1. Feladat – OOP (50%)

Legyen egy osztályunk, Value és ennek egy példánya, mely egy double értéket tárol. Ezt az objektumot fogja több másik objektum "figyelni", és ha megváltozik, akkor frissíteni fogják a saját állapotukat. Legyen többféle megfigyelő objektum:

- RepeatPrinter: a megfigyelt értéket egy sorban kiírja ötször
- Multiplier: a megfigyelt értéket megszorozza valamilyen számmal (konstruktorban átadásra kerül), majd kiírja azt
- Checker: a megfigyelt értéket összehasonlítja valamilyen számmal (konstruktorban átadásra kerül), majd kiírja, hogy egyezik-e vagy sem
- PreviousAdder: a megfigyelt értéket hozzáadja a megfigyelt érték előző értékéhez (amit ő maga tárol), majd kiírja azt. Ha nem volt előző érték, akkor nullát ad hozzá.

Minden megfigyelő objektum megvalósítja az Observer interfészt, mely interfésznek egy "update" függvénye van, melyet a Value osztály példánya hív meg ha megváltozik az általa tárolt érték - átadva az új értéket. A megfigyelő objektumok az "update" implementációjában végezzék el a fentebb leírt műveleteket.

A Value osztály tegye lehetővé, hogy Observer interfészt implementáló objektumok "regisztrálhassanak" a frissítésekre. A regisztrált objektumokat tároljuk egy listában (pl. ArrayList), és ha a tárolt érték megváltozik, akkor minden tárolt objektumon hívjuk meg az update függvényt, átadva az új értéket.

Pontozás: 20 pont - helyes osztályok, interfészek implementációja; 20 pont - megfigyelő objektumok regisztrálása, változás esetén értesítése; 10 pont - tesztelő kód elkészítése, különböző számú és típusú megfigyelőkkel

2. Feladat - Reptér (50%)

A feladat egy reptéri beszállítás szimulációja onnantól, hogy az utasok kiérnek a reptérre, amíg feljutnak a repülőgépre. A reptérre az utasok, véletlenszerű időpontban, egymás után érkeznek 0,5 és 1,5 másodperc közötti intervallumban. A gépre összesen 230 utas fér fel.

Az utasok közül átlagosan minden harmadik ad fel csomagot, amit két helyen tehetnek meg. A csomagfeladás egy embernek 0,2 másodpercig tart, amennyiben mindkét feladó hely foglalt az utasok 0,1 másodpercenként próbálkoznak. Aki már feladta a csomagot (vagy eleve nem adott fel), az mehet a biztonsági ellenőrzésre, ami öt helyen történik párhuzamosan és 0,5 másodpercig tart, ha mind az öt hely foglalt, akkor az utasok 0,1 másodpercenként újra próbálkoznak. Itt 0,1 valószínűséggel meg kell ismételni az ellenőrzést valamilyen hiba miatt (pl. nem tette ki a vizespalackot a táskából, …).

A biztonsági ellenőrzés után az utasok közül néhányan még vásárolgatnak, mások pedig egyből a beszállításhoz mennek, így 1-4 másodperc időt vesz igénybe, míg az utasok a beszállítást végző személyzethez érnek. Az utasok közül minden negyedik vett elsőbbségi beszállítást, ők elsőbbséget élveznek, míg a többiek nem. Ha van valaki az elsőbbségi sorban, akkor mindenképpen ő következik a beszállításnál (tehát ha várakozik elsőbbségi utas, akkor minden nem elsőbbségi utas udvariasan vár, hogy ráeszméljen, hogy ő következik), ami 0,3 másodpercig tart, itt szintén 0,1 másodpercenként próbálnak sorra kerülni az utasok.

A program logolja a történéseket a konzolra (használhatod a rendszeridőt, mint időbélyeg), és a szimuláció végén írja ki, hogy összesen mennyi idő alatt szálltak be az utasok.

Log példa:

10:30:11, A(z) 3. személy feladta a csomagját a(z) 1. számú pultnál.

10:30:16, A(z) 45. személynek meg kell ismételni a biztonsági ellenőrzést a(z) 4. szalagnál.

10:30:18, A(z) 2. személy beszállt a ~~(z) 2. buszba.~~ gépbe.

A megoldás során ne használj busy waitinget se a gyakorlatokon tanul wait-notify-t, használd helyette a Java platform/library-k lehetőségeit, amiket gyakorlatokon néztünk (szálak 1)! Figyelj arra is, hogy a programod szálbiztos legyen!