Pázmány Péter Katolikus Egyetem
Információs Technológiai és Bionikai Kar

Graphical User Interface - GUI

Java PROGRAMOZÁS

6. GYAKORLAT

GUI Javában

AWT, Swing, SWT, JavaFx

GUI Javában

- A Java három beépített keretrendszere GUI-k készítéséhez:
- AWT – Abstract Window Toolkit
  - Az operációs rendszer natív grafikus elemeit használja
- Swing
  - A Java grafikai eszközeit használva megrajzolja a komponenseket
  - Az AWT-t egészíti ki, de NEM helyettesíti azt
- JavaFX
  - Szintén Javát használva hozza létre az ablakot
  - A Swing utódjának szánják
  - Az órán ezt fogjuk tárgyalni

AWT

- Gyorsabb, host oldali C kódot (peer code) hívja
- Nem 100%-ban platformfüggetlen (pontosabban nem mindenhol működik teljesen ugyanúgy)
- A host operációs rendszer kinézetével egyező megjelenést biztosít
- Korlátozottabb funkcionalitást biztosít, nehezebben portolható, viszont gyors

Swing

- Amit lehet, Java kóddal valósít meg – azaz az alapszintű rajzoló utasításokkal maga rajzolja meg az egyes elemeket
  - Emiatt lassabb, szoktak panaszkodni a „responsiveness” hiányára
- Minden környezetben pontosan ugyanúgy működik
- A host kinézetére hasonlító, de mindenhol egységes megjelenés
- Fejlettebb funkciókat biztosít, de az alapok megegyeznek az AWT-vel
- Eseménykezeléshez az AWT-biztosította lehetőségeket használja (ill. egészíti ki)

SWT – Standard Widget Toolkit

- Egy másik ismert és elterjedt toolkit: SWT
- Az IBM fejlesztette a meglévő megoldások helyett
- Nem része a Java API-nak
- A natív grafikus elemeket hívja, csak akkor rajzol Javában, ha muszáj – (félúton az AWT és a Swing között, de inkább AWT)
- Gyorsabbnak tervezték, bár nem egyértelmű az előnye
- Hasonló funkcionalitást biztosít, mint a Swing
- Az Eclipse (eredetileg IBM fejlesztés) ezt használja, a Netbeans (a Sun kezdte fejleszteni) a Swinget

JavaFX

- A 8. verziótól kezdve az Oracle JDK része
  - Az OpenJDK-hoz létezik nyílt forráskódú implementáció: OpenJFX
- Asztali és böngészős alkalmazásokhoz is
- A Swinghez hasonló, de több funkcionalitást nyújt
  - CSS támogatása
  - Egyszerűbb layout management
  - Animációk és 3D grafika támogatása
- Egységes megjelenés
- `javafx` csomagban

JavaFX User Interface

Application

- A JavaFX applikációk közös őse, ebből leszármazva írhatjuk meg a sajátunkat.
- A `start()` metódusát kell felüldefiniálni, ebben lehet létrehozni a felhasználói felületet (GUI-t).
  - További felüldefiniálható metódusok: az `init()`, mely a `start()` előtt hívódik meg, és a GUI-n kívűli inicializációért felel,
  - valamint a `stop()`, amely az alkalmazás bezárulta után hívódik meg.
- Az alkalmazás kétféleképp állhat le:
  - ha bezárják az elsődleges ablakot
  - vagy meghívják a `Platform.exit()` metódust.
  - Ezután a program futása még folytatódhat, ha még vannak futó szálak.

Stage

- Egy ablak kezelését végzi.
- Az ablak maga:
  - átméretezhető, mozgatható, ikonná tehető
  - van kerete, címe, menüsora és ikonja.
- Egy elsődleges `Stage` automatikusan létrejön induláskor, ezt a `Application.start()` paraméterként megkapja.
  - Ha ezt bezárják, alapértelmezésben az `Application` leáll.
- Az ablakot a `Scene` osztály írja le.
  - Az ablakon található elemek (`Node`-ok) hierarchikus szerkezetet (fát) alkotnak.
    - A nyomógombok, szövegek, képek, menük mind ilyen elemek.
    - Azért hierarchikus, mert egyes elemeket csoportokba szervezünk.
  - Az ablakhoz tartozó `Scene` objektumot a `setScene()` metódus állítja be.

Node

- Az összes, az ablakban megjelenő elem `Node` leszármazott. Pl.:
  - `Label`: szöveg megjelenítése
  - `TextField`, `PasswordField`: szöveg bevitelére
  - `Button`: nyomógomb
  - `RadioButton`, `CheckBox`, `ToggleButton`: választási lehetőségek kiválasztására
  - `TextArea`: többsoros szövegbevitel
  - `ScrollBar`, `ScrollPane`: görgethető felület
  - `MenuBar`, `Menu`, `MenuItem`: menük létrehozására
  - `ImageView`: képek megjelenítésére
  - És még sokan mások: https://docs.oracle.com/javase/8/javafx/user-interface-tutorial/ui_controls.htm

User Interface

- Hozzunk létre egy egyszerű ablakos alkalmazást!
- Legyen a gyökérelem egy `Group` objektum: ez egy olyan `Node`, amely másik elemeket tartalmaz (ettől lesz elágazás a hierarchiában)!
  - A `Group` gyerekeit a `getChildren()` által visszaadott lista tartalmazza, ebbe kell belerakni.
- Ne legyenek egymáson a különböző elemek!
  - `setLayoutX()`, `setLayoutY()`
  - Mi történik, ha negatív értéket adunk át?

Panelek

- Felmerülhet az igény, hogy szeretnénk a `Node`-ok egy csoportját egyben kezelni.
  - Megoldás: panelek.
  - Az előbb felhasznált `Group` is ilyen volt.
  - A panelek is `Node` leszármazottak.
- Továbbá nehézkes kézzel, egyesével beállítani az elemek helyét; és problémás, ha átméretezik az ablakot.
  - Megoldás: egyes panelek rendelkeznek szabályokkal arra nézve, hogy a gyerekeiket automatikusan el tudják helyezni, ezek a szabályok a layoutok.
  - A layoutok követik az ablak átméreteződését, és megpróbálják a gyerekeiknek a helyét, illetve méretét úgy változtatni, hogy azok elférjenek az ablakban.
  - A `Stage.sizeToScene()` hatására az ablak átméreteződik a benne található komponensek preferált méretére (`getPrefWidth()`, `getPrefHeight()`).
    - A panelek preferált mérete általában a gyerekeik preferált méretéből és a margóból számolódik.

A layoutok használatának előnyei

- A layout mindig valamilyen szabályt definiál, ami alapján meghatározza az általa kezelt elemek helyét, méretét.
- Megtehetnénk ezt kézzel is, például minden egyes elemre fixen megadva ezt az értéket, de mi van, ha:
  - a miénktől jelentősen eltérő felbontást használ a felhasználó
  - eltérő a betűtípus, betűméret
  - a programnak több nyelven is működnie kell, és ezeken bizonyos szövegek (pl. egy gomb felirata, „exit” -> „kilépés”) jelentősen eltérő hosszúságúak
  - egy újabb komponenst szeretnénk elhelyezni, de nem szeretnénk minden meglévőnek a méretét kézzel megváltoztatni.
- Az esetek jelentős részében csak egy szervező elvet szeretnénk megadni, és nem szeretnénk a fenti problémákkal egyesével foglalkozni – erre valók a layoutok.

Néhány alapvető paneltípus

- `Group`
  - nem rendezi el a gyerekeit, azok mérete a preferált méretük lesz
  - alakja felveszi a gyerekeinek a befoglaló téglalapját (nem átméretezhető)
  - transzformációk alkalmazhatók rá, ezek az összes gyerekére hatni fognak
- `FlowPane`
  - sor-, vagy oszlopfolytonosan helyezi el a gyerekeit; azok mérete a preferált méret lesz
  - közöttük vízszintesen és függőlegesen távolság adható meg
- `HBox`, `VBox`
  - vízszintesen, illetve függőlegesen helyezi el a gyerekeit
    - ebben az irányban a preferált méretükre méretezi őket
    - a `hgrow` és `vgrow` tulajdonságokkal lehet szabályozni a fennmaradó hely kitöltöttségét
    - a másik irány mentén alapértelmezetten megpróbálja a saját méretére méretezni őket
- `AnchorPane`
  - a gyerekeinek meg lehet adni oldalaktól való távolságot
  - ha két ellentétes oldaltól adunk meg távolságot, akkor át is méretezi a gyereket

Néhány alapvető paneltípus

- `GridPane`
  - egy táblázatban helyezi el a gyerekeit; egy akár több cellát is elfoglalhat
  - alapértelmezetten a tartalom határozza meg a sorok és az oszlopok méretét
    - felül lehet írni `ColumnConstraints` és `RowConstraints` objektumokkal
- `BorderPane`
  - 5 helyre tud gyereket elhelyezni: felülre, alulra, balra, jobbra, középre
  - alul és felül a gyerekek a preferált magasságukat kapják, és vízszintesen kiterjednek a panel széléig
    - a jobboldali és baloldali gyerekek a preferált szélességüket kapják, és függőlegesen elfoglalják a felső és az alsó gyerek közötti helyet
    - a középső kapja a többi helyet
- Természetesen ezeket egymásba is lehet ágyazni.
- További információ: https://docs.oracle.com/javase/8/javafx/layout-tutorial/size_align.htm

Panelek

- Módosítsuk az előző alkalmazást, hogy az elemek egy `FlowPane`-ben legyenek!
  - Állítsunk be vízszintesen és függőlegesen is 10 pixel távolságot!
- Nézzük meg a layouts csomag tartalmát!
  - Alakítsuk át a BorderPaneStage ablakot, hogy középre kerüljenek a gyerekek!
    - A „Center” feliratú gomb az ablak közepén van?
  - Alakítsuk át a GridPaneStage ablakot, hogy a gombok egyenletesen töltsék ki a teljes ablakot (mindegyik gomb legyen egyforma méretű)!

FXML és CSS

- FXML
  - Lehetőséget ad, hogy a programkódtól külön helyezzük el a GUI leírását.
    - XML nyelven
  - Általában GUI tervező programok kimenete (pl.: SceneBuilder).
  - Támogatja a lokalizációt.
  - https://docs.oracle.com/javase/8/javafx/fxml-tutorial/why_use_fxml.htm
- CSS
  - Lehetőség van az applikáció formázására CSS segítségével.
    - Ez történhet az elemekhez rendelt azonosítók
    - vagy az elemek típusa alapján
    - `scene.getStylesheets().add(”path/stylesheet.css”)`

Események

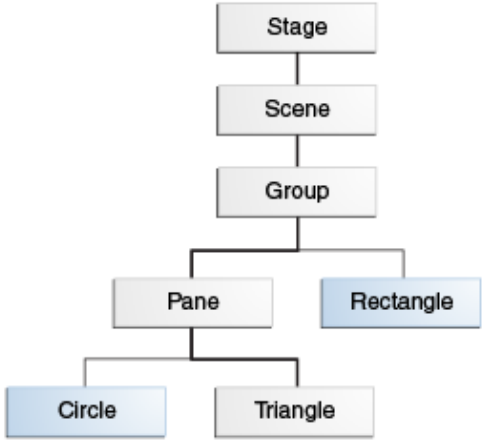
Az események kiváltása és kezelése

Események

- Eseménynek nevezünk minden felhasználói (illetve bizonyos külső) bemenetet, továbbá a programon belül kiváltott jelzést.
- Javában minden, a GUI-hoz kapcsolódó eseményt egy `EventObject` reprezentál.
- Az `EventObject` (pontosabban annak az esemény típusához illő leszármazottja) tárol minden információt az eseményről (pl. a kattintás helye, a használt gomb, stb.).
- JavaFX-ben az események ősosztálya az `Event`.
- Események például a hardver által kiváltott események:
  - egér (`MouseEvent`), billentyűzet (`KeyEvent`), érintőképernyő (`TouchEvent`).
- De eseménynek tekinthető bármi, amely a programozó szemszögéből az (szemantikus események).
  - Pl. az ablakban egy nyomógomb megnyomása (`ActionEvent`), függetlenül attól, hogy ez enterrel vagy egérrel történt.

Eseménykezelés

- Minden eseménynek van egy célja, amely egy `Node`.
  - Például, ha az ablakban (melynek gráfja jobboldalt látható) a háromszögre kattintunk, akkor az lesz a cél.
- A `Stage`-től a célig elkészül egy útvonal a hierarchiában.
  - Illetve majdnem mindig a Stage-től.
  - Ezen az úton minden objektum megkapja az eseményt.
  - Az út kétszer lesz bejárva:
    - először a `Stage`-től a célig, ekkor az esemény szűrők hajtódnak végre
    - utána a céltól vissza a `Stage`-ig, ekkor az esemény kezelők hajtódnak végre.



- Az eseményekre történő reakció az egyes objektumokra feliratkozott esemény szűrő (event filter) vagy esemény kezelő (event handler) objektumokkal történik.
  - A példához visszatérve a kattintás esemény hatására végrehajtódnak a `Stage`, a `Scene`, a `Group`, a `Pane` és a `Triangle` esemény szűrői (ebben a sorrendben),
  - majd végrehajtódnak a `Triangle`, a `Pane`, a `Group`, a `Scene` és a `Stage` esemény kezelői (ebben a sorrendben).

Eseménykezelés

- Mind az esemény szűrők, mind az esemény kezelők olyan objektumok, melyek implementálják az `EventHandler<T extends Event>` interface-t.
  - `T` az esemény típusa
  - az esemény hatására a `handle()` metódus hívódik meg, ez megkapja az eseményt paraméterként.
- Feliratkozás:
  - esemény szűrők: `scene.addEventFilter(MouseEvent.ALL, myEventHandler)`
  - esemény kezelők: `scene.addEventHandler(MouseEvent.ALL, myEventHandler)`
  - Az első paraméter az esemény fajtáját adja meg (pl. összes egéresemény), a második az azt kezelni képes szűrőt vagy kezelőt.
  - Lehetséges kényelmi metódusok használata:
    - `button.setOnAction(myActionEventHandler)`
    - Ez beállít egy esemény kezelőt a gomb megnyomására.
- Egy esemény feldolgozását a lánc közben is meg lehet szakítani:
  - `event.consume()`
  - Az adott objektumra feliratkozott összes szűrő/kezelő még megkapja, de utána a láncon már nem megy tovább.

Eseménykezelés ősosztályai

- `Event`
  - a `java.util.EventObject` leszármazottja
  - a JavaFX események közös őse
  - pl.: `ActionEvent`, `InputEvent` (`MouseEvent`, `KeyEvent`), `WindowEvent`
- `EventType<T extends Event>`
  - az esemény fajtáját (típusát) mondja meg
    - `T` az esemény típusa
  - hierarchiát alkotnak
    - közös ős: `EventType.ROOT` , ami ugyanaz, mint az `Event.ALL` (típusa `EventType<Event>`)
  - az `addEventFilter`/`addEventHandler` metódusokhoz kell
    - ha feliratkozik egy kezelő objektum egy eseményfajtára, akkor a leszármazottaira is

Eseménykezelés ősosztályai

- `EventHandler<T extends Event>`
  - `java.util.EventListener` leszármazottja
  - esemény szűrők és kezelők osztálya
  - `T` az esemény típusa
  - `handle(T)` metódusát kell felüldefiniálni, ez fog meghívódni
    - paramétere a bekövetkezett esemény

Eseménykezelés

- Adjunk interaktivitást az eddig megvalósított applikációnknak!
- Nézzük meg az `events` csomagot!
- Nézzük meg az `example` csomagot, és benne az eseménykezelőket!

Metódus referenciák

- Az előbbi kódokban néhány új módját láthattuk az eseménykezelő regisztrációjának: a metódus referenciákat.
  - Ha egy olyan lambda kifejezést írunk, amely csupán egy metódust hív, akkor ezt helyettesíthetjük metódus referenciával.
  - Szintaxis:
    - objektum::metodus : adott objektum példánymetódusa
    - osztaly::metodus : adott osztály statikus vagy példánymetódusa
    - osztaly::new : adott osztály konstruktora
  - Szemantikailag megegyezik egy lambda kifejezéssel
  - Például az alábbiak ekvivalensek:
    - ```
      node.setOnAction(event -> System.out.println(event));
      node.setOnAction(System.out::println);
      ```
    - ```
      text.setOnAction(this::rememberText);
      text.setOnAction(event -> rememberText(event));
      ```
    - ```
      setMethod(B::method);
      setMethod((b,a) -> b.method(a));
      ```
      ```

Grafika, rajzolás

Grafika

- JavaFX-ben kétféleképp lehet rajzolni:
- Egyrészt a `Canvas` segítségével:
  - rögzített mérete van
  - rá rajzolni a `canvas.getGraphicsContext2D()` által visszaadott, `GraphicsContext` típusú objektummal tudunk
  - `setStroke()`, `setFill()`: a rajzoló és a kitöltőszínt tudjuk beállítani
  - `setFont()`: a betűtípust állítja be
  - `fillRect()`, `fillRoundRect()`, `fillOval()`, `fillArc()`: kitöltött alakzatok
  - `strokeLine()`, `strokeRect()`, `strokeOval()`, `strokePolygon()`: kitöltetlen alakzatok
  - `clearRect()`: radír
  - `drawImage()`: kép rajzolása
  - Ami kilóg, azt figyelmen kívül hagyja.
- Megtekinthető példa: `example` csomag `GraphicsPane` panel.

Grafika

- Másrészt az UI kontrollok között vannak rajzolási primitívek is:
  - `Rectangle`, `Triangle`, `Arc`, `CubicCurve`
  - a `Shape` leszármazottai
  - layoutba lehet őket rendezni
  - animációt lehet rájuk definiálni
- Képek kezelése
  - ImageView: képeket megjelenítő UI kontroll
  - Image: képeket kezelő osztály
    - konstruktorában betölti a képet
    - be lehet állítani, hogy ezt háttérszálon tegye
    - BMP, GIF, JPEG és PNG formátumokat ismeri

Animáció

- Animációkat az `Animation` leszármazottai segítségével tudunk készíteni.
  - Az animáció ciklusokra van bontva, ezek hosszát és számát be lehet állítani.
- `Transition`
  - pl.: `FadeTransition`, `PathTransition`, `RotateTransition`
  - `interpolate(double)` metódusában kell implementálni az animációt
    - ennek paramétere egy 0 és 1 közötti szám, amely az animáció előrehaladtát jelzi (0 a kezdete, 1 a vége)
    - ennek alapján kell az animáció megfelelő állapotát létrehozni
  - példa az `example` csomag `AnimationPane` panelén

Animáció

- `Timeline`
  - a `Node`-ok tulajdonságait (property) lehet vele állítani
    - pl.: `translateXProperty`, `rotate`
  - `KeyFrame`-jei vannak, melyek idő-állapot párok
    - az állapot egy adott node egy tulajdonságának egy értéke
    - ezen időpontok között interpolál a `Timeline`
- `Interpolator`
  - az animáció sebességét szabályozza
  - `Interpolator.DISCRETE`: az animáció diszkrét állapotokból áll
    - `Transition` esetén az interpolate csak 0 és 1 értékkel hívódik meg
    - `Timeline` esetén a `KeyFrame`-ek ideje közben az előző állapotot tartja meg
  - `Interpolator.LINEAR`: az animáció sebessége állandó
  - `Interpolator.EASE_IN`, `EASE_OUT`, `EASE_BOTH`: az elején fokozatosan gyorsul, illetve a végén fokozatosan lassul le
- Animációk egybefűzése
  - `ParallelTransition`: párhuzamosan; `SerialTransition`: sorosan

Szálkezelés JavaFX alatt

JavaFX és a szálak

- Helyes JavaFX programok készítéséhez nagyon fontos ismerni a környezet működésének alapjait.
- Minden JavaFX applikáció eredendően többszálú:
  - az egyik szálat te hozod létre ugyanúgy, ahogy eddig (main)
  - a másik szál automatikusan létrejön a `launch()` meghívása után,
    - feladata a GUI rajzolása és az események kezelése.
    - Ennek neve JavaFX Application Thread (JAT)
    - Az `Application.start()` már ezen a szálon hívódik meg (az `init()` még nem!).
  - Ezt például debuggerben nyomon lehet követni.

A JAT feladatai

- A JAT két fő feladata: kirajzolás és eseménykezelés.
- Amikor bármilyen okból a felület egészét vagy egy részét újra kell rajzolni (akár, mert a környezet detektálta a szükségességét, akár, mert a programozó adott erre utasítást), azt a JAT szál végzi az egyes komponensek megfelelő rajzoló függvényeinek adott sorrendű lefuttatásával.
- Amikor a program bármilyen eseményt észlel (pl. kattintás valamelyik elemen, billentyű leütése egy szövegmezőben, stb.), akkor erre a JAT reagál, azaz a meghatározott eseménykezelő kódokat ez a szál futtatja.

JavaFX szálkezelés

- Bár az egész környezet eredendően többszálú, meglepő lehet, hogy (néhány kivételtől eltekintve) nem szálbiztos!
- Vagyis nincs beépített szinkronizáció, tehát a szinkronizációra a programozónak kell figyelnie.
- Ez a gyakorlatban két ökölszabály betartását jelenti minden esetben:
- 1. A JAT szálon futó kódoknak a lehető legrövidebb idő alatt kell befejeződnie.
- 2. Minden, a GUI-t érintő, annak állapotát megváltoztató kódnak a JAT szálon kell futnia.

Első szabály – rövid futásidő

- 1. A JAT szálon futó kódoknak a lehető legrövidebb idő alatt kell befejeződnie.
- Vagyis az eseménykezelő (és kirajzoló) függvényekben nem szabad hosszú futású kódot elhelyezni (pl. hosszú számítást, hálózati klienstől való nagyobb mennyiségű adat beolvasását, vagy bármit, aminek várhatóan jelentős késleltetése van).
- Ha figyelmen kívül hagyjuk ezt a szabályt, akkor a GUI “lefagy”, mivel a számításunk befejezéséig más eseményre sem tud reagálni, illetve a felületet újrarajzolni.
- Amennyiben nem tudjuk a rövid futásidőt garantálni (pl. egy gomb megnyomásának hatására komoly számítást kell végezni), akkor erre új szálat kell indítani, és abban végezni a számítást.

Második szabály - állapotváltoztatás

- 2. Minden, a GUI-t érintő, annak állapotát megváltoztató kódnak a JAT szálon kell futnia.
  - Ezt sajnos sokszor elfelejtik, pedig rendszeresen okoz például felderíthetetlen képernyőhibákat.
  - Az esetek jelentős részében a képernyő tartalmának módosítása valamilyen esemény hatására történik – például egy gomb megnyomásához tartozó eseménykezelő átállítja a szöveg színét.
  - Ilyenkor nincs probléma, mivel az eseménykezelő a JAT szálon fut, így a szöveg színének átállítása is azon történik.

Második szabály - állapotváltoztatás

- 2. Minden, a GUI-t érintő, annak állapotát megváltoztató kódnak a JAT szálon kell futnia.
  - Gondot okoz viszont, ha például az előzőek értelmében létrehozunk egy új szálat egy hosszabb számítás elvégzésére, majd ennek az eredményét szeretnénk megjeleníteni a GUI-n.
  - Az új szálunkból nem szabad ezt a változtatást közvetlenül megtenni!
    - Azaz az új szálban nem csinálhatunk közvetlenül mondjuk egy `label.setText("új szöveg");` függvényhívást.
  - Ez lényegében `IllegalStateException` kivételt eredményez.

Második szabály - állapotváltoztatás

- 2. Minden, a GUI-t érintő, annak állapotát megváltoztató kódnak a JAT szálon kell futnia.
  - Helyette létre kell hozni egy `Runnable` objektumot, aminek a `run()` metódusa elvégzi a GUI állapotmódosítását, és ezt kell az `Platform.runLater()` utasítással meghívni.
  - Ekkor a módosítást a JAT szál fogja végrehajtani, tehát a szabály teljesül.
  - Tetszőleges kódban a `Platform.isFxApplicationThread()` megadja, hogy a JAT szál futtatja-e az adott kódot.
  - Észrevehető, hogy ez már a GUI inicializálásakor is teljesül, hiszen az `Application.start()` a JAT szálon hívódik meg.
  - Javítsuk ki a `threading` csomagban található programot, hogy helyes szálkezelést alkalmazzon!

Task

- Az előzőekre létezik kényelmi osztály is, ez a `Task<V>`.
  - Implementálja a `Runnable` interface-t, úgy kell elindítani, mint a többi szálat.
  - Felül kell definiálni a `call()` metódusát, ez fog a háttérszálon futni.
    - tehát itt nem lehet GUI-t változtatni
  - Ennek lehet egy visszatérési értéke (ennek típusa a `V` generikus paraméter).
    - ezt el lehet tőle kérni kívűlről a `getValue()` metódussal
    - lehet részeredményeket, státuszinformációt kérni
  - `succeeded()` metódusa a JAT szálon hajtódik végre, ha a feladat sikerült (a `call()` nem dobott kivételt).
- Írjuk át a `threading` csomagbeli példát, hogy `Task`ot használjon!

Gyakorló feladat G07F01

- Egy grafikus alkalmazás elkészítése a feladat
- Az ablakhoz tartozzon menüsor Fájl menüvel.
  - Legyen a Fájl menünek egy Kilépés almenüje, amire az alkalmazásotok álljon le.
- Az ablakot szeparáljátok egy baloldali keskenyebb sávra illetve egy jobboldali nagyobb területre.
- A jobb területre tölts be a background.jpg háttérképet úgy, hogy az tartsa meg az arányait.
- A baloldalon legyen egy beviteli mező és egy gomb.
- Gombnyomásra a beviteli mezőn megadott alakzatok.txt fájlt nyissa meg és olvassa be belőle a következő szerkezetet:
  - newstation x=... y=... id=... category=...
  - Az x és y egy-egy koordináta az id egy azonosító szám
  - A category a következő értékeket veheti fel:
    - Circle, Square, Triangle, Pentagon, Hexagon, Cross, Star

Gyakorló feladat G07F01

- Hozz létre alakzatokat a kategóriák alapján
- Ezeket egy legördülő menüben vagy egy listában id alapján jelenítsd meg az ablak baloldali sávjában.
- Amennyiben az egyiket kiválasztjuk vagy ráklikkelünk az jelenjen meg a jobboldali területen véletlen szerűen kiszínezve.
- A feladat megoldása során törekedj rá, hogy a felhasználói felület barátságos legyen, ne fagyjon ki.
- Továbbá értelmesen legyen elrendezve úgy, hogy ha átméretezem az ablakot akkor se essen szét.
- A program futása legyen „hibatűrő”.

Gyakorló feladat G07F02

- A feladat egy primitív aknakereső készítése
- A program fő felülete egy tetszőleges méretű tábla, aminek minden eleme egy gomb
- Új játék kezdésekor a program véletlenszerűen elhelyez valahány aknát a táblán, ezt természetesen a játékos nem látja
- Egy mezőre (gombra) kattintva felfedi a mezőt, amennyiben ott nincs akna, és a gomb feliratán megjeleníti, hogy a környezetében hány akna van.
  - Ha a mező akna volt, természetesen vége a játéknak
- A mezőn jobb gombbal való kattintás hatására megjelöli a mezőt, mint aknát
  - Ezt a feliraton egy x jelezze
- A játék akkor ér véget, ha minden nem-akna mezőt felfedtünk

Gyakorló feladat G07F02

- A programnak legyen menüje, lehessen új játékot kezdeni
- Írd ki, hogy a játék kezdete óta mennyi idő telt el (perc:másodperc) – ezt természetesen folyamatosan frissítsd is
  - Figyelj a megfelelő szálkezelésre! A GUI ne fagyjon le és a szinkronizációra is ügyelj
- Ha lehet, a játék kezdetén kérdezd meg a felhasználótól, hogy mekkora táblán és milyen nehézségen szeretne játszani, de ez nem alapkövetelmény
  - A nehézséget az aknák sűrűségeként értelmezzük
- Az elkészítés során a szokásos GUI elemeket (pl. Button) és megfelelő Pane-eket használd

Gyakorló feladat G07F03

- A feladat egy Puzzle alkalmazás elkészítése.
- A képernyőn legyen menüsáv, egy Fájl menüponttal, melyhez egy Kilépés almenü tartozik. Erre klikkelve álljon le az alkalmazás.
- Legyen egy beviteli mező és egy gomb, a gomb megnyomására a beviteli mezőn megadott elérésű képfájlt töltse be az alkalmazás.
- A beolvasott képet fel kell darabolni tetszőleges számú darabra.
  - Segítség: képdaraboláshoz használható az `ImageView.setViewPort()` metódus.
- Ezután tetszőleges megjelenítéssel, nem kell bonyolulttá tenni, legyenek megjelenítve a darabolt képek.
- Készíts egy, a darabokkal azonos méretű táblát; ide fognak kerülni az általunk kiválasztott képdarabok.
- A megjelenített, összekevert képdarabokra kattintva legyen lehetőség azt a táblára áthelyezni. Így lehetőségünk van az képet újra „összerakni”.

Gyakorló feladat G07F03

- A feladat megoldása során törekedj rá, hogy a felhasználói felület barátságos legyen, ne fagyjon ki.
- Továbbá értelmesen legyen elrendezve úgy, hogy ha átméretezem az ablakot akkor se essen szét.
- A program futása legyen „hibatűrő”.

Gyakorló feladat G07F04

- A feladat egy játék megvalósítása, amelyben a játékosok a reakcióidejüket mérik össze.
- Minden játékos kap egy betűt véletlenszerűen
  - Az ablakban legyen egy start gomb, amivel egy kör indul
  - Továbbá legyen egy téglalap, amely feketéről kékre vált a kör indulása után véletlen idő múlva
  - Ezután kell a játékosoknak a leghamarabb megnyomni a kapott betűt
  - Miután megnyomták és amíg a kör be nem fejeződött, a start gomb ne legyen megnyomható
- Oldalt legyen egy lista, amely a legutolsó kör eredményét jelzi
  - A játékosok neve és ideje, utóbbi szerint rendezve
  - Akik a téglalap átváltása előtt reagáltak, azok a végén legyenek, egyébként a reakcióidő szerinti értelemszerű sorrendben
- Egy gombbal lehessen új játékost hozzáadni
  - Ekkor egy új dialóg ablakban lehessen megadni a játékos nevét (ez ablak nyilván ezután záródjon be)

Gyakorló feladat G07F04

- A feladat megoldása során törekedj rá, hogy a felhasználói felület barátságos legyen, ne fagyjon ki.
- Továbbá értelmesen legyen elrendezve úgy, hogy ha átméretezem az ablakot akkor se essen szét.
- A program futása legyen „hibatűrő”.