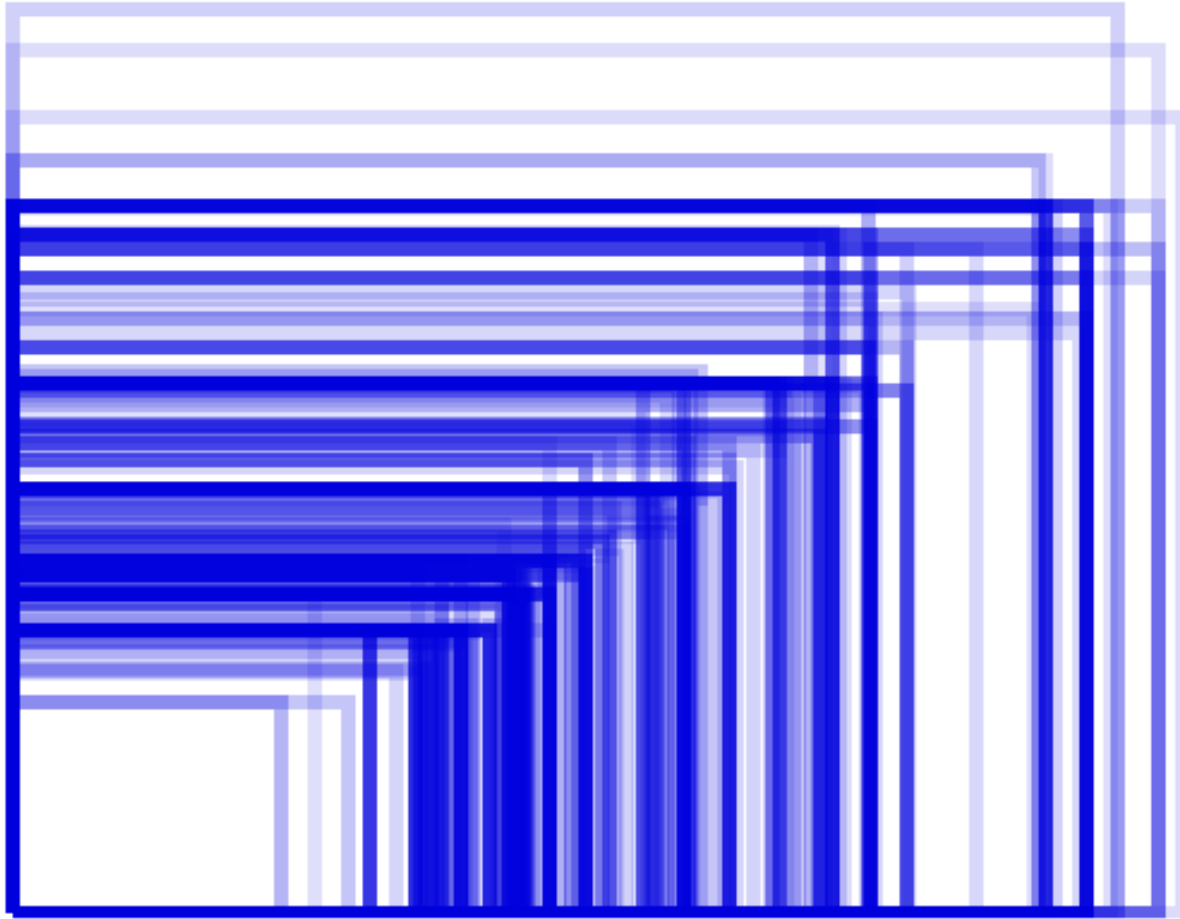
Comparison



Some example

	Low density (120)	Medium density (160)	High density (240)	Extra high density (320)
Small	240x320		480x640	
Normal	240x400 240x432	320,480	480,800 480,854 600x1024	640x960
Large	480x800 400x854	480x800 480x854 600x1024		
Extra large	1024x600	1280x800 1024x768 1280x768	1536x1152 1920x1152 1920x1200	2048x1536 2560x1536 2560x1600

Screen sizes



How to create application supporting different screens?

- Preparations: declare sizes in the AndroidManifest.xml
- Create different layouts for different screen sizes
 - res/layout-ldpi/main.xml
 - res/layout-mdpi/main.xml
- Naming convention
 - smallestWidth - sw<N>dp
 - Available screen width - w<N>dp
 - Available screen height - h<N>dp
- Example
 - Sw600dp
- How to use?
 - 320dp: typical (240x320 ldpi, 320x480 mdpi, 480x800 hdpi).
 - 600dp: 7" tablet (600x1024 mdpi).
 - 720dp: 10" tablet (720x1280 mdpi, 800x1280 mdpi).

Homework – Deadline 11/19 10.15 am

- Create a multilingual (English, Hungarian, ...) application
 - A list of several items (such as music playlist)
 - A content of the list is arbitrary, it must contain as many items as they do not fit into the screen.
 - When the button has pressed some action have to happen
 - Log, display, etc.
- The application must support
 - Landscape mode: the components are next to each other
 - Portrait mode: the components are vertically arranged



Kotlin

Next week



Pázmány Péter Catholic University
Faculty of Information Technology and Bionics

Basics of Mobile Application Development

Kotlin



Kotlin



History

- The name comes from Kotlin Island, near St. Petersburg.
- July 2011 - JetBrains unveiled Project Kotlin
 - One of the stated goals of Kotlin is to compile as quickly as Java.
- February 2012 - JetBrains open sourced the project under the Apache 2 license
- Kotlin v1.0 was released on February 15, 2016
- At Google I/O 2017, Google announced first-class support for Kotlin on Android
- Kotlin v1.2 was released on November 28, 2017
- Kotlin v1.3.5 was released on August 22, 2019

Features

- Statically typed programming language
- Runs on the Java virtual machine
 - can be compiled to JavaScript source
- Syntax is not compatible with Java
 - The JVM implementation of the Kotlin standard library interoperates with Java code
 - Relies on Java code from the existing Java Class Library
- Uses aggressive type inference to determine the types of values and expressions for which type has been left unstated.
- Kotlin code can run on JVM up to latest Java 11.
- As of Android Studio 3.0, Kotlin is fully supported by
 - The Android Kotlin compiler lets the user choose between targeting [Java 6](#), or Java 8-compatible bytecode.

Philosophy

- Kotlin is designed to be an industrial-strength object-oriented language
 - A "better language" than Java, but still be fully interoperable with Java code
- Semicolons are optional as a statement terminator
 - in most cases a newline is sufficient for the compiler to deduce that the statement has ended
- Kotlin variable declarations and parameter lists have the data type come after the variable name (and with a colon separator), similar to Pascal.
- Variables in Kotlin can be immutable, declared with the `val` keyword, or mutable, declared with the `var` keyword.
- Class members are public by default, and classes themselves are sealed by default, meaning that creating a derived class is disabled unless the base class is declared with the `open` keyword.
- In addition to the classes and methods of object-oriented programming, Kotlin also supports procedural programming with the use of functions.

Variables – immutable

- Immediate assignment
 - `val a: Int = 1`
- Int type is inferred
 - `val b = 2`
- Type required when no initializer is provided
 - `val c: Int`
- Deferred assignment
 - `c = 3`

Variables – mutable

- Int type is inferred
 - `var x = 5`
 - `x += 1`
- Value assignment

Basic types

- Numbers are similar as in Java, but not exactly the same.
 - There are no implicit widening conversions for numbers, and literals are slightly different in some cases.
- Built-in types for numbers
 - Double 64 bit
 - Float 32 bit
 - Long 64 bit
 - Int 32 bit
 - Short 16 bit
 - Byte 8 bit

Literals

- Decimals: `123`
 - Longs are tagged by a capital L: `123L`
- Hexadecimals: `0x0F`
 - Binaries: `0b00001011`
- Underscore
 - `val oneMillion = 1_000_000`
 - `val hexBytes = 0xFF_EC_DE_5E`

Basic types

- Characters are represented by the type Char
- The type Boolean represents booleans, and has two values: true and false
- Arrays are represented by the Array class
 - get and set functions
 - size property
- Strings are represented by String.
 - Immutable
 - Elements of a string are characters that can be accessed by the indexing operation: `s[i]`

String template

- Strings may contain template expressions
 - A template expression starts with a dollar sign (\$) and consists of either a simple name:
 - `val i = 10`
 - `println("i = $i")` // prints "i = 10"
 - `val s = "abc"`
 - `println("$s.length is ${s.length}")` // prints "abc.length is 3"

Range

- Defined by ..

```
val x = 10
```

```
val y = 9
```

```
if (x in 1..y+1) {  
    println("fits in range")  
}
```

- Skip numbers

- 1..10 step 2

- Reverse

- 9 downTo 0 step 3

Collections

- List

- `val fruits = listOf("banana", "avocado", "apple", "kiwifruit")`

Functions

- Declaring a function (not only as part of a class!)

```
fun sum(a: Int, b: Int): Int {  
    return a + b  
}
```

- As expression

```
fun sum(a: Int, b: Int) = a + b
```

- Method

```
fun printSum(a: Int, b: Int): Unit {  
    println("sum of $a and $b is ${a + b}")  
}
```

: Unit is optional

Nested functions

```
fun saveUser(user: User) {  
    fun validate(user: User, value: String, fieldName: String) {  
        if (value.isEmpty()) {  
            throw IllegalArgumentException("Can't save user")  
        }  
    }  
    validate(user, user.name, "Name")  
    validate(user, user.address, "Address")  
}
```

Named Arguments

- Function parameters can be named when calling functions

```
fun reformat(str: String,  
            normalizeCase: Boolean = true,  
            upperCaseFirstLetter: Boolean = true,  
            wordSeparator: Char = ' ') {  
    ...  
}  
  
reformat(str)  
reformat(str, true, true, '_')  
reformat(str,  
    normalizeCase = true,  
    upperCaseFirstLetter = true,  
    wordSeparator = '_'  
)
```

Infix notation

- Functions marked with the infix keyword can also be called using the infix notation

```
infix fun Int.shl(x: Int): Int { ... }
```

```
1 shl 2
```

```
// is the same as
```

```
1.shl(2)
```

Control statements

- Conditional statement

```
if (a > b) {  
    // do a  
} else {  
    // do b  
}
```

- You can use if as an expression

```
fun maxOf(a: Int, b: Int) = if (a > b) a else b
```

Control statements

- Conditional statement as an expression

```
fun maxOf(a: Int, b: Int) = if (a > b) a else b
```

```
val max = if (a > b) {  
    print("Choose a")  
    a  
} else {  
    print("Choose b")  
    b  
}
```

Control statements

- When (instead of switch)

```
when (x) {  
    1 -> print("x == 1")  
    2 -> print("x == 2")  
    else -> {  
        print("x is neither 1 nor 2")  
    }  
}
```

Control statements

- When, multiple cases

```
when (x) {  
    0, 1 -> print("x == 0 or x == 1")  
    else -> print("otherwise")  
}
```

- Expressions

```
when (x) {  
    parseInt(s) -> print("s encodes x")  
    else -> print("s does not encode x")  
}
```


Control statements

- Intervals

```
when (x) {  
    in 1..10 -> print("x is in the range")  
    in validNumbers -> print("x is valid")  
    !in 10..20 -> print("x is not in the range")  
    else -> print("none of the above")  
}
```

- When and functions

```
fun hasPrefix(x: Any) = when(x) {  
    is String -> x.startsWith("prefix")  
    else -> false  
}
```

Control statements

- For loop

```
for (item in collection) print(item)
```

```
for (i in 1..3) {  
    println(i)  
}
```

```
for (i in 6 downTo 0 step 2) {  
    println(i)  
}
```

Control statements

- While loop

```
while (x > 0) {  
    x--  
}
```

```
do {  
    val y = retrieveData()  
} while (y != null) // y is visible here!
```

Control statements

- Unconditional

- `return`: By default returns from the nearest enclosing function or anonymous function.
- `break`: Terminates the nearest enclosing loop.
- `continue`: Proceeds to the next step of the nearest enclosing loop.

- Labels

- Any expression in Kotlin may be marked with a label and can be used with `break` and `continue`

```
loop@ for (i in 1..100) {  
    for (j in 1..100) {  
        if (...) break@loop  
    }  
}
```

Null-safety

- Error:
 - `var a: String = "abc"`
 - `a = null`
- Allowed to be null
 - `var b: String? = "abc"`
 - `b = null`
 - `print(b)`

Null-safety

- Checking
 - `val l = if (b != null) b.length else -1`
 - `val l = b?.length ?: -1`
 - `println(b?.length)`
- Assertion
 - `val l = b!!.length`
- Safe cast
 - `val aInt: Int? = a as? Int`

Type safety and casts

```
fun getStringLength(obj: Any): Int? {  
    if (obj is String) {  
        return obj.length  
    }  
    return null  
}
```

Classes

- Class declaration
 - `class Car { ... }`
- It can be empty
 - `class Empty`
- Creating an instance
 - `val car = Car()`
 - `val customer = Customer("Joe Smith")`

Properties

- Example

```
class Address {  
    var name: String = ...  
    var street: String = ...  
    var city: String = ...  
    var state: String? = ...  
    var zip: String = ...  
}
```

Get and set

- Defaults

- explicit initializer required, default getter and setter implied
 - `var allByDefault: Int? // error`
- default getter and setter
 - `var initialized = 1`
- default getter, must be initialized in constructor
 - `val simple: Int?`
- default getter
 - `val inferredType = 1`

Get and set

- Custom get

```
val isEmpty: Boolean  
    get() = this.size == 0
```

- Alternative get

```
val isEmpty get() = this.size == 0
```

- Get and set

```
var stringRepresentation: String  
    get() = this.toString()  
    set(value) {  
        setDataFromString(value)  
    }
```

Fields

- Fields cannot be declared directly in Kotlin classes
 - When a property needs a backing field, Kotlin provides it automatically
- This backing field can be referenced
 - The `field` identifier can only be used in the accessors of the property.

```
var counter = 0
    set(value) {
        if (value >= 0) field = value
    }
```

- No backing field

```
val isEmpty: Boolean
    get() = this.size == 0
```

Constructors

- Primary constructor (optional)
 - Part of the class header
 - Cannot contain any code
 - Receives parameters that can be used
 - For property initialization
 - Or inside init blocks
 - `class Person constructor(firstName: String) { ... }`
 - If the primary constructor does not have any annotations or visibility modifiers, the constructor keyword can be omitted
 - `class Person(firstName: String) { ... }`

Initialization

- The primary constructor cannot contain any code.
- Initialization code can be placed in initializer blocks.
- During an instance initialization, the initializer blocks are executed in the same order as they appear in the class body, interleaved with the property initializers:

```
class InitOrderDemo(name: String) {  
    init {  
        println("Accessing the field $name")  
    }  
    val nameProperty: String  
    init {  
        println("Setting to a property")  
        nameProperty=name  
    }  
}
```

Primary constructor and initialization

- For declaring properties and initializing them the primary constructor can be used as follows:

```
class Person(val firstName: String, val  
lastName: String, var age: Int)  
{ ... }
```

- Much the same way as regular properties, the properties declared in the primary constructor can be mutable (var) or read-only (val).
- Also you can call functions on that parameters

```
class Customer(name: String) {  
    val customerKey = name.toUpperCase()  
}
```

Constructors – Secondary

- Secondary constructor(s)

```
class Person {  
    constructor(parent: Person) {  
        parent.children.add(this)  
    }  
}
```

- If the class has a primary constructor, each secondary constructor needs to delegate to the primary constructor

- either directly or indirectly

```
class Person(val name: String) {  
    constructor(name: String, parent: Person) : this(name) {  
        parent.children.add(this)  
    }  
}
```


Inheritance

- All classes in Kotlin have a common superclass `Any`
 - `class Example`
 - `Any` is not `java.lang.Object`;
 - Members
 - `equals()`
 - `hashCode()`
 - `toString()`
- Classes are „closed” in default
 - Cannot inherit from them
 - Thus we need to „open” to create children classes

Inheritance

- Example
 - `open class Base(p: Int)`
 - `class Derived(p: Int) : Base(p)`

Constructor

- If the derived class has a primary constructor, the base class must be initialized right there
 - using the parameters of the primary constructor
- If the class has no primary constructor
 - then each secondary constructor has to initialize the base type using the super keyword
 - or to delegate to another constructor which does that

```
class MyView : View {  
    constructor(ctx: Context) : super(ctx)  
    constructor(ctx: Context, attrs: AttributeSet) : super(ctx, attrs)  
}
```

Overriding

- Kotlin requires explicit modifiers for overridable members

```
open class Base {  
    open fun v() { ... }  
    fun nv() { ... }  
}  
class Derived() : Base() {  
    override fun v() { ... }  
}
```

Overriding

- A member marked override is itself open
 - It may be overridden in subclasses
 - If you want to prohibit re-overriding, use final
- ```
open class AnotherDerived() : Base() {
 final override fun v() { ... }
}
```

# Properties

- Overriding properties works in a similar way to overriding methods

```
open class Foo {
 open val x: Int get() { ... }
}
```

```
class Bar1 : Foo() {
 override val x: Int = ...
}
```

# Calling the superclass implementation

```
open class Foo {
 open fun f() { println("Foo.f()") }
 open val x: Int get() = 1
}

class Bar : Foo() {
 override fun f() {
 super.f()
 println("Bar.f()")
 }
 override val x: Int get() = super.x + 1
}
```

# Abstract class

- A class and some of its members may be declared abstract
- An abstract member does not have an implementation in its class

```
open class Base {
 open fun f() {}
}
```

```
abstract class Derived : Base() {
 override abstract fun f()
}
```



# Interfaces

- Interfaces are very similar to Java 8

```
interface MyInterface {
 fun bar()
 fun foo() {
 // optional body
 }
}
```

- Implementation

```
class Child : MyInterface {
 override fun bar() {
 }
}
```

# Properties

- You can declare properties in interfaces.
  - A property declared in an interface can either be abstract, or it can provide implementations for accessors.
  - Properties declared in interfaces can't have backing fields, and therefore accessors declared in interfaces can't reference them

```
interface MyInterface {
 val prop: Int // abstract
 val propertyWithImplementation: String
 get() = "foo"
 fun foo() {
 print(prop)
 }
}

class Child : MyInterface {
 override val prop: Int = 29
}
```

# Interface inheritance

- An interface can derive from other interfaces and thus both provide implementations for their members and declare new functions and properties

```
interface Named {
 val name: String
}
```

```
interface Person : Named {
 val firstName: String
 val lastName: String

 override val name: String get() = "$firstName $lastName"
}
```

# Multiple inheritance

```
interface A {
 fun foo() { print("A") }
 fun bar()
}

interface B {
 fun foo() { print("B") }
 fun bar() { print("bar") }
}

class C : A {
 override fun bar() {
 print("bar")
 }
}
```

```
class D : A, B {
 override fun foo() {
 super<A>.foo()
 super.foo()
 }
 override fun bar() {
 super.bar()
 }
}
```

# Visibility

- Classes, objects, interfaces, constructors, functions, properties and their setters can have visibility modifiers.
  - Getters always have the same visibility as the property.
- There are four visibility modifiers
  - private, protected, internal, public
- The default visibility is public.

# Visibility

- Explanation
  - If you do not specify any visibility modifier, public is used by default, which means that your declarations will be visible everywhere;
  - If you mark a declaration private, it will only be visible inside the file containing the declaration;
  - If you mark it internal, it is visible everywhere in the same module;
  - protected is not available for top-level declarations.

# Visibility

- Classes and Interfaces
  - private: visible inside this class only (including all its members);
  - protected: same as private + visible in subclasses too;
  - internal: any client inside this module who sees the declaring class sees its internal members;
  - public: any client who sees the declaring class sees its public members

# Packages – Modules

- Packages

- Functions, properties and classes, objects and interfaces can be declared directly inside a package

```
package foo
fun baz() { ... }
class Bar { ... }
```

- Modules

- A set of Kotlin files compiled together:
  - an IntelliJ IDEA module;
  - a Maven project;
  - a Gradle source set (with the exception that the test source set can access the internal declarations of main);
  - a set of files compiled with one invocation of the <kotlinc> Ant task



# Extension

- To extend a class with new functionality without having to inherit from the class
- Or use any type of design pattern such as Decorator

```
fun MutableList<Int>.swap(index1: Int, index2: Int) {
 val tmp = this[index1] // the list
 this[index1] = this[index2]
 this[index2] = tmp
}
```

```
val l = mutableListOf(1, 2, 3)
l.swap(0, 2)
```

# Extension

- Extension properties

```
val <T> List<T>.lastIndex: Int
 get() = size - 1
```

# Data classes

- To create classes whose main purpose is to hold data
  - data class `User(val name: String, val age: Int)`
- The compiler automatically derives the following members from all properties declared in the primary constructor:
  - `equals()/hashCode()` pair;
  - `toString()` of the form `"User(name=John, age=42)"`;
  - `componentN()` functions corresponding to the properties in their order of declaration;
  - `copy()` function

# Data classes

- Data classes have to fulfill the following requirements:
  - The primary constructor needs to have at least one parameter;
  - All primary constructor parameters need to be marked as `val` or `var`;
  - Data classes cannot be abstract, open, sealed or inner.

# Enum

- The most basic usage of enum classes is implementing type-safe enums:

```
enum class Direction {
 NORTH, SOUTH, WEST, EAST
}
```

```
enum class ProtocolState {
 WAITING {
 override fun signal() = TALKING
 },
 TALKING {
 override fun signal() = WAITING
 };
 abstract fun signal(): ProtocolState
}
```

# Delegation

- Supported natively:
  - A class Derived can implement an interface Base by delegating all of its public members to a specified object:

```
interface Base {
 fun print()
}
class BaseImpl(val x: Int) : Base {
 override fun print() { print(x) }
}
class Derived(b: Base) : Base by b
fun main() {
 val b = BaseImpl(10)
 Derived(b).print()
}
```

# Generics

- As in Java, classes in Kotlin may have type parameters:

```
class Box<T>(t: T) {
 var value = t
}

val box: Box<Int> = Box<Int>(1)
```

- Functions can have type parameters

```
fun <T> singletonList(item: T): List<T> {
 // ...
}

val l = singletonList<Int>(1)
```

# Java <-> Kotlin

- Existing Java code can be called from Kotlin in a natural way  
`import java.util.*`

```
fun demo(source: List<Int>) {
 val list = ArrayList<Int>()
 // 'for'-loops work for Java collections:
 for (item in source) {
 list.add(item)
 }
 // Operator conventions work as well:
 for (i in 0..source.size - 1) {
 list[i] = source[i] // get and set are called
 }
}
```



# Java <-> Kotlin

- Existing Java code can be called from Kotlin in a natural way
  - Escaping  
foo.` is` (bar)

# Java <-> Kotlin

- Kotlin is called in Java (smooth)
  - A Kotlin property is compiled to the following Java elements:
    - A getter method, with the name calculated by prepending the get prefix;
    - A setter method, with the name calculated by prepending the set prefix (only for var properties);
    - A private field, with the same name as the property name (only for properties with backing fields).
  - Package-Level Functions
    - All the functions and properties declared in a file example.kt inside a package org.foo.bar, including extension functions, are compiled into static methods of a Java class named org.foo.bar.ExampleKt

# Java <-> Kotlin

- Kotlin is called in Java (smooth)
  - The Kotlin visibilities are mapped to Java in the following way:
    - private members are compiled to private members;
    - private top-level declarations are compiled to package-local declarations;
    - protected remains protected (note that Java allows accessing protected members from other classes in the same package and Kotlin doesn't, so Java classes will have broader access to the code);
    - internal declarations become public in Java;
    - public remains public.

# Homework – Deadline 11/26 10.15 am

- You have to create a demonstration of Kotlin object oriented capabilities
- Details
  - Create a class to represent any cards
    - Content, initialization
    - Comparison
  - Create a class to represent playing cards
    - Suit, rank
    - Use inheritance
    - Use enum
  - Create two classes to store deck for cards and playing cards as well
- Test your code



# Android Kotlin

Next week



Pázmány Péter Catholic University  
Faculty of Information Technology and Bionics

# Basics of Mobile Application Development

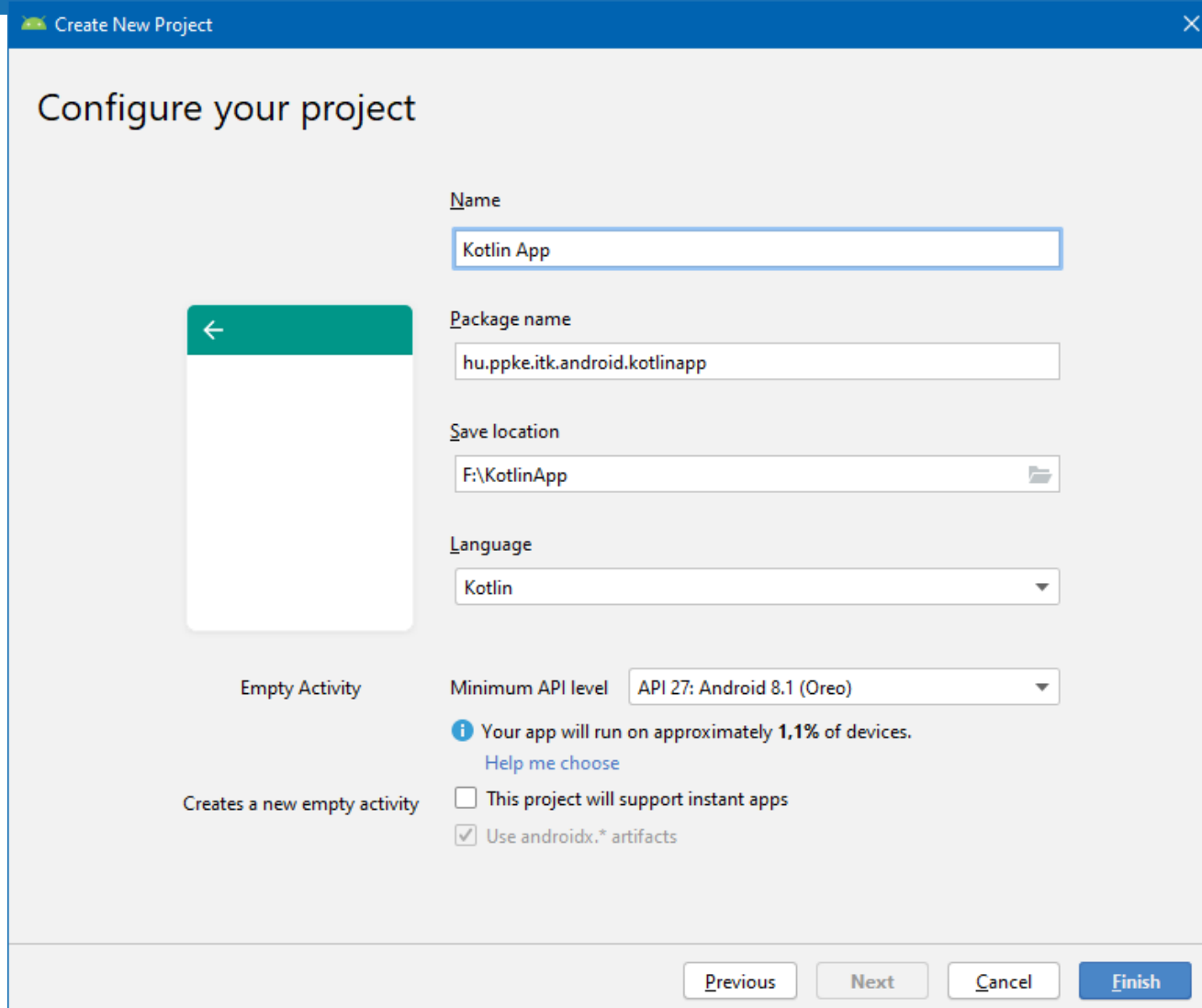
Android Basics in Kotlin



# Applications in Kotlin

# Hello World

- As previously



The screenshot shows the 'Create New Project' dialog in Android Studio. The title bar reads 'Create New Project'. The main heading is 'Configure your project'. On the left, there is a preview of an 'Empty Activity' with a green header bar containing a white back arrow. Below the preview, it says 'Empty Activity' and 'Creates a new empty activity'. On the right, there are several configuration fields: 'Name' (Kotlin App), 'Package name' (hu.ppke.itk.android.kotlinapp), 'Save location' (F:\KotlinApp), 'Language' (Kotlin), and 'Minimum API level' (API 27: Android 8.1 (Oreo)). There is also an information icon and text stating 'Your app will run on approximately 1,1% of devices.' with a link 'Help me choose'. At the bottom, there are checkboxes for 'This project will support instant apps' (unchecked) and 'Use androidx.\* artifacts' (checked). At the very bottom, there are buttons for 'Previous', 'Next', 'Cancel', and 'Finish'.

Create New Project

## Configure your project

Name  
Kotlin App

Package name  
hu.ppke.itk.android.kotlinapp

Save location  
F:\KotlinApp

Language  
Kotlin

Minimum API level  
API 27: Android 8.1 (Oreo)

*i* Your app will run on approximately 1,1% of devices.  
[Help me choose](#)

☐ This project will support instant apps

☒ Use androidx.\* artifacts

Previous Next Cancel Finish



# Default Activity

```
package hu.ppke.itk.android.kotlinapp

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle

class MainActivity : AppCompatActivity() {

 override fun onCreate(savedInstanceState: Bundle?) {
 super.onCreate(savedInstanceState)
 setContentView(R.layout.activity_main)
 }
}
```

# Add some action

- A Button and a TextView

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
 android:layout_width="match_parent"
 android:layout_height="wrap_content"
 android:orientation="vertical">

 <TextView
 android:id="@+id/textview"
 android:layout_width="match_parent"
 android:layout_height="wrap_content"
 android:text="@string/hello" />

 <Button
 android:id="@+id/button"
 android:layout_width="match_parent"
 android:layout_height="wrap_content"
 android:layout_weight="1"
 android:text="@string/button" />

</LinearLayout>
```

# Add some action

- Click action – lambda function

```
findViewById<Button>(R.id.button).setOnClickListener {
 v -> findViewById<TextView>(R.id.textview)
 .setText(getString(R.string.clicked)) }
}
```

- Click action – conventional

```
class MainActivity : Activity(), OnClickListener {

 protected fun onCreate(savedInstanceState: Bundle) {
 val button: Button = findViewById(R.id.button)
 button.setOnClickListener(this)
 }

 fun onClick(v: View) {
 findViewById<TextView>(R.id.textview).setText(getString(R.string.clicked))
 }
}
```

# Event listeners

- An event listener is an interface in the View class that contains a single callback method.
- These methods will be called by the Android framework when the View to which the listener has been registered is triggered by user interaction with the item in the UI.
- Included in the event listener interfaces are the following callback methods:
- `onClick()`
  - From `View.OnClickListener`.
  - This is called when the user either touches the item (when in touch mode), or focuses upon the item with the navigation-keys or trackball and presses the suitable "enter" key or presses down on the trackball.

# Event listeners

- `onLongClick()`
  - From `View.OnLongClickListener`.
  - This is called when the user either touches and holds the item (when in touch mode), or focuses upon the item with the navigation-keys or trackball and presses and holds the suitable "enter" key or presses and holds down on the trackball (for one second).
- `onFocusChange()`
  - From `View.OnFocusChangeListener`.
  - This is called when the user navigates onto or away from the item, using the navigation-keys or trackball.

# Event listeners

- `onKey()`
  - From `View.OnKeyListener`.
  - This is called when the user is focused on the item and presses or releases a hardware key on the device.
- `onTouch()`
  - From `View.OnTouchListener`.
  - This is called when the user performs an action qualified as a touch event, including a press, a release, or any movement gesture on the screen (within the bounds of the item).
- `onCreateContextMenu()`
  - From `View.OnCreateContextMenuListener`.
  - This is called when a Context Menu is being built (as the result of a sustained "long click").

# Event listeners

- Notice that the `onClick()` callback in the above example has no return value, but some other event listener methods must return a boolean.
- The reason depends on the event. Reasons:
  - `onLongClick()`
    - This returns a boolean to indicate whether you have consumed the event and it should not be carried further.
    - That is, return `true` to indicate that you have handled the event and it should stop here; return `false` if you have not handled it and/or the event should continue to any other on-click listeners.

# Event listeners

- Reasons:
  - `onKey()`
    - This returns a boolean to indicate whether you have consumed the event and it should not be carried further.
    - That is, return true to indicate that you have handled the event and it should stop here; return false if you have not handled it and/or the event should continue to any other on-key listeners.
  - `onTouch()`
    - This returns a boolean to indicate whether your listener consumes this event.
    - The important thing is that this event can have multiple actions that follow each other.
    - So, if you return false when the down action event is received, you indicate that you have not consumed the event and are also not interested in subsequent actions from this event.
    - Thus, you will not be called for any other actions within the event, such as a finger gesture, or the eventual up action event.





# More on Activities

# Activity state and ejection from memory

- The system kills processes when it needs to free up RAM
  - The likelihood of the system killing a given process depends on the state of the process at the time.
  - Process state, in turn, depends on the state of the activity running in the process.
  - Table shows the correlation among process state, activity state, and likelihood of the system's killing the process.

# Activity state and ejection from memory

Likelihood of being killed	Process state	Activity state
Least	Foreground (having or about to get focus)	Created Started Resumed
More	Background (lost focus)	Paused
Most	Background (not visible)	Stopped
	Empty	Destroyed

# Saving state

- The system never kills an activity directly to free up memory.
- Instead, it kills the process in which the activity runs, destroying not only the activity but everything else running in the process, as well.
- When the activity is destroyed due to system constraints, you should preserve the user's transient UI state using a combination of ViewModel, onSaveInstanceState(), and/or local storage.

# Save simple: onSaveInstanceState()

- As your activity begins to stop, the system calls the `onSaveInstanceState()` method so your activity can save state information to an instance state bundle.
- The default implementation of this method saves transient information about the state of the activity's view hierarchy
  - such as the text in an `EditText` widget or
  - the scroll position of a `ListView` widget.
- To save additional instance state information for your activity, you must override `onSaveInstanceState()`
  - and add key-value pairs to the `Bundle` object that is saved in the event.

# Example

```
override fun onSaveInstanceState(outState: Bundle?)
{
 outState?.run {
 putInt(STATE_SCORE, currentScore)
 putInt(STATE_LEVEL, currentLevel)
 }

 super.onSaveInstanceState(outState)
}

companion object {
 val STATE_SCORE = "playerScore"
 val STATE_LEVEL = "playerLevel"
}
```

# Kotlin – Companion object

- If you need a singleton you can declare the class in the usual way, but use the `object` keyword instead of `class`:

```
object CarFactory {
 val cars = mutableListOf<Car>()

 fun makeCar(horsepowers: Int): Car {
 val car = Car(horsepowers)
 cars.add(car)
 return car
 }
}
```

# Kotlin – Companion object

- If you need a function or a property to be tied to a class rather than to instances of it, you can declare it inside a companion object.
- The companion object is a singleton, and its members can be accessed directly via the name of the containing class
  - although you can also insert the name of the companion object if you want to be explicit about accessing the companion object
- A companion object is initialized when the class is loaded (typically the first time it's referenced by other code that is being executed), in a thread-safe manner.
  - You can omit the name, in which case the name defaults to Companion.
- A class can only have one companion object, and companion objects can not be nested.



# Restore activity UI state

- When your activity is recreated after it was previously destroyed, you can recover your saved instance state from the Bundle.
- Both the `onCreate()` and `onRestoreInstanceState()` callback methods receive the same Bundle.
- The `onCreate()` method is called whether the system is creating a new instance of your activity or recreating a previous one
  - You must check whether the state Bundle is null before you attempt to read it.
  - If it is null, then the system is creating a new instance of the activity, instead of restoring a previous one that was destroyed.

# Example

```
override fun onCreate(savedInstanceState: Bundle?)
{
 super.onCreate(savedInstanceState)

 if (savedInstanceState != null) {
 with(savedInstanceState) {
 currentScore = getInt(STATE_SCORE)
 currentLevel = getInt(STATE_LEVEL)
 }
 } else {
 }
}
```

# Kotlin – with

- Detour: let

- let can be used to invoke one or more functions on results of call chains.
- For example, the following code prints the results of two operations on a collection:

```
val numbers = mutableListOf("one", "two", "three", "four", "five")
val resultList = numbers.map { it.length }.filter { it > 3 }
println(resultList)
```

- Rewrite

```
numbers.map { it.length }.filter { it > 3 }.let {
 println(it)
 // and more function calls if needed
}
```

- Even better

```
numbers.map { it.length }.filter { it > 3 }.let(::println)
```

# Kotlin – with

- with

- A non-extension function: the context object is passed as an argument, but inside the lambda, it's available as a receiver (this).
  - The return value is the lambda result.
- For calling functions on the context object without providing the lambda result.

- In the code, with can be read as "with this object, do the following."

```
val numbers = mutableListOf("one", "two", "three")
with(numbers) {
 println("'with' is called with argument $this")
 println("It contains $size elements")
}
```

# Kotlin – with

- Detour: run

- The context object is available as a receiver (this).
  - The return value is the lambda result.
- run does the same as with but invokes as let - as an extension function of the context object.
- run is useful when your lambda contains both the object initialization and the computation of the return value.

```
val service = MultiportService("https://example.kotlinlang.org", 80)

val result = service.run {
 port = 8080
 query(prepareRequest() + " to port $port")
}
```

# Kotlin – with

- Detour: apply
    - The context object is available as a receiver (this).
      - The return value is the object itself.
    - Use apply for code blocks that don't return a value and mainly operate on the members of the receiver object.
      - The common case for apply is the object configuration.
      - Such calls can be read as “apply the following assignments to the object.”
- ```
val adam = Person("Adam").apply {  
    age = 32  
    city = "London"  
}
```

Kotlin – with

- Detour: also
 - The context object is available as an argument (it).
 - The return value is the object itself.
 - This is good for performing some actions that take the context object as an argument.
 - Use also for additional actions that don't alter the object, such as logging or printing debug information.
 - Usually, you can remove the calls of also from the call chain without breaking the program logic.
 - When you see also in the code, you can read it as “and also do the following”.
 - The common case for apply is the object configuration.
 - Such calls can be read as “apply the following assignments to the object.”
- ```
val numbers = mutableListOf("one", "two", "three")
numbers
 .also { println("The list elements before adding new one: $it") }
 .add("four")
```

# Configuration change occurs

- There are a number of events that can trigger a configuration change.
- Example:
  - Change between portrait and landscape orientations.
    - User has the power to rotate the device 😊
  - Change to language or input device.
    - Etc.
- When a configuration change occurs, the activity is destroyed and recreated.
  - The original activity instance will have the onPause(), onStop(), and onDestroy() callbacks triggered.
  - A new instance of the activity will be created and have the onCreate(), onStart(), and onResume() callbacks triggered.



# Fix screen orientation

- The orientation also can be fixed
  - `android:screenOrientation="portrait"`

# Bundle

- Bundles are generally used for passing data between various Android activities.
  - It depends on you what type of values you want to pass
  - Bundles can hold all types of values and pass them to the new activity.



# Fragment

# Fragments

- A Fragment represents a behavior or a portion of user interface in a `FragmentActivity`.
- You can combine multiple fragments in a single activity to build a multi-pane UI and reuse a fragment in multiple activities.
- You can think of a fragment as a modular section of an activity,
  - which has its own lifecycle,
  - receives its own input events,
  - which you can add or remove while the activity is running
    - (sort of like a "sub activity" that you can reuse in different activities).

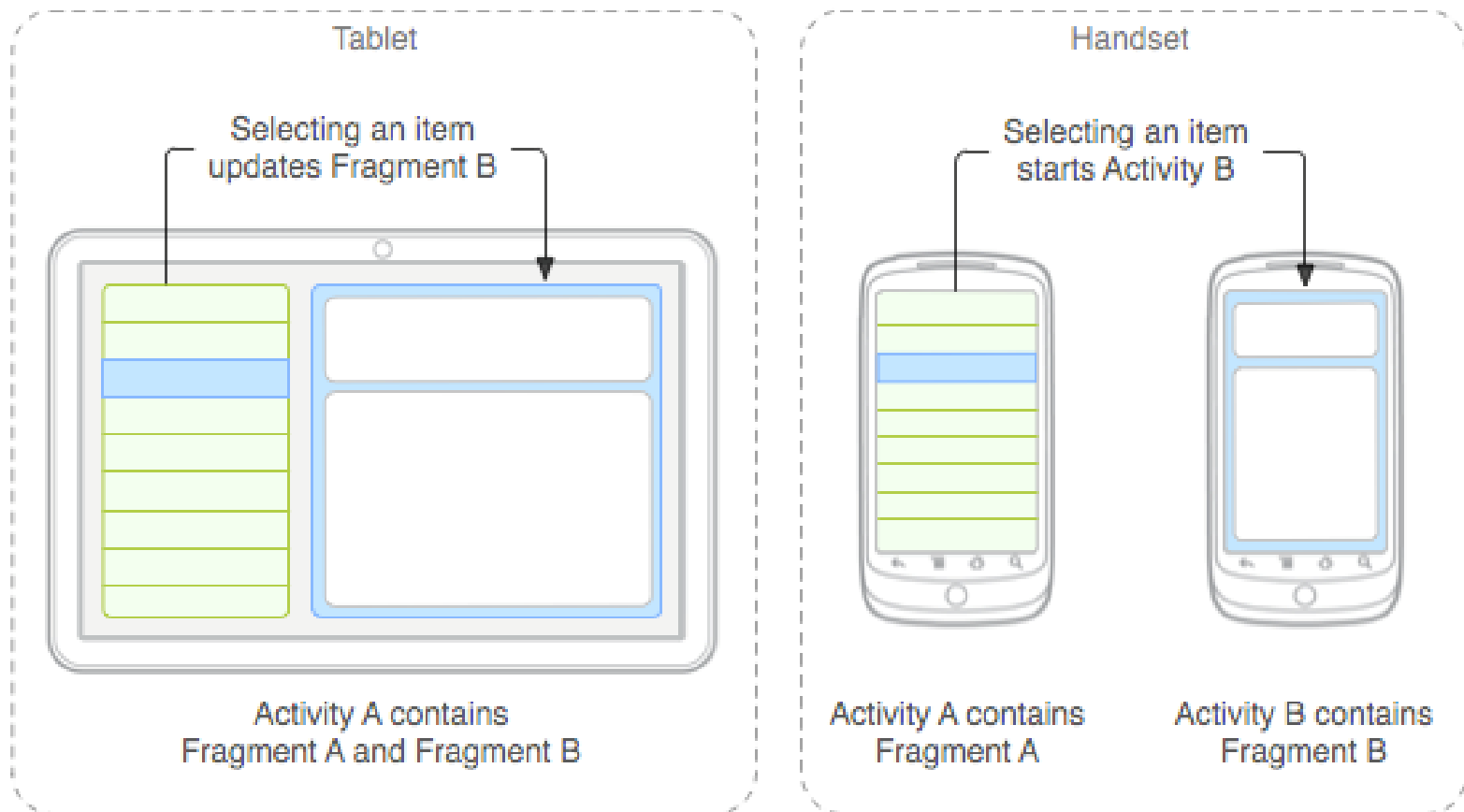
# Fragments

- A fragment must always be hosted in an activity
- The fragment's lifecycle is directly affected by the host activity's lifecycle.
  - For example, when the activity is paused, so are all fragments in it, and when the activity is destroyed, so are all fragments.
  - However, while an activity is running (it is in the resumed lifecycle state), you can manipulate each fragment independently, such as add or remove them.

# Reasons

- Android introduced fragments in Android 3.0 primarily to support more dynamic and flexible UI designs on large screens, such as tablets.
  - A tablet's screen is much larger than that of a handset, there's more room to combine and interchange UI components.
- Fragments allow such designs without the need for you to manage complex changes to the view hierarchy.
  - The activity's appearance can be modified at runtime.

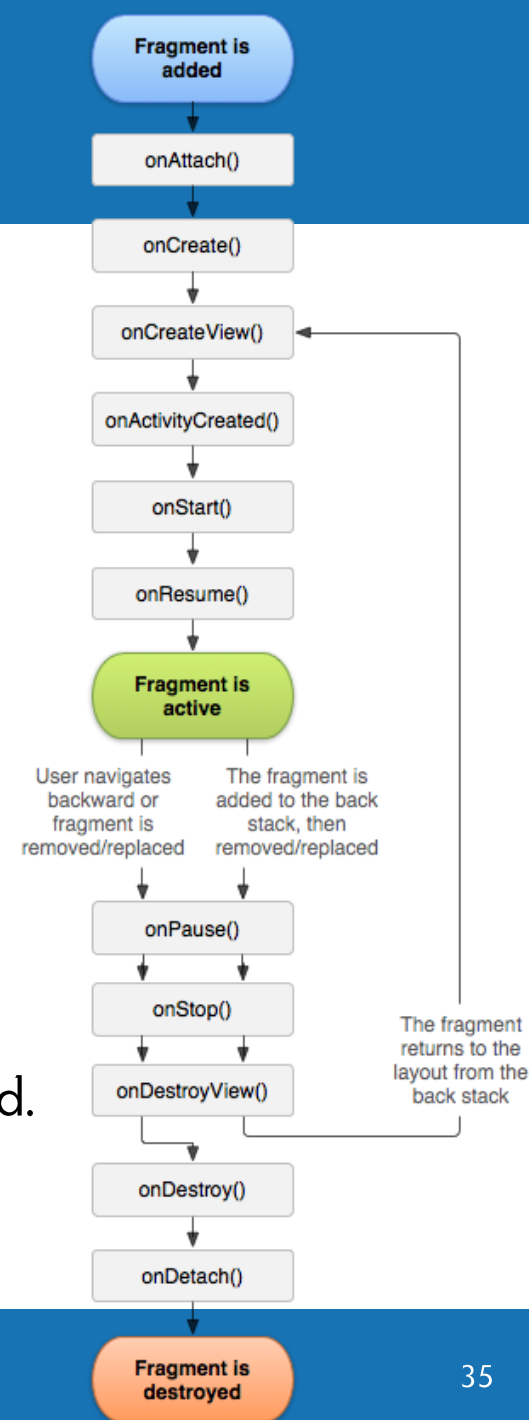
# Why?



# Fragment lifecycle

## • Functions

- **onAttach**
  - When the fragment has been associated with the activity (the Activity is passed in here).
- **onCreate**: creating an initialization
- **onCreateView**
  - To create the view hierarchy associated with the fragment.
- **onActivityCreated**:
  - when the activity's onCreate() method has returned.
- **onViewStateRestored**: state is restored

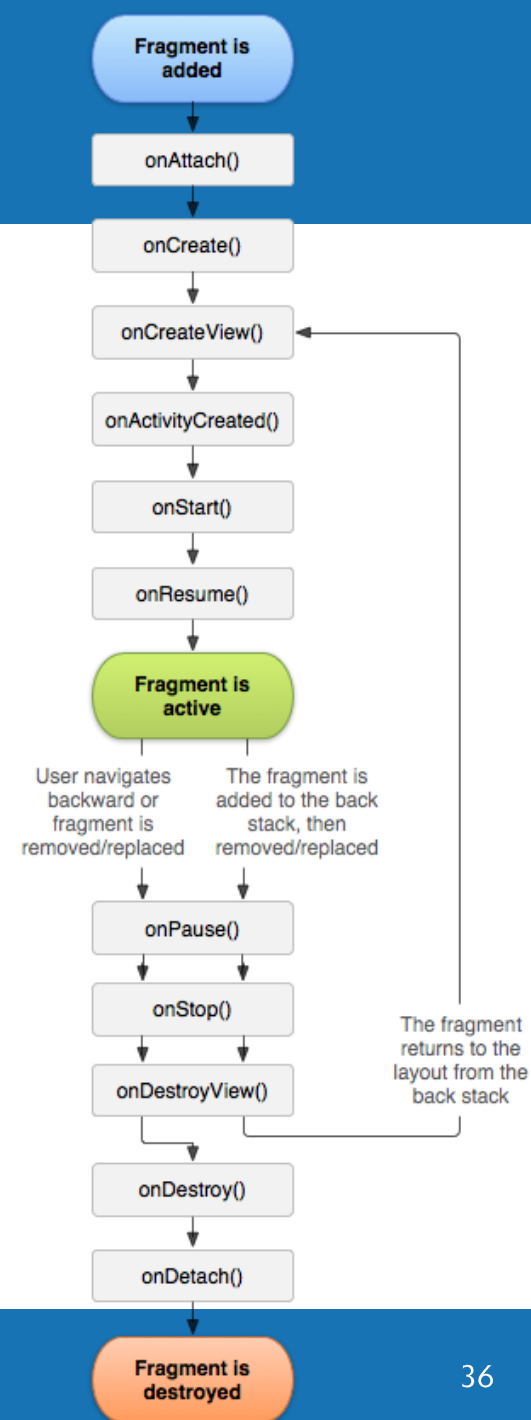




# Fragment lifecycle

## • Functions

- onStart
- onResume
- onPause
- onStop
- onDestroyView:
  - when the view hierarchy associated with the fragment is being removed.
- onDestroy
- onDetach:
  - when the fragment is being disassociated from the activity



# Fragment lifecycle

- To create a fragment, you must create a subclass of Fragment (or an existing subclass of it).
- The Fragment class has code that looks a lot like an Activity.
  - It contains callback methods similar to an activity, such as onCreate(), onStart(), onPause(), and onStop().
  - In fact, if you're converting an existing Android application to use fragments, you might simply move code from your activity's callback methods into the respective callback methods of your fragment.

# Fragment lifecycle

- `onCreate()`
  - The system calls this when creating the fragment.
  - Within your implementation, you should initialize essential components of the fragment that you want to retain when the fragment is paused or stopped, then resumed.
- `onCreateView()`
  - The system calls this when it's time for the fragment to draw its user interface for the first time.
  - To draw a UI for your fragment, you must return a View from this method that is the root of your fragment's layout.
  - You can return null if the fragment does not provide a UI.
- `onPause()`
  - The system calls this method as the first indication that the user is leaving the fragment (though it doesn't always mean the fragment is being destroyed).
  - This is usually where you should commit any changes that should be persisted beyond the current user session (because the user might not come back).

# Usage

```
class ExampleFragment : Fragment() {

 override fun onCreateView(
 inflater: LayoutInflater,
 container: ViewGroup?,
 savedInstanceState: Bundle?
): View {
 // Inflate the layout for this fragment
 return inflater.inflate(R.layout.example_fragment,
 container, false)
 }
}
```

# Usage

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
 android:orientation="horizontal"
 android:layout_width="match_parent"
 android:layout_height="match_parent">
 <fragment android:name="com.example.news.ArticleListFragment"
 android:id="@+id/list"
 android:layout_weight="1"
 android:layout_width="0dp"
 android:layout_height="match_parent" />
 <fragment android:name="com.example.news.ArticleReaderFragment"
 android:id="@+id/viewer"
 android:layout_weight="2"
 android:layout_width="0dp"
 android:layout_height="match_parent" />
</LinearLayout>
```

# From code

```
val fragmentManager = supportFragmentManager
val fragmentTransaction = fragmentManager.beginTransaction()
val fragment = ExampleFragment()
fragmentTransaction.add(R.id.fragment_container, fragment)
fragmentTransaction.commit()
```

# Transactions

- For example, here's how you can replace one fragment with another, and preserve the previous state:

```
val newFragment = ExampleFragment()
val transaction =
 supportFragmentManager.beginTransaction()
transaction.replace(R.id.fragment_container, newFragment)
transaction.addToBackStack(null)
transaction.commit()
```

# Communicating with the Activity

- A Fragment is implemented as an object that's independent from a FragmentActivity and can be used inside multiple activities
  - A given instance of a fragment is directly tied to the activity that hosts it.
  - The fragment can access the FragmentActivity instance with getActivity() and easily perform tasks such as find a view in the activity layout

```
val listView: View? = activity?.findViewById(R.id.list)
```



# Communicating with the Activity

- Likewise, your activity can call methods in the fragment by acquiring a reference to the Fragment from FragmentManager, using `findFragmentById()` or `findFragmentByTag()`.
- For example:

```
val fragment = supportFragmentManager
 .findFragmentById(R.id.example_fragment) as ExampleFragment
```



# Custom View

# Custom View

- In some cases you may want to create special Views of Widgets
- To do so you can explicitly control the drawing of a View

# onDraw and onMeasure function

- **onDraw()**
  - During drawing the GUI, when Android arrives to a View it calls the onDraw method
  - A Canvas is passed as parameter
    - You can draw on the canvas
  - It is protected
    - Thus you can override
  - What can we do?
    - Actually anything
    - There are basic drawing functions to create any shape and text, etc.
- **onMeasure()**
  - Tells the expected size of the element to the system
  - If you fail to implement this method the size of the View will be zero

# Canvas class

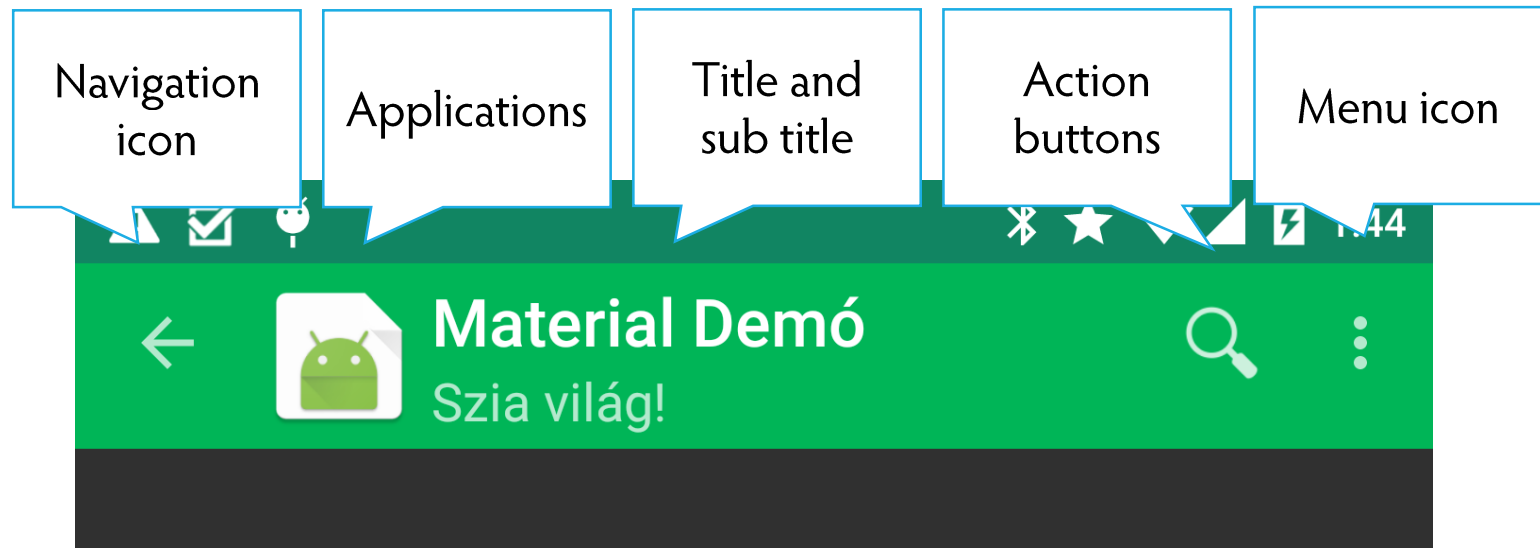
- `getWidth, getHeight()`
  - The size of the Canvas
- Drawing functions
  - `drawBitmap, drawCircle, drawColor, drawLine, drawOval, drawPoint, drawPosText, drawRGB, drawRect, drawRoundRect, drawText...`



# Toolbar

# Toolbar

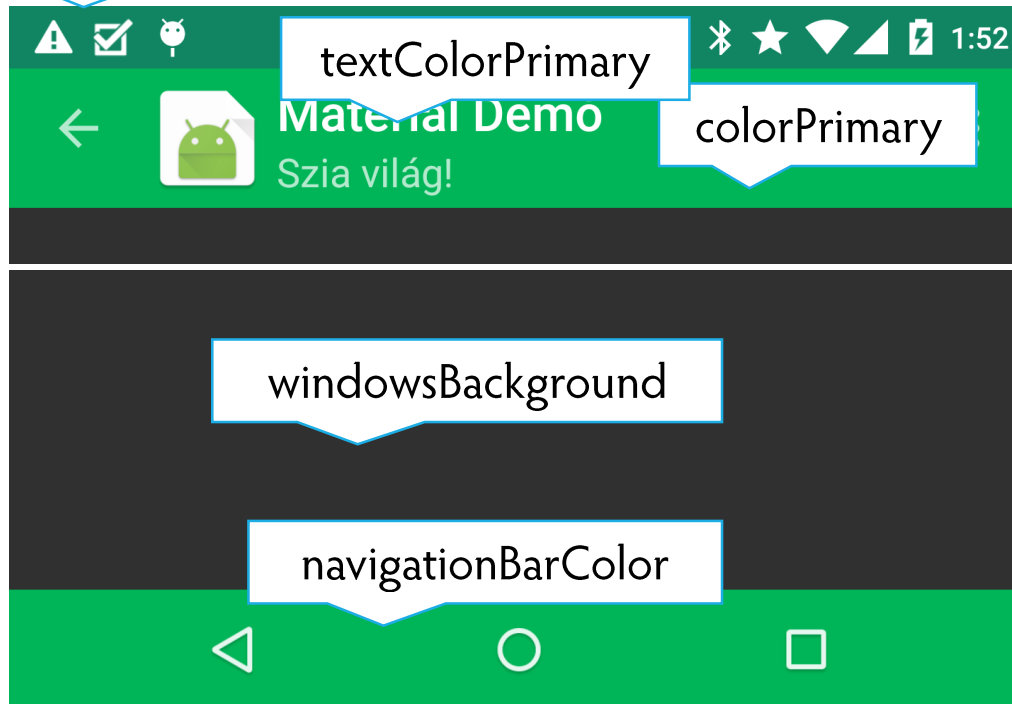
- Uses material design



# Toolbar

- Colors

colorPrimaryDark



textColorPrimary

colorPrimary

windowsBackground

navigationBarColor



# Homework – Deadline 12/03 10.15 am

- Create a basic calculator program for Android
  - In Kotlin
  - Large buttons for
    - Numbers
    - Basic operations (+ - \* /)
    - Clearing the input / screen (CE)
  - EditText
    - Indicate the results
    - Indicate the input
      - Direct input (on keyboard)
  - There is no need to implement Polish notation or such things
    - Keep simple as possible focusing on the UI and events



# Data Storage

Next week



# Basics of Mobile Application Development

Storage



# iOS method

# Core Data

- Core Data
  - A framework that you use to manage the model layer
  - Provides generalized and automated solutions to common tasks associated with object life cycle and object graph management, including persistence
- Essentially an object graph is created, which can be backed with a DBS
  - Often in SQL, but it can be in XML, or memory based DBS as well

# Core Data

- Properties
  - Maintenance of change propagation, including maintaining the consistency of relationships among objects
  - Lazy loading of objects, partially materialized futures (faulting), and copy-on-write data sharing to reduce overhead
  - Automatic validation of property values
  - Schema migration tools that simplify schema changes and allow you to perform efficient in-place schema migration
  - Grouping, filtering, and organizing data in memory and in the user interface
  - Automatic support for storing objects in external data repositories
  - Sophisticated query compilation.
    - Instead of writing SQL, you can create complex queries by associating an NSPredicate object with a fetch request

# Creating a model

- Data Model
  - The database entities and the objects are bounded together
- File | New | File
  - New file for the data model
- Section
  - Core Data
- Template
  - Data Model

# Parts of data model

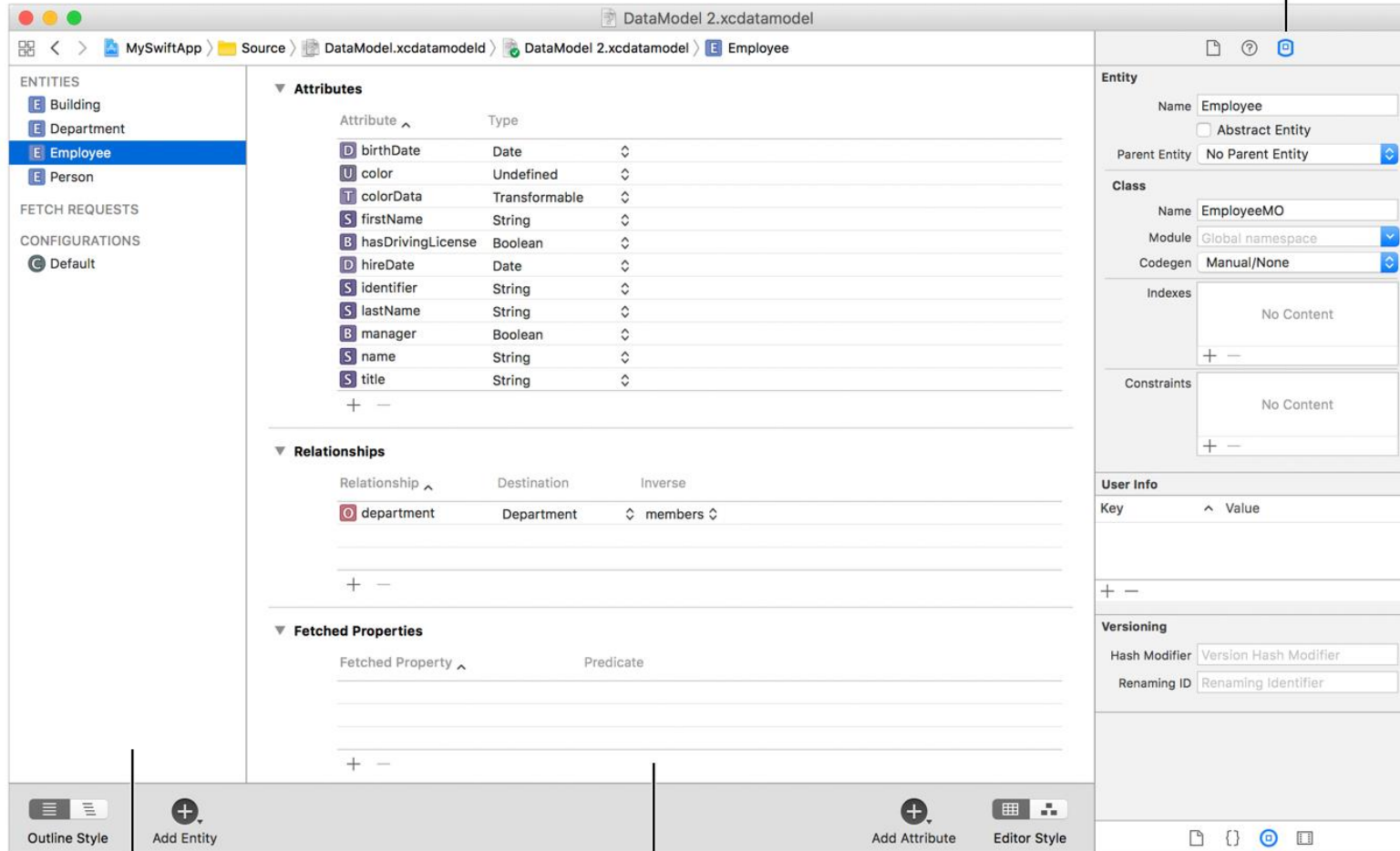
- Data model uses the following notation
  - Entity
    - As it has been used in the notation of Database Management
  - Attributes
    - Name and type
  - Relationships
    - Between entities



# Parts of data model

- For each entity a class is defined which is a subclass of `NSManagedObject`
- Attributes corresponds to the fields of classes
  - The attributes can be accessed through the methods of the `NSManagedObject`
- You can specify the data model by using a graphical editor

### Data Model inspector



The screenshot shows the Data Model inspector application interface. The main window is titled "DataModel 2.xcdatamodel" and displays the "Employee" entity. The interface is divided into three main sections: the Navigator Area on the left, the Editor Area in the center, and the Data Model inspector on the right.

**Navigator Area:** Contains a list of entities (Building, Department, Employee, Person) and configurations (Default). The "Employee" entity is selected.

**Editor Area:** Displays the attributes and relationships of the selected entity. The "Attributes" section lists attributes such as birthDate, color, colorData, firstName, hasDrivingLicense, hireDate, identifier, lastName, manager, name, and title. The "Relationships" section shows a relationship named "department" with a destination of "Department" and an inverse relationship named "members".

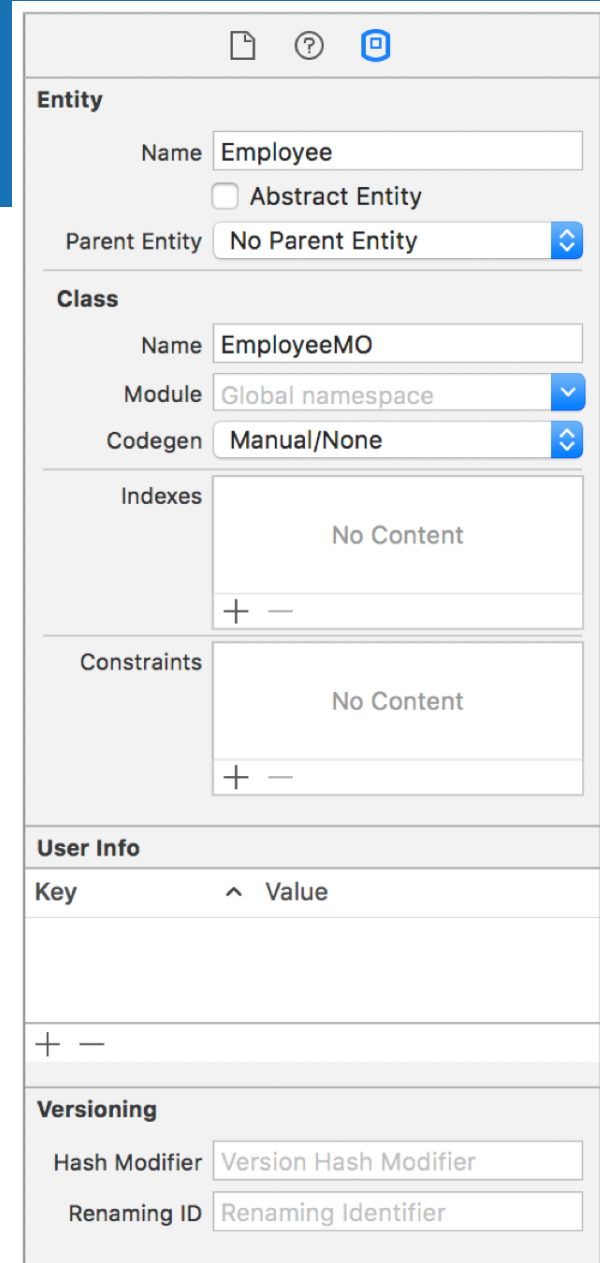
**Data Model inspector:** Provides detailed information about the selected entity. It includes fields for the entity name (Employee), parent entity (No Parent Entity), class name (EmployeeMO), module (Global namespace), and codegen (Manual/None). It also shows sections for indexes, constraints, user info, and versioning.

Navigator Area

Editor Area

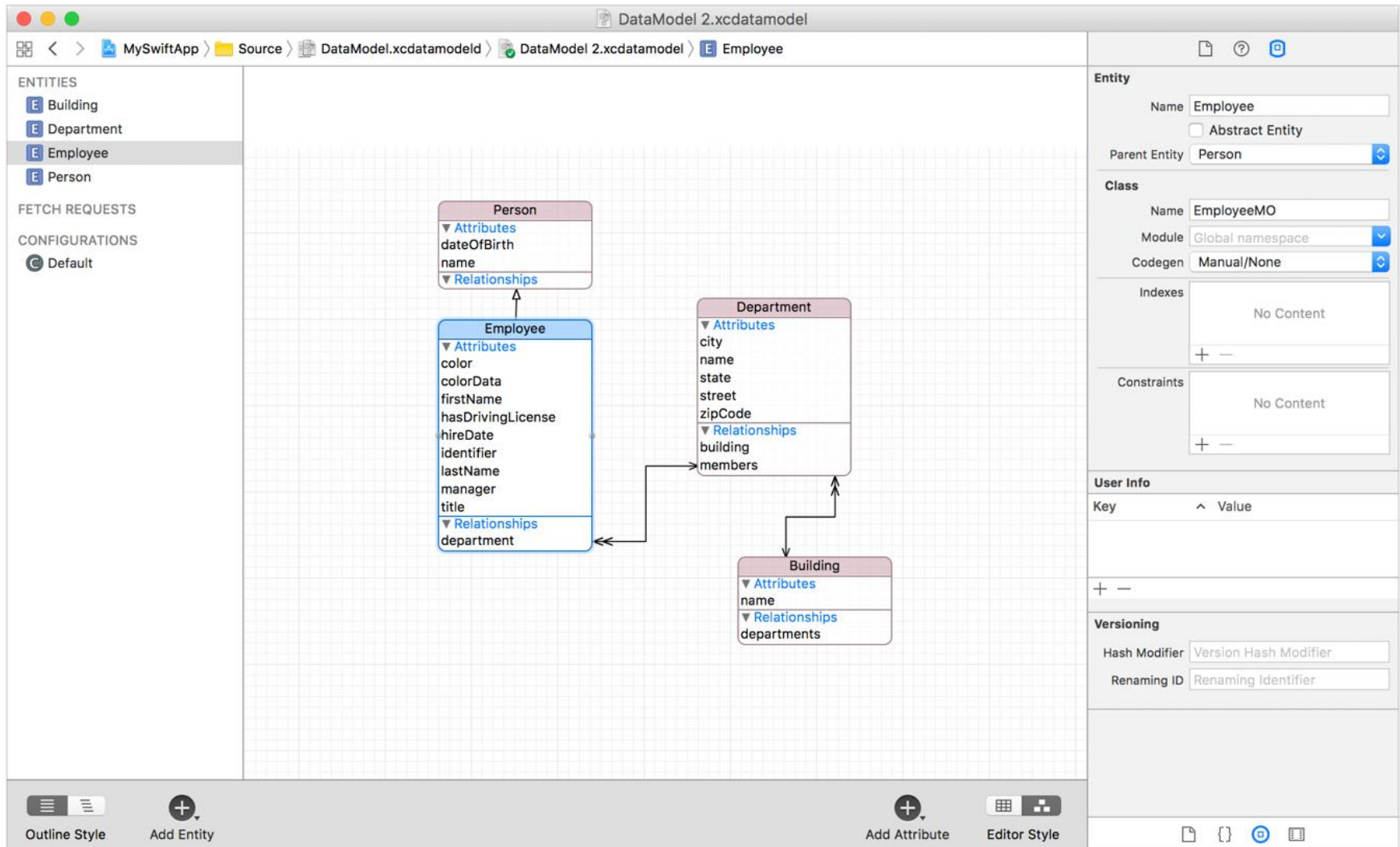
# Entity – example

- Note that the entity name and the class name are not the same
- The entity structure in the data model does not need to match the class hierarchy
- When an entity is abstract then we cannot create instances of that entity
- Entity inheritance works in a similar way to class inheritance



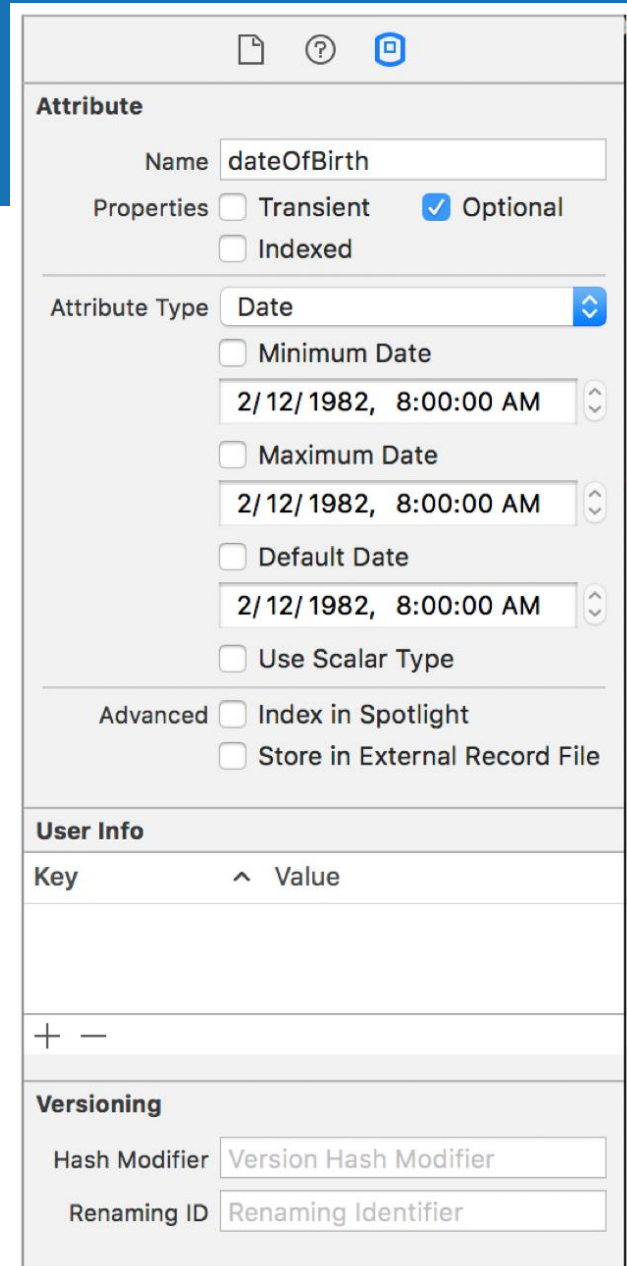
The screenshot shows a configuration window for an entity and its corresponding class. The 'Entity' section has a 'Name' field set to 'Employee', an 'Abstract Entity' checkbox that is unchecked, and a 'Parent Entity' dropdown set to 'No Parent Entity'. The 'Class' section has a 'Name' field set to 'EmployeeMO', a 'Module' dropdown set to 'Global namespace', and a 'Codegen' dropdown set to 'Manual/None'. Below these are expandable sections for 'Indexes' and 'Constraints', both currently showing 'No Content'. At the bottom, there is a 'User Info' table with columns 'Key' and 'Value', and a 'Versioning' section with 'Hash Modifier' set to 'Version Hash Modifier' and 'Renaming ID' set to 'Renaming Identifier'.

Entity	
Name	Employee
<input type="checkbox"/> Abstract Entity	
Parent Entity	No Parent Entity
Class	
Name	EmployeeMO
Module	Global namespace
Codegen	Manual/None
Indexes	No Content
Constraints	No Content
User Info	
Key	Value
+ -	
Versioning	
Hash Modifier	Version Hash Modifier
Renaming ID	Renaming Identifier



# Attributes

- Core Data supports
  - String (NSString)
  - Date (NSDate)
  - Integer (NSNumber)
  - ...
- An attribute can be optional
  - NULL in a database is not equivalent to an empty string or empty data blob



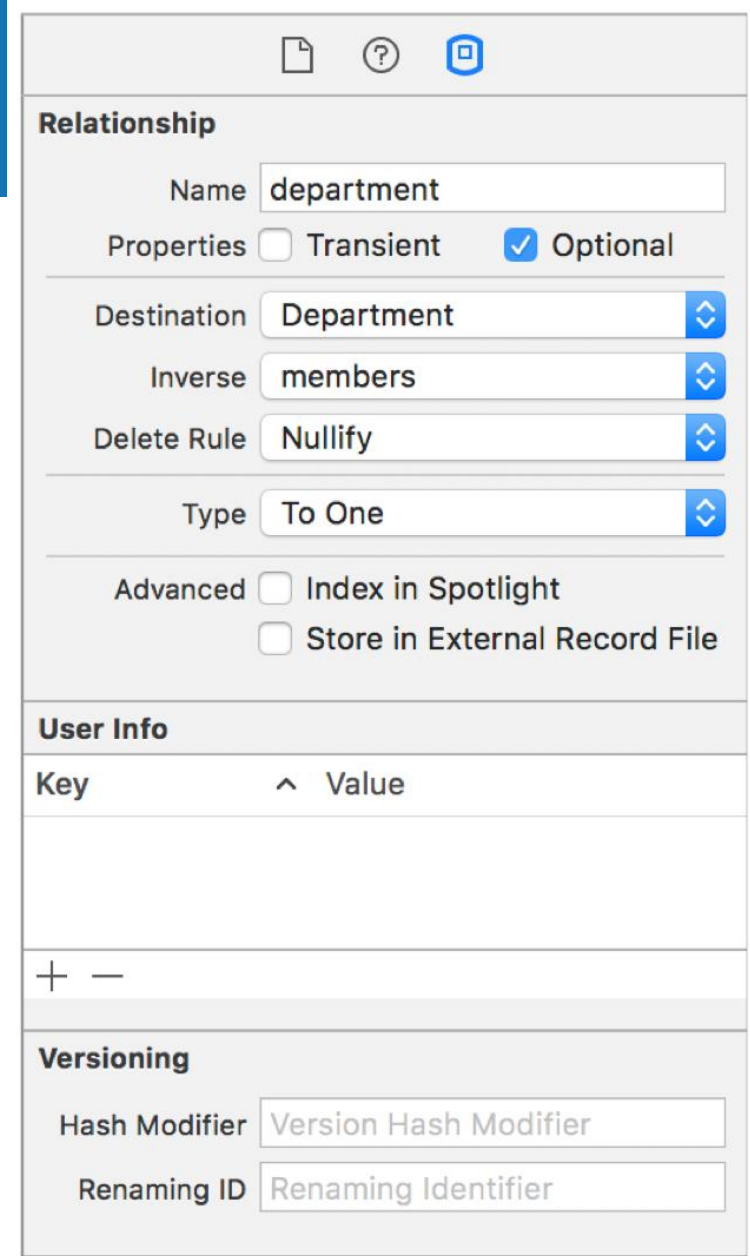
The screenshot shows the 'Attribute' configuration window in Xcode. The 'Name' field is set to 'dateOfBirth'. Under 'Properties', the 'Optional' checkbox is checked, while 'Transient' and 'Indexed' are unchecked. The 'Attribute Type' is set to 'Date'. Below this, there are three date-related fields, each with a 'Minimum Date', 'Maximum Date', and 'Default Date' checkbox, all of which are currently unchecked. The date values shown are '2/12/1982, 8:00:00 AM'. There is also a 'Use Scalar Type' checkbox, which is unchecked. In the 'Advanced' section, 'Index in Spotlight' and 'Store in External Record File' are both unchecked. Below the attribute configuration is a 'User Info' section with a table with two columns: 'Key' and 'Value'. The table is currently empty. At the bottom is a 'Versioning' section with two fields: 'Hash Modifier' (set to 'Version Hash Modifier') and 'Renaming ID' (set to 'Renaming Identifier').

# Relationship

- Relationships can be defined between entities
  - Ctrl + drag
  - Basically it is a reference to an other NSObject
  - The type of relationship can be 1-N, 1-1 as well
  - The relationship is represented in the code as references in a NSSet
- In case deletion the behavior also can be defined
  - E.g.: Nullify means that the reference is set to NULL

# Relationship

- Core Data supports to-one and to-many relationships and fetched properties
  - Fetched properties represent weak, one-way relationships
- A relationship can be
  - Many-to-one type relationship
  - Many-to-many type relationship



The screenshot shows the 'Relationship' configuration window in Xcode. It includes fields for Name, Properties (Transient and Optional), Destination, Inverse, Delete Rule, Type, and Advanced options (Index in Spotlight, Store in External Record File). Below the Relationship section is a 'User Info' table with a 'Key' column and a 'Value' column. The table is currently empty. Below the User Info section is a 'Versioning' section with fields for Hash Modifier and Renaming ID.

Key	Value
-----	-------

Versioning

Hash Modifier	Version Hash Modifier
Renaming ID	Renaming Identifier

# Example

```
import UIKit
import CoreData
class DataController: NSObject {
 var managedObjectContext: NSManagedObjectContext
 init(completionClosure: @escaping () -> ()) {
 persistentContainer = NSPersistentContainer(name: "DataModel")
 persistentContainer.loadPersistentStores() { (description, error) in
 if let error = error {
 fatalError("Failed to load Core Data stack: \(error)")
 }
 completionClosure()
 }
 }
}
```



# Example

```
let psc = NSPersistentStoreCoordinator(managedObjectModel: mom)

managedObjectContext = NSManagedObjectContext(concurrencyType:
 NSManagedObjectContextConcurrencyType.mainQueueConcurrencyType)
managedObjectContext.persistentStoreCoordinator = psc

let queue = DispatchQueue.global(qos: DispatchQoS.QoSClass.background)
queue.async {
 guard let docURL =
 FileManager.default.urls(for: .documentDirectory,
 in: .userDomainMask).last
 else {
 fatalError("Unable to resolve document directory")
 }
 let storeURL = docURL.appendingPathComponent("DataModel.sqlite")
 do {
 try psc.addPersistentStore(ofType: NSSQLiteStoreType,
 configurationName: nil, at: storeURL, options: nil)

 DispatchQueue.main.sync(execute: completionClosure)
 } catch {
 fatalError("Error migrating store: \(error)")
 }
}
```

# Example

- Creating an object

```
let employee = NSEntityDescription
 .insertNewObjectForEntityForName("Employee",
 inManagedObjectContext: managedObjectContext)
as! AAASEmployeeMO
```

# Saving

- Storing an managed object

- Manually

```
do {
 try managedObjectContext.save()
} catch {
 fatalError("Failure to save context:
\\(error)")
}
```

# Fetching objects

```
let moc = managedObjectContext
let employeesFetch = NSFetchRequest(entityName: "Employee")
do {
 let fetchedEmployees = try
 moc.executeFetchRequest(employeesFetch) as!
 [AAAEmployeeMO]
} catch {
 fatalError("Failed to fetch employees: \(error)")
}
```

# Filtering

```
let firstName = "Trevor",
fetchRequest.predicate =
 NSPredicate(format: "firstName == %@",
firstName)
```

# Properties of objects

```
class MyManagedObject: NSObject {
 @NSManaged var title: String?
 @NSManaged var date: NSDate?
}
```

# SQLite in iOS

- We can use SQLite as well
  - `sqlite3_open` – Opening, and creating the database
  - `sqlite3_prepare_v2` – Converting the SQL command from string
  - `sqlite3_step` – Executing the SQL command
  - `sqlite3_column_count` – Number of columns in the response
  - `sqlite3_column_text` – Data of the result table in text
  - `sqlite3_column_name` – Name of the attributes (column)
  - `sqlite3_changes` – Number of affected rows
  - `sqlite3_last_insert_rowids` – ID's of the inserted rows
  - `sqlite3_errmsg` – Text message about the last error
  - `sqlite3_finalize` – Finalization of the created commands
  - `sqlite3_close` – Closing the database, freeing up resources

# Example – Open connection

```
func openDatabase() -> OpaquePointer? {
 var db: OpaquePointer? = nil
 if sqlite3_open(part1DbPath, &db) == SQLITE_OK {
 print("\(part1DbPath) has been opened")
 return db
 } else {
 print("Unable to open database.")
 PlaygroundPage.current.finishExecution()
 }
}
```



# Example – Create table

```
let createTableString = ""
CREATE TABLE Contact(Id INT PRIMARY KEY NOT NULL, Name CHAR(255));
""

func createTable() {
 var createTableStatement: OpaquePointer? = nil
 if sqlite3_prepare_v2(db, createTableString, -1,
 &createTableStatement, nil) == SQLITE_OK {
 if sqlite3_step(createTableStatement) == SQLITE_DONE {
 print("Contact table created.")
 } else {
 print("Contact table could not be created.")
 }
 } else {
 print("CREATE TABLE statement could not be prepared.")
 }
 sqlite3_finalize(createTableStatement)
}
```

# Example – Insert

```
let insertStatementString = "INSERT INTO Contact (Id, Name) VALUES (?, ?);",
func insert() {
 var insertStatement: OpaquePointer? = nil
 if sqlite3_prepare_v2(db, insertStatementString, -1, &insertStatement, nil) ==
 SQLITE_OK {
 let id: Int32 = 1
 let name: NSString = "Ray"
 sqlite3_bind_int(insertStatement, 1, id)
 sqlite3_bind_text(insertStatement, 2, name.utf8String, -1, nil)
 if sqlite3_step(insertStatement) == SQLITE_DONE {
 print("Successfully inserted row.")
 } else {
 print("Could not insert row.")
 }
 } else {
 print("INSERT statement could not be prepared.")
 }
 sqlite3_finalize(insertStatement)
}
```



# Android method

# Data storage

- Shared Preferences
- Internal Storage
- External Storage
- SQLite database
- Cloud storage

# Android partitions

- /boot, /system, /recovery
  - System partitions, here is the kernel, the system image, and recovery system
- /data
  - User storage
  - Cannot be accessed directly, however the internal storage is also here
- /cache
- /misc
  - Drivers, system settings, ...
- /sdcard
- (/sd-ext, /radio, /wimax, ...)

# Shared Preferences

- Storing simple data types, key-value pairs
- boolean, float, int, long, and string
- Stored data is persistent, can be accessed after application restart
- You may use the `PreferenceActivity` for storing application preferences
- To edit an Editor has to be used

```
val sharedPref = activity?.getSharedPreferences(
 getString(R.string.preference_file_key), Context.MODE_PRIVATE)

with (sharedPref.edit()) {
 putInt(getString(R.string.saved_high_score_key), newHighScore)
 commit()
}

val defaultValue =
 resources.getInteger(R.integer.saved_high_score_default_key)
val highScore = sharedPref.getInt(getString(R.string.saved_high_score_key),
 defaultValue)
```

# Files in the APK file

- Project path
  - res/raw/directory.
- To open
  - `openRawResource()`
  - `R.raw.<filename>`
  - Returns an `InputStream`, which can be read

# Internal Storage

- Private data
  - After uninstall, the data is deleted
  - ```
String FILENAME = "hello_file";  
String string = "hello world!";  
FileOutputStream fos =  
    openFileOutput(FILENAME,  
        Context.MODE_PRIVATE);  
fos.write(string.getBytes());  
fos.close();
```


Internal Storage

- `getFilesDir()`
 - The absolute path to the storage of the application
- `getDir()`
 - Create and/or open a directory
- `deleteFile()`
- `fileList()`
 - Returns an array with the file list

External Storage

- Public storage

- Can be the SD card as well the internal memory (public storage)
- There is no security access
- Example

- `Environment.getExternalStorageDirectory();`

```
String state = Environment.getExternalStorageState();
```

```
if (Environment.MEDIA_MOUNTED.equals(state)) {
```

```
    mExternalStorageAvailable = mExternalStorageWriteable = true;
```

```
} else if (Environment.MEDIA_MOUNTED_READ_ONLY.equals(state)) {
```

```
    mExternalStorageAvailable = true;
```

```
    mExternalStorageWriteable = false;
```

```
} else {
```

```
    mExternalStorageAvailable = mExternalStorageWriteable = false;
```

```
}
```

External Storage

- Further features
 - `getExternalFilesDir()` returns the path to the directory
 - `getExternalStoragePublicDirectory()`
 - There are several pre-defined categories
 - Music, Podcasts, Ringtones, Alarms, Notifications, Pictures, Movies, Download
 - `getCacheDir()`
 - Internal storage
 - `getExternalCacheDir()`
 - External storage
 - `getExternalStorageDirectory()`
 - `/Android/data/<package_name>/cache/`

Temporary files

- A possible way, combining the well-known Java functions

```
File outputDir = context.getCacheDir();
```

```
File outputFile = File.createTempFile("prefix", "extension", outputDir);
```

Using SQL Database

- SQLite
 - Accessed by name
 - Private to the application
 - You may use the `SQLiteOpenHelper` class and override its `onCreate()` method

Using SQL Database

- Example

```
public class DictionaryOpenHelper extends SQLiteOpenHelper {  
    private static final int DATABASE_VERSION = 2;  
    private static final String DICTIONARY_TABLE_NAME =  
        "dictionary";  
    private static final String DICTIONARY_TABLE_CREATE =  
        "CREATE TABLE " + DICTIONARY_TABLE_NAME + " (" +  
            KEY_WORD + " TEXT, " +  
            KEY_DEFINITION + " TEXT);";  
  
    DictionaryOpenHelper(Context context) {  
        super(context, DATABASE_NAME, null, DATABASE_VERSION);  
    }  
  
    @Override  
    public void onCreate(SQLiteDatabase db) {  
        db.execSQL(DICTIONARY_TABLE_CREATE);  
    }  
}
```

Using SQL Database

- Once it is done, the database can be accessed by using the `getReadableDatabase()` and `getWritableDatabase()` functions
 - Both returns an `SQLiteDatabase` object
 - Then you can use the `SQLiteDatabase.query(...)` function, which returns a `Cursor`
 - For more complex query you may want to use the `SQLiteQueryBuilder` class
- It is a good practice to create a static database handling class, and then the database access can be hidden (Proxy)
 - Two ways can be used
 - `public static Cursor getAllData()`
 - Optimal as the `Cursor` requires less memory
 - `public static ArrayList<MyData> getAll data`
 - More elegant

Backup service

- You can backup the persistent data of your application to the Google Cloud services
 - After the factory reset of the phone, or on new device, or after reinstall of the application, these data can be restored
- It is important, that these data are not synchronized automatically, but the API provides functions to initiate the backup and restore
- The process is the following
 - BackupManager retrieves the data from the application and stores/uploads them
 - The BackupManager downloads the data and informs the application to start the restoration process

Backup – Purpose

- You should not use this service to synchronize between several devices
 - Although Google do not check abuse
- Application settings, user data can be stored
- The backup and the restore is executed automatically, not by request
- It is not guaranteed that it is available on all device/platform
 - You cannot rely on the backup, to provide new service
- Other application cannot access the data
- Communication is not necessarily secured

Backup in practice

1. Declare in the manifest file
2. Register the application in the backup service
3. Define a backup client, either
 - Subclassing from the BackupAgent class, or
 - Subclassing from the BackupAgentHelper class

```
<manifest ... >
    <application android:label="MyApplication"
                 android:backupAgent="MyBackupAgent">
        <activity ... >
            </activity>
        </application>
    </manifest>
```

Backup – registration

<http://code.google.com/android/backup/signup.html>

```
<application android:label="MyApplication"
            android:backupAgent="MyBackupAgent">

    ...

    <meta-data android:name="com.google.android.backup.api_key"
        android:value=" ..." />
</application>
```

BackupAgent

- The `onBackup()` and `onRestore()` function have to be overridden
 - You are responsible for the data format, version, coding, decoding
 - You are responsible for selecting the data
 - SQLite data also can be saved
- `onBackup()`
 - 3 parameter: `oldState`, `data`, `newState`

BackupHelper

- Pre-defined backup functions
 - You do not have to override the `onBackup()` and `onRestore()`
- `SharedPreferencesBackupHelper`
 - ☺
- `FileBackupHelper`

Usage

```
public class MyPrefsBackupAgent extends BackupAgentHelper {  
    // The name of the SharedPreferences file  
    static final String PREFS = "user_preferences";  
  
    // A key to uniquely identify the set of backup data  
    static final String PREFS_BACKUP_KEY = "prefs";  
  
    // Allocate a helper and add it to the backup agent  
    @Override  
    public void onCreate() {  
        SharedPreferencesBackupHelper helper = new  
            SharedPreferencesBackupHelper(this, PREFS);  
        addHelper(PREFS_BACKUP_KEY, helper);  
    }  
}
```

- `dataChanged()` call indicate the change.
 - After the signal the `onBackup()` will be called
 - There is no specific time interval or deadline



Final Exam!

Next week