Liftszimulátor

Tutkovics András

SLQ311

Osztályok:

Call:

Az elemi szintű lifthívások osztálya. Tárolva van benne, hogy honnan hova utazik az illető, illetve, hogy a maga a hívás melyik fázisban van. Pl: felvette-e már az utast a lift, vagy még meg kell állnia a szinten.

CallsController:

A nagyházi azon osztálya, ami megjeleníti a szimulációban résztvevő hívásokat. Itt lehet a már meglévő hívásokat törölni, módosítani. Valamint itt történik az új hívások bevitele is.

Elevator:

Egy liftet reprezentáló osztály. Minden liftnek be lehet állítani az egyéni attribútumait, mint pl: hány fő fér el benne, mennyi a gyorsulása, valamint azt is, hogy melyik szintekre közlekednek.

ElevatorController:

Tárolja a benne lévő fólifteket, és közöttük kiosztja a beérkező kéréseket. Fel tudunk venni további lifteket is, amennyiben a meglévőek nem lennének elegendőek, azonban az itt felvitt liftek nem fognak megjelnni a grafikusan, csak a konzolban látjuk nyomaikat.

Main:

Nem sok feladata van, csak annyi, hogy megjelenítsen egy Jframe objektumot (*Simulation)*, ami a későbbiekben fontos szerepet kap.

Simulation:

A házinak az összefogásáért felelős, Megjeleníti a fő ablak komponenseit, beleértve a menüt, az időzítőt, a vezérlő gombokat és az alul lévő szöveges mezőt, ahol a vizuálisan kevésbé ábrázolható részletek jelennek meg.

Timer:

A saját timer osztályom, ami nem tud túl sokat, csak annyit, hogy egy időt reprezentál, ami a szimulációban nagyon fontos szerepet tölt be. Minden időnek 3 attribútuma van: óra, perc, másodperc.

Enumerációk

Call.Status:

Egy hívás státuszát mutatja. Lehet: Amikor hívták a liftet, de még nem érkezett meg, vagy már be is szálltak vagy pedig az, ami már ki is lett szolgálva (kiszállt az utas). Az utolsónak az a jelentőssége, hogy a szimuláció végeztével össze lehet számolni, mennyi és milyen hívásokat szolgált ki az adott lift.

Elevator.Status:

A liftek állapotát jelzi. Vagy vannak benne hívások, és éppen dolgozik, vagy pedig éppen nincsen utasa, így szabadon mehet a következő hívás helyére. Ennek a grafikus megjelenése a liftek *nevének* háttérszíne (zöld/piros).

Kivételek

WrongFloorNumberException:

Bár a végső alkalmazásba nem sikerült használni, csak a tesztelés során volt jelentősége.

Fontosabb függvények

Call.compareTo:

Implementálni kellett, mert a *Call* osztály megvalósítja a Comparable interfészt.

CallsController.CallsController:

Vizuálisan megjelníti a beolvasott hívásokat egy Jtable-ben. Valamint szerkeszthetővé teszi azokat. Lehetőséget biztosít a meglévő hívások közül bármelyiket törölni, valamint itt tudunk hozzáadni új hívást a szimulációnkhoz.

Minden módosítás után újra kiírásra kerülnek a hívások.

CallsController.save/backUp:

A fájlba írás és olvasásért felelős függvények. Itt történik a szerializálás, valamint értelemszerűen a visszaolvasása is.

Elevator.timeBeetweenTwoFloor:

Kiszámolja, hogy a lift adott tulajdonságai mellett (gyorsulás/maximális sebessége) mennyi ideig tart neki elérnie egyik emeletről a másikra (nincs közte megálló). Fontos figyelembe venni, hogy a gyorsulás az csak a maximális sebességig történhet, ezután konstans sebességgel kell hogy továbbhaladni. Szintén oda kell figyelni, hogy a le is kell lassulnia a liftnek a cél állomás előtt.

Elevator.calculateArrivalTime:

Egy doublet ad vissza, ami másodpercben jelenti, mennyi ideig tart a jelenlegi pozíciójából és jelen körülmények között (benne lévő hívások) oda érnie a hívó szintjére. Nagyon bonyolult és összetett függvény, mert sokmindent figyelembe kell venni, és nagyon sok eset lehetséges. Hazudnék, ha azt mondanám, hogy tökéletesen működik.

Elevator.runSimulation:

A már benne lévő hívások alapján modellezi, hogy merre, kell mennie. Itt történik az attribútumok frissítése. (aktuális sebesség, benne lévő utasok száma, aktuális szint).

ElevatorController.addNewCall:

Ha beérkezik egy új hívás, akkor végigmegy az összes benne tárolt liften, és megnézi, a korábban említett függvények segítségével, hogy melyik lift érkezne legkorábban. Ha megvan a leggyorsabb, akkor fel veszi annak a híváslistájába.

Simulation.initCalls:

Beolvassa a fájlban lévő hívásokat.

Simulation.addComponents:

A grafikus megjelenítésért felelős. Elég hosszúra (~210 sor) sikerült a függvény, ráadásul nehezen lehetne hozzávenni egy új elemet, köszönhetően a szerencsétlenül választott layoutmanagernek.

Simulation.refresh:

A korábban kirajzolt elemeknek a feliratát frissíti minden iterációban.

Simulation.actionPerformed:

Lekezeli a beérkező szimuláció indítás és vége parancsokat, majd továbbítja a *run* metódusához.

Simulation.run:

A thread indításakor meghívódó függvény. Feladata: elindítani a liftek szálait, visszaállítani a liftek tulajdonságait, hogy a korábbról benne maradt adatok ne legyenek zavaróak. Beolvassa a hívásokat, időszerint növekvő sorrendbe rendezi, majd amikor a timer eléri a hívás idejét, akkor kiadja, hogy jött egy új hívás, amit le kell kezelni a lifteknek.

Timer.addSecond:

Növeli az objektumot egy másodperccel, figyel arra, hogy helyesen kezelje a “túlcsordulásokat”.

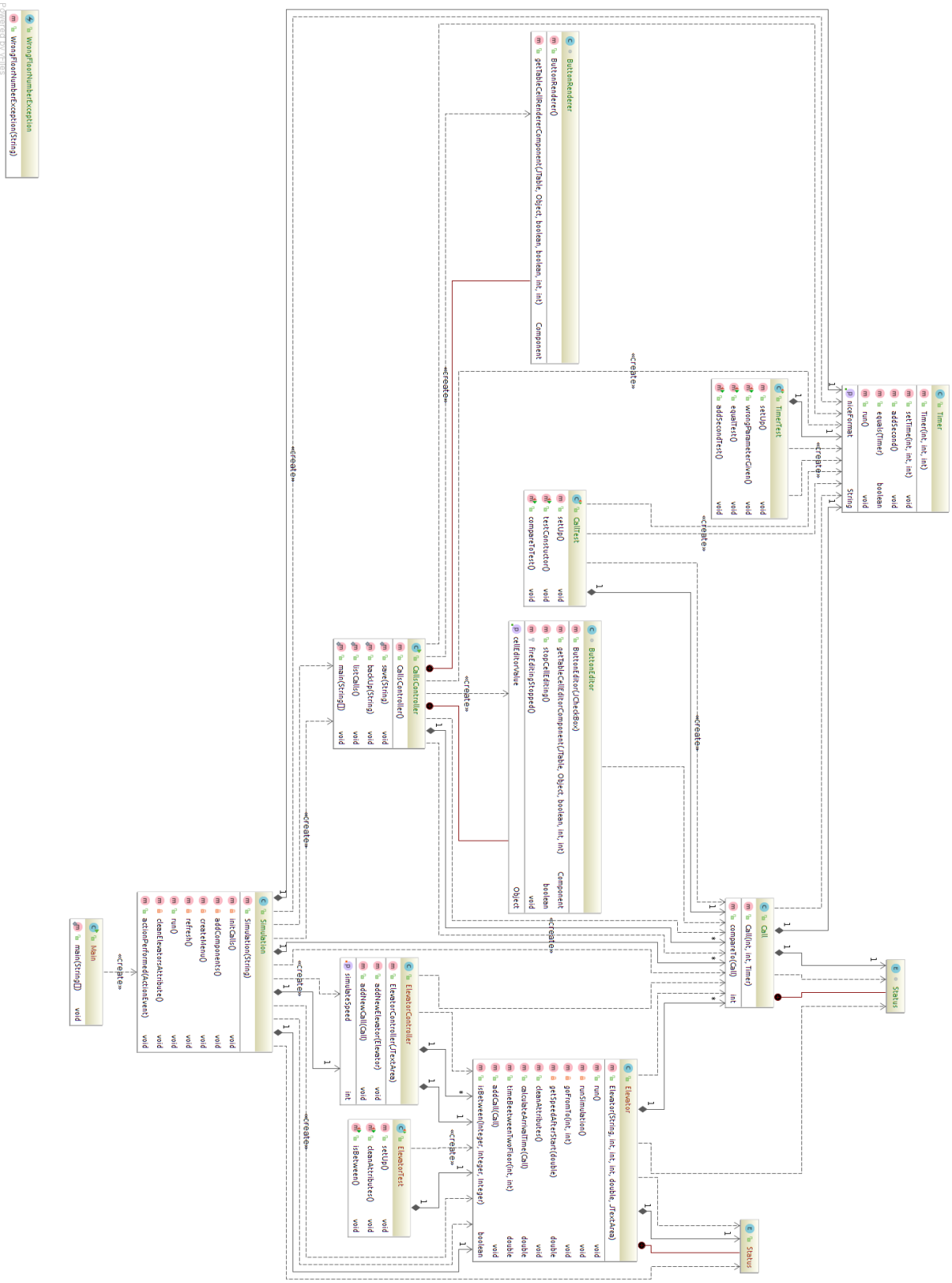
Timer.equals:

Visszaadja, hogy a két *Timer* értéke egyezik-e. Fontos szerepe van a szimuláció összehangolásánál.

Timer.getNiceFormat:

Visszaadja *Stringként* megformázva az objektum értékét.

Osztálydiagram



Felhasznált eszközök:

- **IntelliJ IDEA** 2017.2.5
  - JRE: 1.8.0_131-b11 amd64
  - JVM: Java HotSpot(TM) 64-Bit Server VM by Oracle Corporation
- **LibreOffice Writer**
- Rengeteg **Google** (stackowerflow.com)
- github.com (bár ennek ellenére is egyszer megijedtem, hogy mi romlott el, mert full meghalt az egész program)
- A végleges projekt ~1500 soros lett :)