Szoftvertechnológia és – technikák
4. gyakorlat – szekvenciadiagram

1. A gyakorlat menete

A gyakorlat első felében két vezetett feladat lesz, utána pedig következnek az önálló feladatok. Az első vezetett feladatban egy meglévő Java kód alapján fogunk papírra szekvencia diagramot rajzolni közösen. A második vezetett feladat keretein belül egy szöveges specifikáció alapján osztálydiagramot készítünk, majd a dinamikus működés leírására pedig egy szekvenciadiagramot. A gyakorlat második felében két önálló feladat lesz, amelyek a szekvenciadiagram és a kód kapcsolatára világítanak rá.

2. Vezetett feladatok

Szekvencia diagram készítése Java kód alapján – bemelegítő feladat

Rajzoljunk szekvenciadiagramot az alábbi Java kódrészlet alapján! A szekvenciadiagram a WashingMachine objektum **startMachine()** metódusának meghívásával kezdődjön! Tételezze fel, hogy a **startLogging()** metódus aszinkron módon hívható meg!

```java
public class WashingMachine {
 private Engine engine;
 private Timer timer;
 private WashOption washOp;
 private Logger logger = new Logger();

 public WashingMachine(Engine e,
            Timer t, WashOption w) {
  engine = e;
  timer = t;
  washOp = w;
 }

 public void startMachine() {
  logger.startLogging();
  int op = washOp.getWashSelection();
  switch (op) {
   case 1:
    standardWash();
    break;
   case 2:
    twiceRinse();
    break;
  }
  engine.turnOn();
  timer.start();
  engine.turnOff();
 }


 public void standardWash() {
  timer.setDuration(3600);
 }


 public void twiceRinse() {
  timer.setDuration(7200);
 }
}
```

```java
public class Engine {
 public void turnOn() {}
 public void boost() {}
 public void turnOff() {}
}

public class Timer {
 private int duration;
 private int time;

 public void start() {
  time = 0;
  while (time <= duration) {
   tick();
  }
 }

 public void tick() {
  time++;
 }

 public void setDuration(int d) {
  this.duration = d;
 }

 public int getDuration() {
  return duration;
 }
}

public class WashOption {
 public int getWashSelection() {…}
 public void runTestMode() {…}
}

public class Logger {
 public void startLogging(){…}
 public void reset(){…}
}
```

Furious Flights Online check-in rendszer – osztálydiagram készítése

Modellezzük az Furious Flights online check-in rendszerét osztálydiagram segítségével!

FELADAT: A Furious Flights repülőgép társaság szeretné kiépíteni online check-in rendszerét, mert a visszajelzések alapján az utasok nagy része kifogásolta az online csekkolás lehetőségének hiányát. Az Furious Flights minket kért meg a rendszer modellezésére. A megrendelővel történő beszélgetés után a modellezés szempontjából fontos információkat pontokba szedve a rendelkezésünkre bocsájtották a rendszertervezők:

CheckInWebDesk: Egy *CheckInWebDesk*-hez egy *SeatReservationSystem* tartozik és egy *CheckInWebDesk*-et tetszőleges számú utas használhat. Műveletek:

- Furious Flights-hoz kapcsolódó hirdetések megjelenítése (privát művelet)
- A felhasználó bejelentkeztetése.
- Adott repülőgépen található ülések megjelenítése. (privát művelet)
- Felhasználási feltételek megjelenítése. (privát művelet)
- Információk megjelenítése egy konkrét járatról (privát művelet)
- Ülésfoglalási folyamat elindítása.
- Ülésfoglalás feldolgozása. (privát művelet)
- Ülésfoglalási eredmény megjelenítése. (privát művelet)
- Beszállókártya (BoardingPass) kiküldése e-mailben (privát művelet)

Passenger: A rendszer megvalósítása során az utasokat is le kell modellezni. Az utas egyszerre csak egy *CheckInWebDesk*-et használhat. Egy utashoz egyszerre csak egy ülés tartozik. Műveletek:

- Felhasználási feltételek elfogadása. A művelet igazzal tér vissza, ha a felhasználási feltételek elfogadásra kerültek.
- Ülés kiválasztása.
- Értesülés a sikeres ülésfoglalásról.
- Értesülés a sikertelen ülésfoglalásról.

SeatReservationSystem: Az ülésfoglalás kezelésért felelős rendszer. Műveletek:

- Az ülésfoglalás validálása a rendszerben.
- Ülésfoglalás regisztrálása a rendszerben. Ennek két eredménye lehet: sikeres vagy hiba (privát művelet)

ReservationStatus: Enumárció, amely két állapotot tárol az ülésfoglalás sikerességéről: sikeres vagy hibával záruló ülésfoglalás.

Seat: Az ülést reprezentáló osztály. Egy ülés egyszerre csak egy utashoz tartozhat.

Furious Flights Online check-in rendszer – szekvenciadiagram készítése

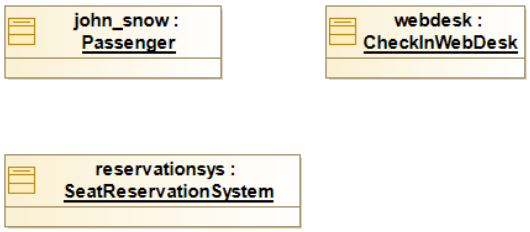
Modellezük a rendszer működésének dinamikus viselkedését szekvenciadiagram segítségével!

FELADAT: A rendszer megvalósítása során az utasokat is modellezük le. A teljes folyamat minden lépését az utaskezdeményezi, ő, mint szolgáltatást használja a webes check-in rendszert. Ez a következő lépéseket foglalja magába:

1. Az utas belép a check-in rendszerbe.
   1.1. A webes felülelet megjeleníti a Furious Flights hirdetéseit az akciós repülőutakra vonatkozóan.
   1.2. Az előző ponttal párhuzamosan megjelennek a járattal kapcsolatos információk.
2. Az előző pont után rendszer addig jeleníti meg a felhasználási feltételeket, amíg az utas el nem fogadja azokat.
3. Az utas jelzi a felületen, hogy szeretne ülést foglalni a repülőgépen.
4. A webes felület megmutatja a repülőgépen található üléseket.
5. Az utas kiválasztja a számára megfelelő ülést.
6. A webes felület feldolgozza a foglalás tényét.
7. A webes felület továbbítja az igényt az ülésfoglalási rendszernek.
8. A rendszer megpróbálja beregisztrálni az ülésfoglalást. Ennek két eredménye lehet: sikeres vagy hiba.
9. A webes felület visszajelzést ad a foglalás sikerességéről.
   9.1. Ha a foglalás sikeres volt, akkor a sikeres ülésfoglalásról tájékoztatja az utast, illetve elküldi az utas e-mail címére a beszállókártyát.
   9.2. Ha a foglalás sikertelen volt, akkor a sikertelen foglalásról tájékoztatja az utast.

TIPPEK A SZEKVENCIADIAGRAM ELKÉSZÍTÉSÉHEZ:

- Az osztálydiagram elkészítése után készíteni kell egy objektum diagramot is, hiszen a szekvenciadiagram épít az objektumdiagramra (Create Diagram > Object Diagram)
- Az objektumdiagramra elég csak feldobálni az osztályok példányait, a feladat szempontjából nem kell foglalkozni a köztük lévő linkekkel:



- A LifeLine behúzása után be kell állítani a Represented By Property-t, ami egy objektumra kell, hogy mutasson. Ezután érdemes behúzni az üzeneteket.
- Figyeljünk a szinkron és aszinkron hívások pontos jelölésére (Synchronous/ Asynchronous message)
- A Modelio nem szereti, ha felvesszük az execution specification-öket (dobozokat) és utána húzzuk be a metódushívásokat, jobban szereti, ha behúzzuk a hívást és ő veszi fel nekünk az execution specification-öket. Ez minden fajta elemnél igaz.
- Az üzeneteknél egyszerűen ki tudjuk jelölni a meghívott metódust az Invoked property beállításával.
- A szinkron műveleteket lefelé nyújtani úgy lehet, ha a return message-et lejjebb húzzuk (nyúlik vele a doboz).
- A kombinált részletek (Combined Fragment) behúzása után a Property-k között be lehet állítani az Operator típusát (Seq, Alt, Loop).
- A gyakorlat folyamán használandó Combined Fragment típusok:
  - Alt: Feltételes elágazás (if – else if – else szerkezet)
  - Loop: Feltétel szerinti elöltesztelős ciklus

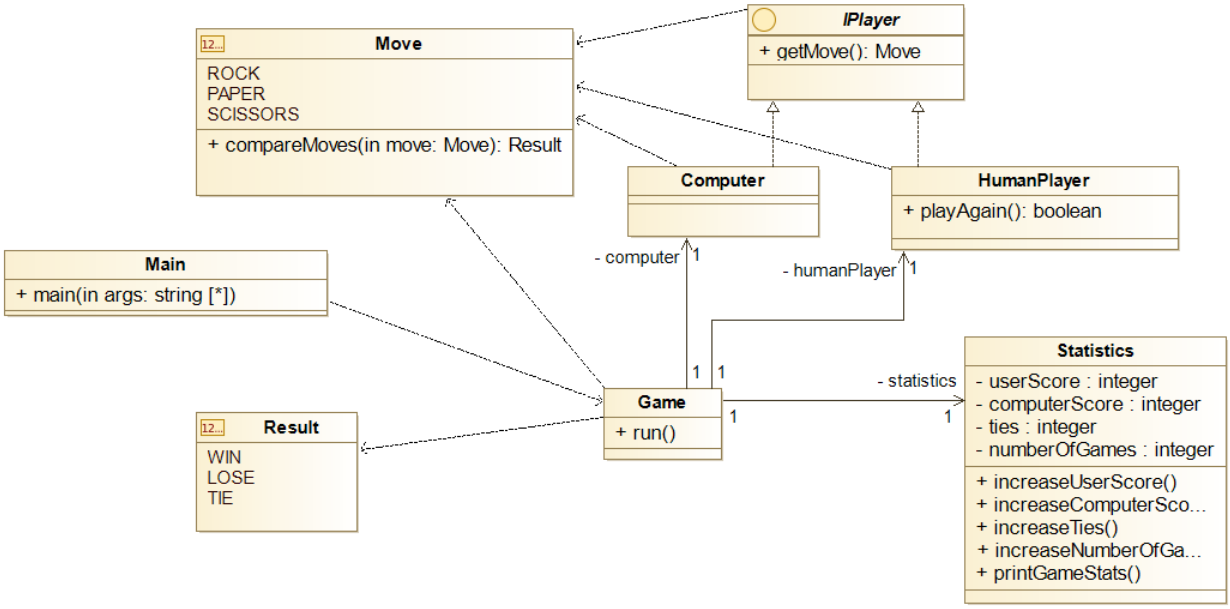
    - Opt: Opcionális utasítások (else ág nélküli if)
    - Par: Párhuzamosan végrehajtandó utasítások
- Egy Combined Fragment-be több Interaction Operand-ot is tudunk húzni. Erre szükség lehet például a párhuzamosan végrehajtandó utasítások jelölésénél (Par).
- Ne felejtsük el beállítani a Interaction Operand-ok Guard property-jét, ahol szükséges (pl. a Loop feltételénél).

3. Önálló feladatok

Kő-papír-olló (hiányos implementáció kiegészítése szekvenciadiagram alapján)

Egy játékfejlesztő cégnél egy egyszerű konzolos kő-papír-olló játékot fejlesztenek. Az implementáció nagy része elkészült, azonban vannak olyan részei kódnak, amelyek hiányosak. A cég senior fejlesztője elkészítette a játék architektúrális leírását egy osztálydiagram formájában, a hiányos részekhez pedig egy szekvenciadiagramot készített, amely alapján egyértelműen befejeztető az implementáció.

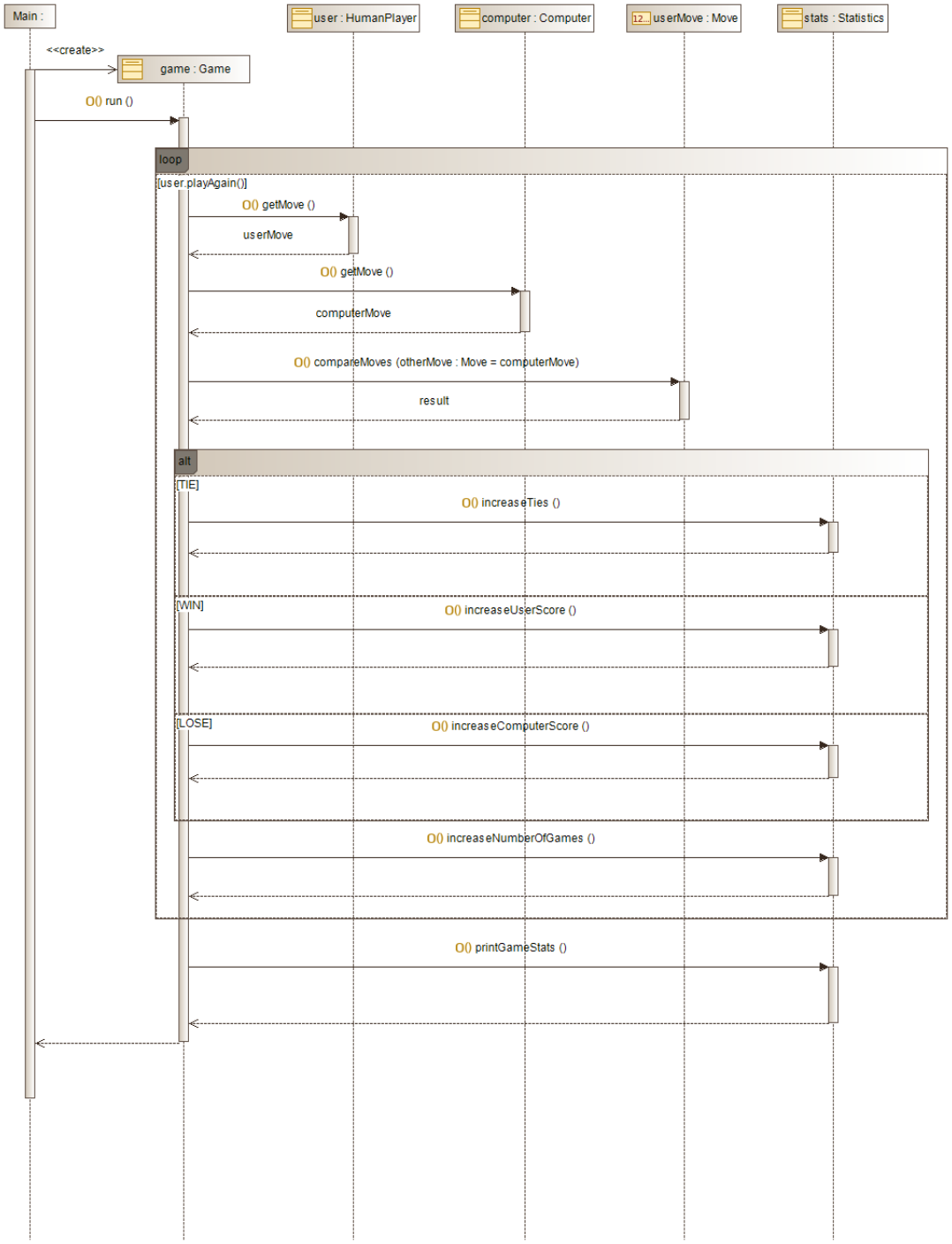
Először tekintsük át és értelmezzük a játék felépítését az osztálydiagram alapján:



A kiinduló Java projekt megtalálható a *RockPaperScissors* néven. A feladat a *TODO*-val jelölt részek kiegészítése a kódban, amely két osztályra vonatkozik:

- Main
- Game

FELADAT: Értelmezd az alábbi szekvenciadiagramot, amely alapján egészítsd ki a hiányos implementációt. A cél, hogy a labor végére egy kipróbálható és működő játék szülessen. A könnyebb áttekinthetőséget segítheti a hallgatói laboranyagban kiadott *Modelio* projekt, amely tartalmazza az osztálydiagramot és a szekvenciadiagramot is.



Szekvenciadiagram készítése kód alapján

A *Smart Industries* autógyártó cégnél robotok végzik az autók gyártásának bizonyos folyamatait. A robotok egy futószalag mellett dolgoznak, és különböző munkafolyamatokat végeznek az autón (karosszéria összerakása, kerek felszerelése, visszapillantótükrök felszerelése, valamint festés). Vannak olyan folyamatok, amelyeket egymással párhuzamosan végeznek (kerek és visszapillantótükrök felszerelése), és vannak olyan folyamatok, amelyek szigorúan csak egymás után következhetnek (pl. karosszéria összerakása után következik a kerekek és visszapillantó tükrök felszerelése). A gyártósor implementációját

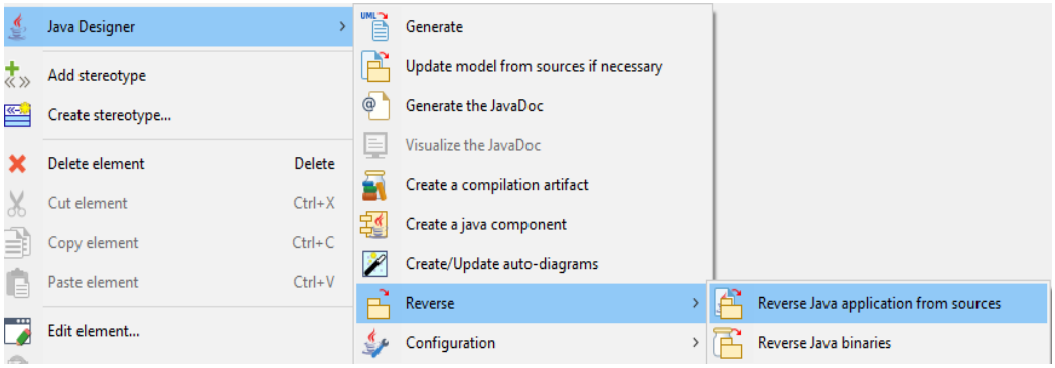
egy fejlesztő cég már elkészítette, azonban a szálkezelés miatt nehézkes megérteni az implementáció részleteit. A Java projekt megtalálható *AutomatedCarFactory* néven, a kiadott anyagok között.

FELADAT: Tekintsd át az *AutomatedCarFactory* implementációját és készítsd el hozzá az osztálydiagramot és a működést leíró szekvenciadiagramot. A jelöléseknél vedd figyelembe az aszinkron hívásokat, valamint a párhuzamos részeket is.

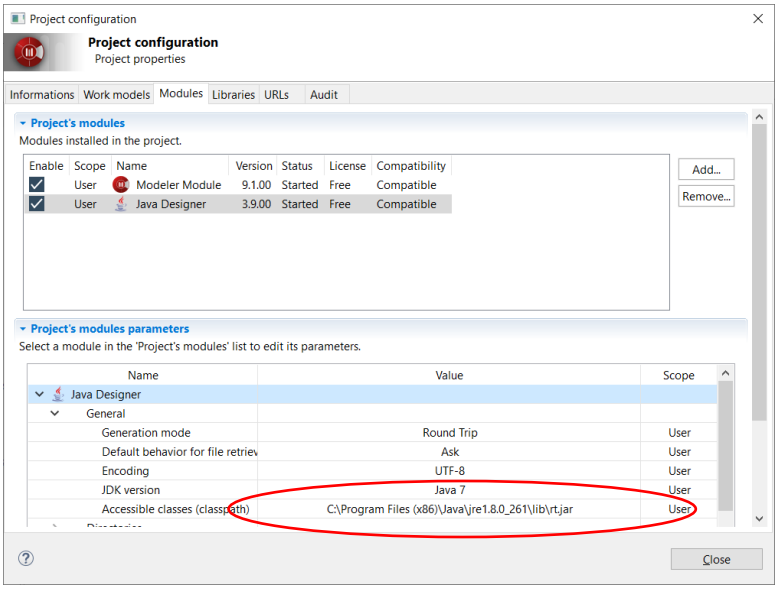
TIPP: *Kis emlékeztető a szálkezeléshez:*

- *Szálak definiálása: Leszármazás a Thread osztályból vagy Runnable interfész megvalósítása.*
- *Szálak elindítása a Thread objektum start metódusával történik.*
- *A join metódus lehetőséget biztosít arra, hogy egy szál egy másik befejeződésére várjon. Azaz, ha például t egy futó szál t.join() hívás esetén az aktuális szál végrehatjása szüneteltetésre kerül, amíg t befejeződik.*
- *A run() metódus a szálban végrehajtandó kódot tartalmazza.*

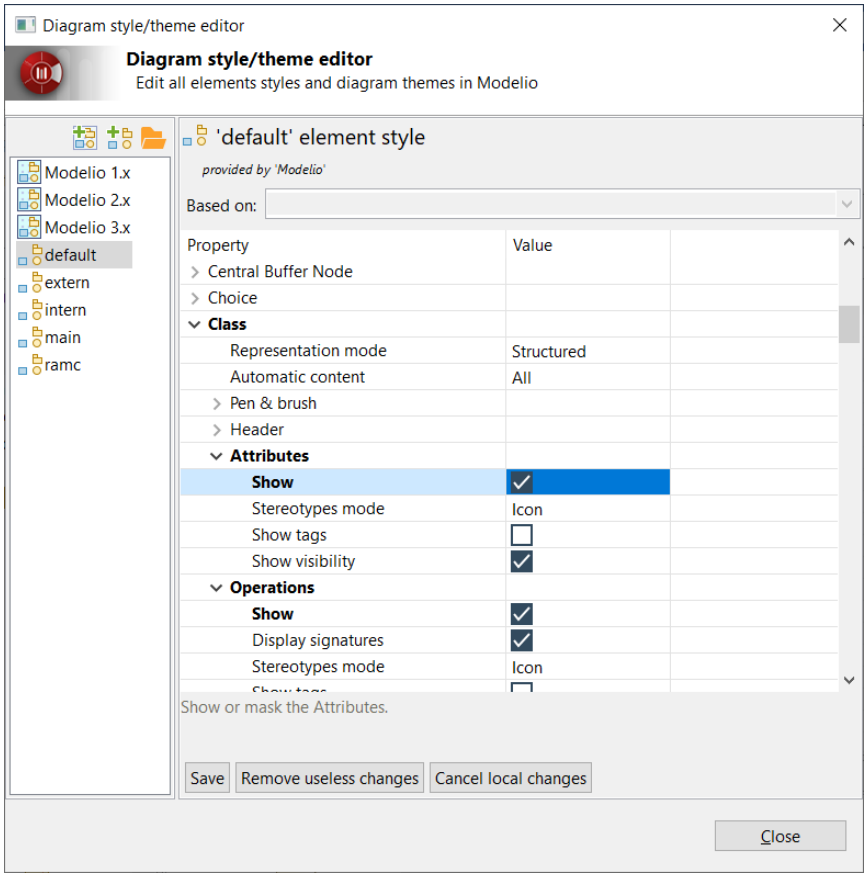
TIPP*: Az osztálydiagram gyors elkészítéséhez használjuk a Modelio Java Designer Reverse Java application from sources funckcióját.*



- Szükség lehet arra, hogy a Java Designer beállításain módosítsunk: Configuration ->Modules->Java Designer. Ellenőrizzük, hogy a General alatt az Accessible Classes beállításnál megtalálható-e az rt.jar helyes elérési útvonala:



- Az adott package-en jobb klikk >Java Designer> Reverse >Reverse Java application from sources, majd a projekt könyvtárát keressük ki, és pipáljuk ki a checkboxot az src folder mellett
- OK után nézzük meg a generálódott model elemeket a bal oldali fában
- A modelt ábrázolni akarjuk osztálydiagramon, ehhez jobb klikk az adott package-en, Create Diagram… > Class diagram
- Húzzuk rá a fából az elemeket, fontos, hogy ha egy osztályt ráhúzunk, attól a mezők/műveletek/asszociációk nem kerülnek rá by default, azokat is ki kell jelölni és behúzni. (Többes kijelölés megy Shift+klikkel)
- **Javaslat1**: Configuration->Diagram Themes->default->Class-t kiválasztva a listából, megadható, hogy az attribútumok és metódusok automatikusan jelenjek meg, ha bepipáljuk a Show beállítást:



- **Javaslat2**: a műveleteket ne húzzuk be, mert azok úgyis csak getter/setterek. Szintén ne húzzuk rá a konstruktorokat, mert azok magukkal hozzák az összes műveletet. (Ctrl+Z-vel mindent visszavonhatunk, ha esetleg véletlenül bejöttek a getterek/setterek is.)