Szoftvertechnológia és – technikák
3. gyakorlat – osztálydiagram, kódgenerálás alapok

1. A gyakorlat menete

A gyakorlat első felében egy autókölcsönző működését modellező osztálydiagramot készítünk el közösen egy szöveges specifikáció alapján. Az osztálydiagramot felhasználva kódot generálunk Java nyelvre a Modelio segítségével. A Modelio licenszelt változatában lehetőség lenne C# kódot is generálni, azonban a labor keretein belül az ingyenes változatot használjuk, így C# kódot nem fogunk generálni, hanem a tárgy honlapjáról töltjük majd le. A generált Java kódot a C# kóddal összehasonlítjuk, majd bizonyos módosításokat eszközölve a modellen megnézzük, hogy milyen változások történtek a Java kódban. Ezt követi egy reverse engineering feladat, ahol egy leegyszerűsített webshop Java forráskódjából kiindulva generálunk osztálydiagramot. Innentől a labor további része önálló feladat

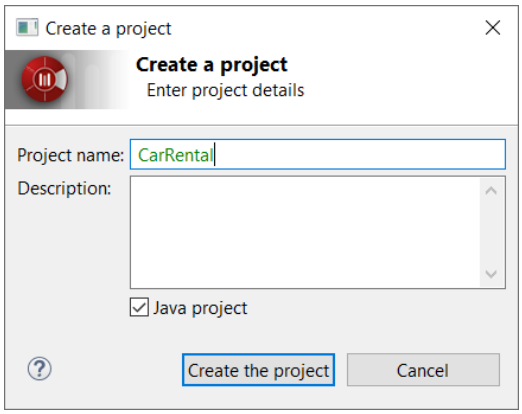
2. Vezetett feladatok

Autókölcsönző – osztálydiagram készítése

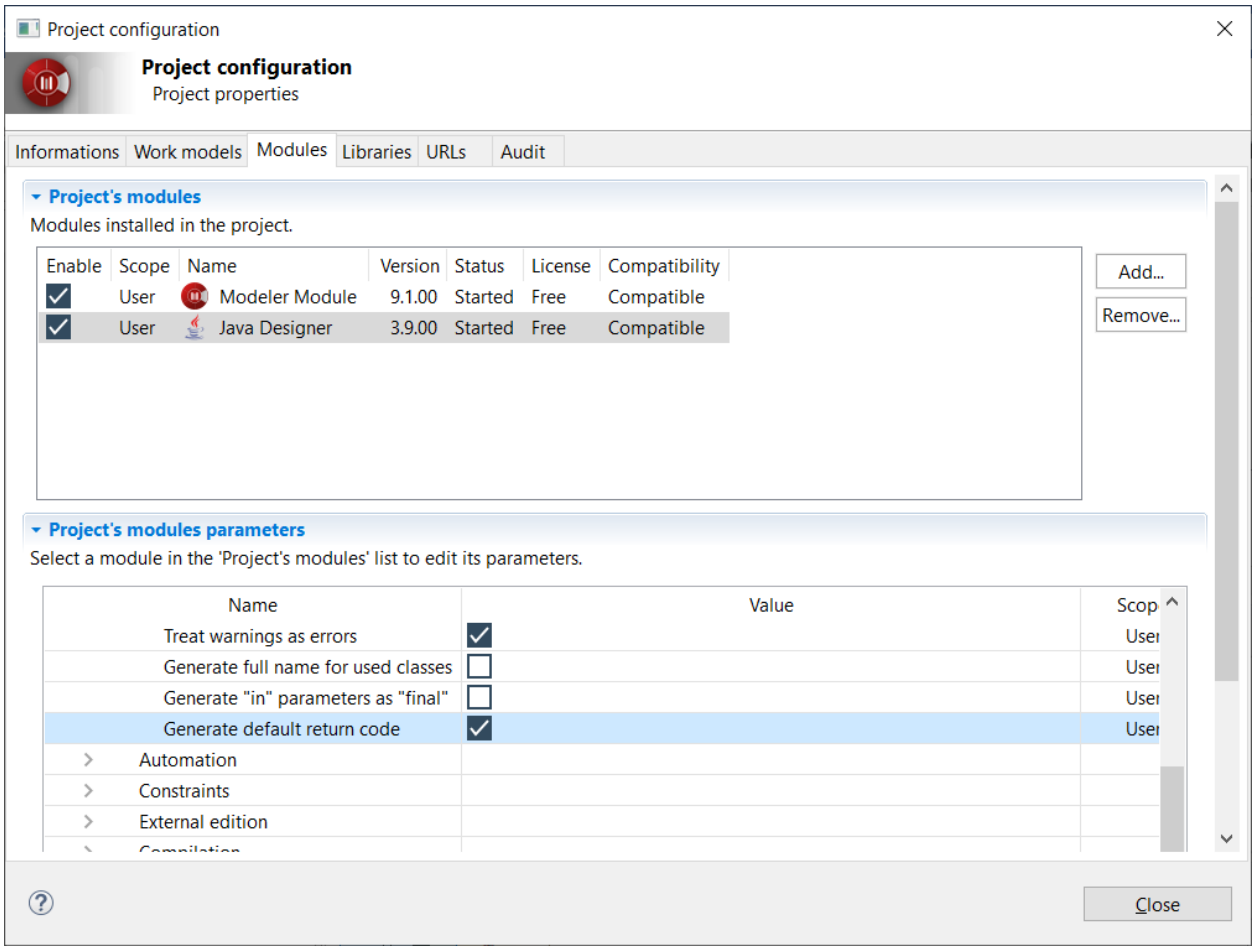
Készítsünk közösen osztálydiagramot Modelio-ban az alábbi szöveges leírás alapján!

Tippek a Modelio használatához

- File > Create Project … > a projekt neve legyen *CarRental*
- **Pipáljuk be a Java project beállítást**!



- CarRental > CarRental> CarRental folderen jobb klikk, Create diagram… Class diagram
- Ha esetleg elfelejtettük bepipálni a Java project beállítást a Configuration menüpontból a projekt létrehozása után is hozzáadhatjuk a Java Designert: Configuration ->Modules->Add…->Válasszuk ki a Java Designer-t!
- Ellenőrizzük a Modules ablakban, hogy a Java Designer beállításainál be van-e pipálva a Generate default return code lehetőség.



- Készítsük el az osztálydiagramot a szöveges specifikáció alapján! (Hasonlóan az előző gyakorlathathoz.)

FELADAT:

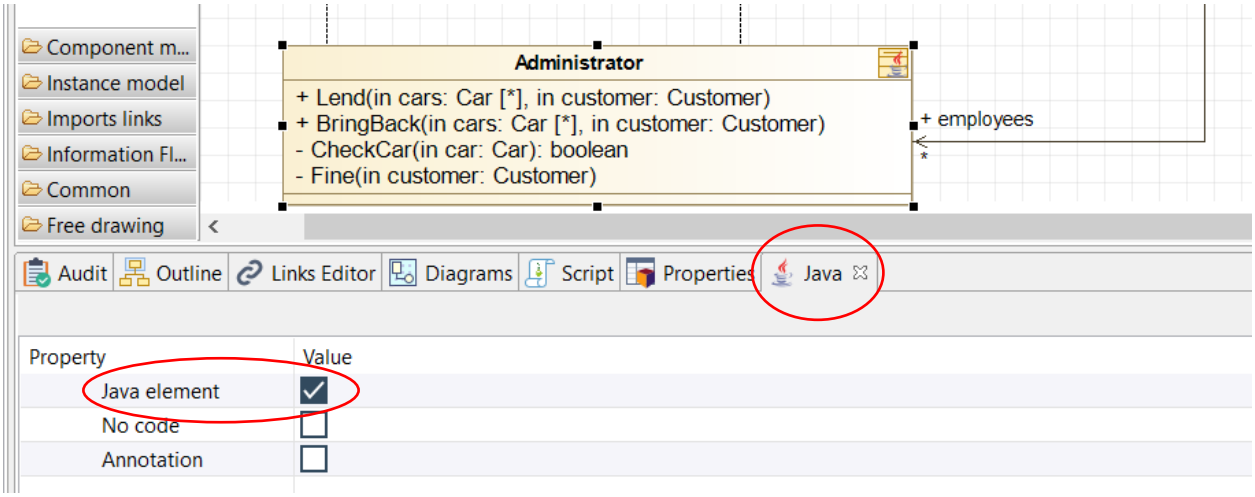
Egy autókölcsönző nyilvántartási rendszerét modellezük. Az autókölcsönzőben (**CarRental**) természetesen vannak autók (**Car**), amelyekről tároljuk a gyártóját, a rendszámát és az alvázszámát. Ezen kívül az autónak típusa (**CarType**) is van, ami lehet sedan, SUV, pickup vagy roadster. A gyártó és/vagy típus alapján képesek vagyunk az autókölcsönzőn keresztül autót keresni. Az autókölcsönzőnek vannak ügyfelei (**Customer**), akik autókat kölcsönözhetnek. Egy ügyfél egyszerre több autót is kölcsönözhet. Emellett az ügyfél elő is jegyezhet magának autót, de csak egyet. Az ügyfelekről tároljuk a személyi számát és a születési dátumát. Az ügyfélnek és az autókölcsönzőnek is van neve és címe (irányítószám, város, utca), amelyeknél fontos követelmény, hogy később *egységes formában* legyenek lekérdezhetőek. Az autókölcsönzőben ügyintézők (**Administrator**) dolgoznak, akiken keresztül az ügyfelek képesek autókat kölcsönözni és visszahozni. E két művelethez kapcsolódóan az ügyintéző képes a kölcsönzés során ellenőrizni, hogy az autó kikölcsönzése lehetséges-e, illetve késedelmes visszahozatal esetén képes az ügyfelet megbüntetni.

Autókölcsönző – kódgenerálás

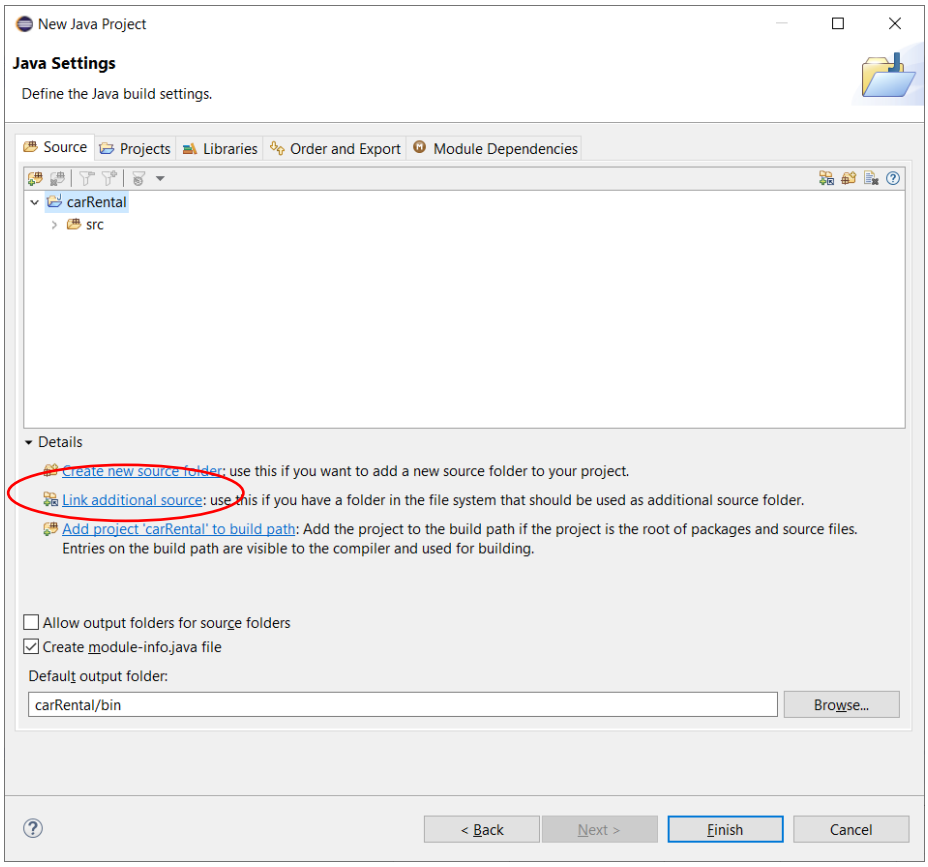
Generáljuk kódot az elkészült osztálydiagramból (Java), majd elemezzük közösen a C# és Java kódot!

A Java kódgenerálással kezdünk:

- Minden **osztályt, enumerációt, interfészt és csomagot** Java elemmé kell tennünk. Ezt az adott elem kiválasztásával, majd a Java fülre navigálásával tehetjük meg, ahol pipáljuk be a „Java element" beállítást. Az interfészeknél szükség lehet a modell frissítésére (Update model from the interface… felugró ablak). Ebben az esetben válasszuk a „Yes" lehetőséget. Így az interfész metódusai is megjelennek az osztályokban, ami az osztálydiagramon redundanciát okoz, viszont a kódgenerálásnál így fognak helyesen bekerülni az egyes metódusok az osztályokba.



- A CarRental > CarRental> CarRental folderen jobb klikk >Java Designer >Generate. A forráskód ide generálódik: c:\Users\…\modelio\workspace\CarRental\src (**Fontos:** Ha mégsem találjuk a generált fájlokat ezen az elérési útvonalon, akkor használjuk a Help->Open Log File lehetőséget, ahol ki tudjuk keresni a valós elérési útvonalat!) Ezt az IDE-ben be kell linkelni, mint source folder, a következő módon:
- Eclipse-ben File > New > Java Project (Ha nincs by default ott, akkor Project… és ott kikereshető)
- Projekt neve pl. carRental
- Next után a Source tabon lesz egy *Link additional source* link:



- Kattintás után ki kell browse-olni a foldert, ahova a Modelio generál: C:\Users\...\modelio\workspace\CarRental\src. Az eclipse-projekt beli nevet is adjuk meg, pl. modelio-src
- A generált forrásban lesznek modelio-s annotációk, amik a forrás és modell közti kapcsolatot definiálják. Hogy ezek is lefordulhanak, a build path-ba be kell tenni egy jar fájlt, ennek módja:
- Keressük ki ezt a fájlt: c:\Users\...\modelio\workspace\CarRental\data\backups\modules\JavaDesigner\JavaDesigner_3.9.00.jmdac
- Ez egy tömörített fájl, csomagoljuk ki valahová
- Jobb klikk Eclipse-ben a projekten > Build Path > Configure build path > Libraries tab, Add External JARs, majd válasszuk ki a kicsomagolt könyvtárban ezt a fájlt: \JavaDesigner\bin\javadesigner.jar

C# kód: A Modelio licenszelt változatában lehetőség lenne C# kódot is generálni, azonban a labor keretein belül az ingyenes változatot használjuk, így C# kódot nem fogunk generálni, hanem töltsük le a tárgy honalpjáról (**CarRentalModositasElott**)! A C# kód a Modelio C# designerével lett generálva.

A Java és C# kód elemzése és összehasonlítása:

Kezdjük az elemzést a **CarRental.cs** és **CarRental.java** fájlokkal:

C#:

```
[objingid("49e16132-8115-4f4a-90a1-689be4d40046")]
public class CarRental : IAdressedEntity, INamedEntity
```

```
{
    #region cars
    [objingid("4ff41c40-a4a7-4631-a7d8-daedf2b5e0af")]
    public List<Car> cars = new List<Car>();
    #endregion

    #region customers
    [objingid("320e9899-1bb2-4fec-912d-bdaac0b169fd")]
    public List<Customer> customers = new List<Customer>();
    #endregion

    #region employees
    [objingid("898abf9e-6076-403b-a02a-df0bfd0ef5c2")]
    public List<Administrator> employees = new List<Administrator>();
    #endregion

    [objingid("adff0b91-9afd-42c6-902e-0690cf0be081")]
    public List<Car> SearchCar(String manufacturer,CarType carType)
    {
    }

    [objingid("1e073611-7087-4bbf-a4a7-65801c95bc8d")]
    public string getZipCode()
    {
    }

    [objingid("ed140350-d945-4cc4-a03e-c56518b2241f")]
    public string getCity()
    {
    }

    [objingid("f9a20ee3-0647-4164-8b55-3094fcfe882d")]
    public string getStreet()
    {
    }

    [objingid("4c23529a-7a0c-498f-b6ee-cf3701a784d1")]
    public string getName()
    {
    }

}
```

Java:

```
@objid ("49e16132-8115-4f4a-90a1-689be4d40046")
public class CarRental implements IAdressedEntity, INamedEntity {
    @objid ("4ff41c40-a4a7-4631-a7d8-daedf2b5e0af")
    public List<Car> cars = new ArrayList<Car> ();

    @objid ("320e9899-1bb2-4fec-912d-bdaac0b169fd")
    public List<Customer> customers = new ArrayList<Customer> ();

    @objid ("898abf9e-6076-403b-a02a-df0bfd0ef5c2")
    public List<Administrator> employees = new ArrayList<Administrator> ();
```

```
    @objid ("adff0b91-9afd-42c6-902e-0690cf0be081")
    public List<Car> SearchCar(final String manufacturer, final CarType carType) {
        // TODO Auto-generated return
        return null;
    }

    @objid ("1e073611-7087-4bbf-a4a7-65801c95bc8d")
    public String getZipCode() {
        // TODO Auto-generated return
        return null;
    }

    @objid ("ed140350-d945-4cc4-a03e-c56518b2241f")
    public String getCity() {
        // TODO Auto-generated return
        return null;
    }

    @objid ("f9a20ee3-0647-4164-8b55-3094fcfe882d")
    public String getStreet() {
        // TODO Auto-generated return
        return null;
    }

    @objid ("4c23529a-7a0c-498f-b6ee-cf3701a784d1")
    public String getName() {
        // TODO Auto-generated return
        return null;
    }

}
```

Bár a képeken nem szerepel a lehetséges eltérések miatt, de fontos megjegyezni, hogy attól függően, hogy a modell elkészítésekor milyen package neveket használtunk, úgy C# kód esetén a namespace, Java kód esetén pedig a package neve fogja ezt tükrözni.

Ettől eltekintve láthatjuk, hogy a két nyelv szintaktikailag ebben az esetben nagyon hasonló, de természetesen lesznek szituációk, amikor ugyanennyire el is térnek egymástól.

A jelenlegi eltérések egyrészt az interfész implementációban fedezhetőek fel, hiszen C# esetén a Java-val ellentétben a „**:**” jelöli a megvalósítást az osztály neve után. Másrészt a típusokban találhatunk eltéréseket, hiszen C# esetén **List**-et és **string**-et használunk. Ezen kívül láthatjuk még a Java **annotációk** és C# **attribútumok** közötti különbséget is. A szintaktikai eltérések tisztázása után figyeljük meg, hogy az egyes UML konstrukciók hogyan képződnek le kódra.

Az interfészek megvalósítása egyértelmű, azaz az interfészekben definiált metódusok megvalósítása megtalálható az őket megvalósító osztályokban. Ezen felül az autókölcsönzőnek van egy-egy asszociációja az autókra, az ügyfelekre és az ügyintézőkre, ahol mindhárom asszociáció 0..* multiplicitású. Ennek megfelelően a kódban ez három lista típusú adattagot eredményez, mely az adott cél típust tárolja. Végezetül természetesen, ha az osztálydiagramban egy osztály definiál egy metódust vagy adattagot,

akkor az a kódban is az osztály adott nevű és paraméterezésű metódusa vagy adott típusú adattagja lesz, lásd **SearchCar(…)** vagy a **Car** osztály adattagjai.

Amennyiben az asszociáció 0..1 vagy 1..1 multiplicitású, akkor lista helyett a kódban egy egyszerű referenciát találunk, ahogyan azt az autó osztály is mutatja a **borrower** esetén:

C#:

```
[objingid("49b62373-134d-483d-83f7-064a2c907118")]
public class Car
{
    #region borrower
    [objingid("b0f6fdad-a872-4c15-808c-084b87a768a8")]
    public Customer borrower = null;
    #endregion

    #region customer
    [objingid("766c0612-bfd3-47e0-89cf-5d21bf36ea54")]
    public Customer customer = null;
    #endregion

    #region manufacturer
    [objingid("4862f9e7-3c19-4393-bc95-3dad3835947f")]
    public string manufacturer;
    #endregion

    #region plateNum
    [objingid("ab14795f-5532-4394-ac7e-dca2c7d15e42")]
    public string plateNum;
    #endregion

    #region chassisNum
    [objingid("6c5755b6-08f1-4950-9c48-2874d9ae6def")]
    public int chassisNum;
    #endregion

    #region carType
    [objingid("1e489c12-551a-46b0-8ec9-06ca63a841e4")]
    public CarType carType;
    #endregion

}
```

Java:

```
@objid ("49b62373-134d-483d-83f7-064a2c907118")
public class Car {
    @objid ("4862f9e7-3c19-4393-bc95-3dad3835947f")
    public String manufacturer;

    @objid ("ab14795f-5532-4394-ac7e-dca2c7d15e42")
    public String plateNum;

    @objid ("6c5755b6-08f1-4950-9c48-2874d9ae6def")
    public int chassisNum;
```

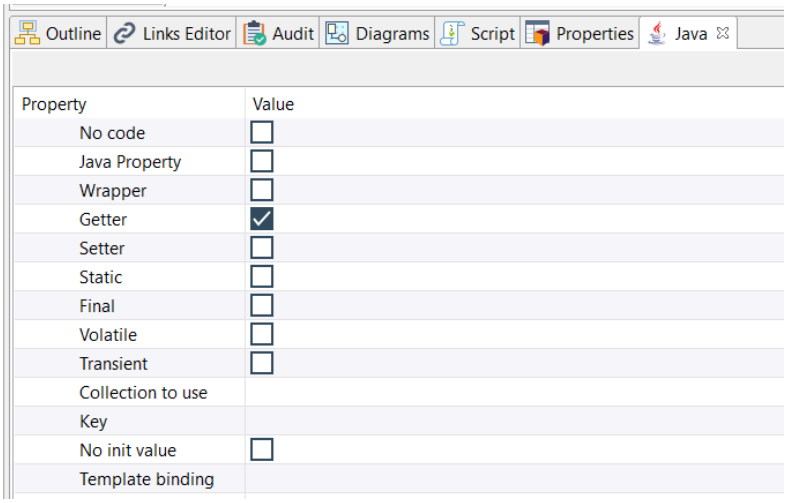
```
    @objid ("1e489c12-551a-46b0-8ec9-06ca63a841e4")
    public CarType carType;

    @objid ("b0f6fdad-a872-4c15-808c-084b87a768a8")
    public Customer borrower;

    @objid ("766c0612-bfd3-47e0-89cf-5d21bf36ea54")
    public Customer customer;

}
```

A többi osztályban kód szinten jelenleg nem találunk újdonságot, csak a **Customer** dátum típusú adattagjában fedezhetünk fel eltérést a két nyelv között: C# **DateTime**.

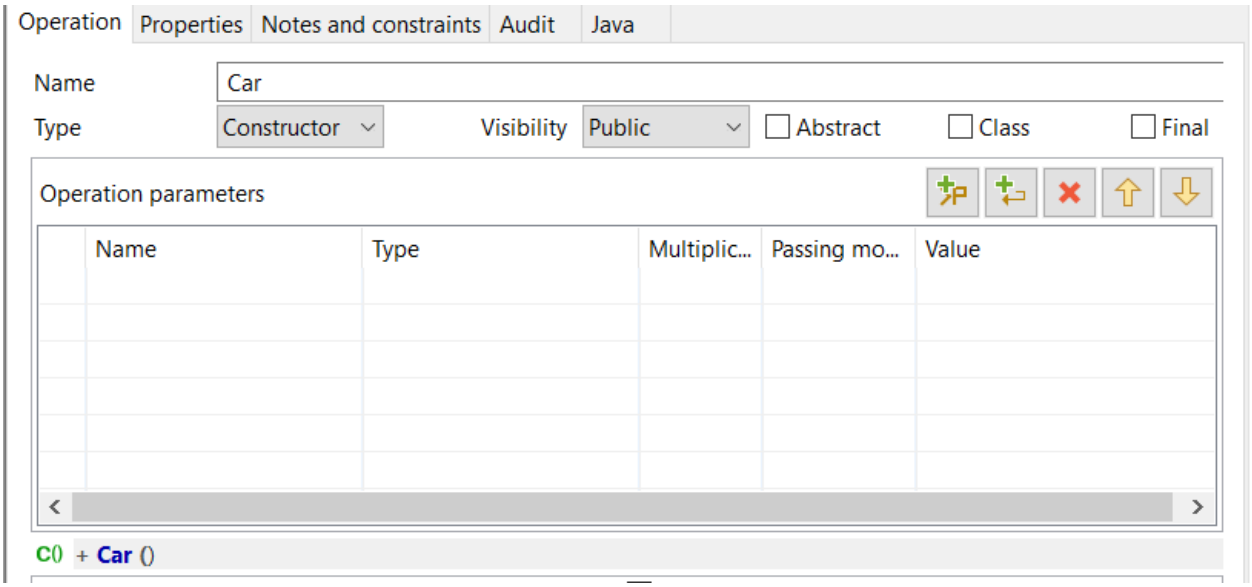
Autókölcsönző – módosítás

A kód vizsgálatát követően módosítsuk a modellt a funkcionalitás növelése és a Clean Code elvek követése érdekében. A módosítás során figyeljünk arra, hogy a Modelio nyelv specifikus eszközei képesek bizonyos változtatásokat automatikusan elvégezni (pl. getter és setter generálás).

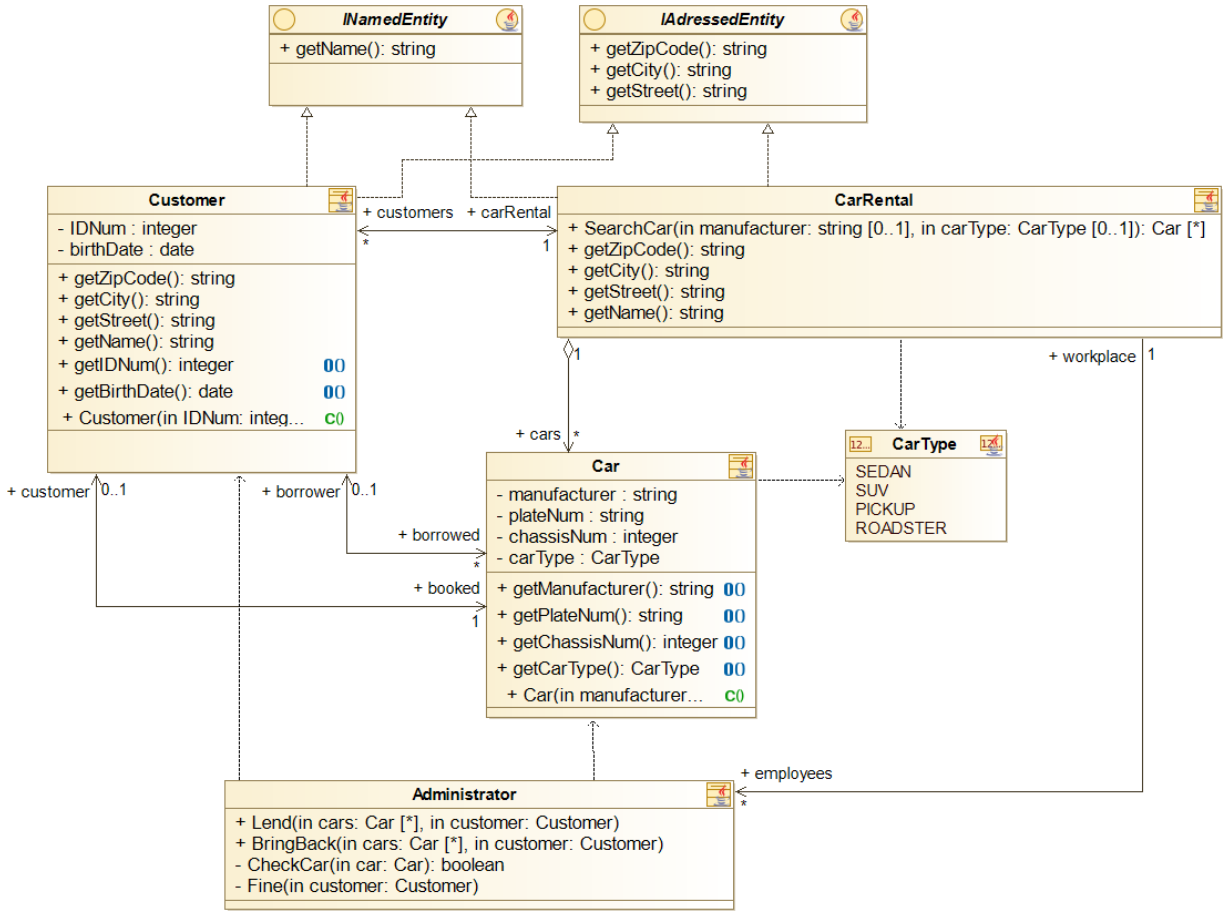
A változtatások során az első és legfontosabb, hogy próbáljuk meg azokat az attribútumokat rejtve tartani, esetleg csak olvashatóvá tenni, amelyeket nem szeretnénk, hogy el lehessen érni vagy validáció nélkül változtatni lehessen. Egy autó adatait, miután az autó hivatalosan legyártásra került és bekerült az autókölcsönző adatbázisába, ne lehessen módosítani. Hasónlóképpen érjük el azt is, hogy az ügyfél adatait se lehessen módosítani. Ennek megfelelően végezzük el a következő módosításokat a modellen:

- Rejtsük el az adattagokat a **Car** és **Customer** osztályokban. Ehhez jelöljük ki a megfelelő adattagot, majd a „Visibility” tulajdonságot állítsuk „Private” -ra
- Biztosítsunk getter metódusokat a privát attribútumok eléréshez és készítsünk egy konstruktort az autók és az ügyfelek létrehozásához.
- A getter generáláshoz a Java fülön pipáljuk be a megfelelő tulajdonságot (az attribútum kijelölése után Getter beállítás). Az JavaDesigner által készített Getterek mögött kék színnel O() felirat jelenik meg. A Java property-t *nem kell* bepipálni!
- 

- A konstruktorhoz készítsünk egy új metódust az osztályban, majd kattintsunk rá duplán és a megjelenő ablakban a metódus „Type” tulajdonságát állítsuk „Constructor” -ra.



- A módosítások után az alábbi modellt kapjuk:



Generáljunk ismét kódot Java nyelvre és vizsgáljuk meg a változtatásaink hatásait. A C# kód megvizsgálásához használjuk a módosítás utáni projektet (**CarRentalModositasUtan**). Szintaktikailag a

változtatásaink semmilyen újdonságot nem vezettek be, így az elemzést kezdhetjük rögtön a modell és kód kapcsolatának vizsgálatával, például vizsgáljuk meg a Car osztályt:

C#:

```
[objingid("49b62373-134d-483d-83f7-064a2c907118")]
public class Car
{
    [objingid("ab78b35d-5f26-4c12-89f1-c127cdd79adc")]
    public Car(string manufacturer, string plateNum, int chassisNum, CarType carType)
    {
    }

    #region borrower
    [objingid("b0f6fdad-a872-4c15-808c-084b87a768a8")]
    public Customer borrower = null;
    #endregion

    #region customer
    [objingid("766c0612-bfd3-47e0-89cf-5d21bf36ea54")]
    public Customer customer = null;
    #endregion

    #region manufacturer
    [objingid("4862f9e7-3c19-4393-bc95-3dad3835947f")]
    private string manufacturer;
    #endregion

    #region plateNum
    [objingid("ab14795f-5532-4394-ac7e-dca2c7d15e42")]
    private string plateNum;
    #endregion

    #region chassisNum
    [objingid("6c5755b6-08f1-4950-9c48-2874d9ae6def")]
    private int chassisNum;
    #endregion

    #region carType
    [objingid("1e489c12-551a-46b0-8ec9-06ca63a841e4")]
    private CarType carType;
    #endregion

    [objingid("3438b30b-80fa-4541-8ac2-3a2f143bf63d")]
    public string getManufacturer()
    {
        return this.manufacturer;
    }

    [objingid("385a9343-f66c-4073-8c9c-6aa629255e8a")]
    public string getPlateNum()
    {
        return this.plateNum;
    }

    [objingid("bd3ae419-361b-41aa-8fec-d82cf636e4fb")]
    public int getChassisNum()
```

```
        {
            return this.chassisNum;
        }

        [objingid("f2bd3559-4469-4172-b604-55781aa3b351")]
        public CarType getCarType()
        {
            return this.carType;
        }

    }
```

Java:

```
@objid ("49b62373-134d-483d-83f7-064a2c907118")
public class Car {
    @objid ("4862f9e7-3c19-4393-bc95-3dad3835947f")
    private String manufacturer;

    @objid ("ab14795f-5532-4394-ac7e-dca2c7d15e42")
    private String plateNum;

    @objid ("6c5755b6-08f1-4950-9c48-2874d9ae6def")
    private int chassisNum;

    @objid ("1e489c12-551a-46b0-8ec9-06ca63a841e4")
    private CarType carType;

    @objid ("b0f6fdad-a872-4c15-808c-084b87a768a8")
    public Customer borrower;

    @objid ("766c0612-bfd3-47e0-89cf-5d21bf36ea54")
    public Customer customer;

    @objid ("3438b30b-80fa-4541-8ac2-3a2f143bf63d")
    public String getManufacturer() {
        // Automatically generated method. Please delete this comment before entering
specific code.
        return this.manufacturer;
    }

    @objid ("385a9343-f66c-4073-8c9c-6aa629255e8a")
    public String getPlateNum() {
        // Automatically generated method. Please delete this comment before entering
specific code.
        return this.plateNum;
    }

    @objid ("bd3ae419-361b-41aa-8fec-d82cf636e4fb")
    public int getChassisNum() {
        // Automatically generated method. Please delete this comment before entering
specific code.
        return this.chassisNum;
    }

    @objid ("f2bd3559-4469-4172-b604-55781aa3b351")
```

```
    public CarType getCarType() {
        // Automatically generated method. Please delete this comment before entering
specific code.
        return this.carType;
    }

    @objid ("ab78b35d-5f26-4c12-89f1-c127cdd79adc")
    public Car(final String manufacturer, final String plateNum, final int
chassisNum, final CarType carType) {
    }
}
```

Az autó osztály esetén láthatjuk a modellen végrehajtott főbb változtatások hatását a kódra. Az új konstruktor hagyományos konstruktorra képződött. Továbbá az újonnan bevezetett getter metódusok megjelentek az osztályokban, mint publikus láthatóságú metódusok a privát adattagok lekérdezésére. A további módosításaink hasonló hatással voltak az ügyfél osztályra is.

Fontos megjegyezni, hogy mivel a C# támogatja a **property**-ket (tulajdonságokat), ezért a getterek és setterek helyett a generált kód sokkal „C#-osabb" lenne, ha a Modelio-s C# kódgenerátor támogatná a property generálást.

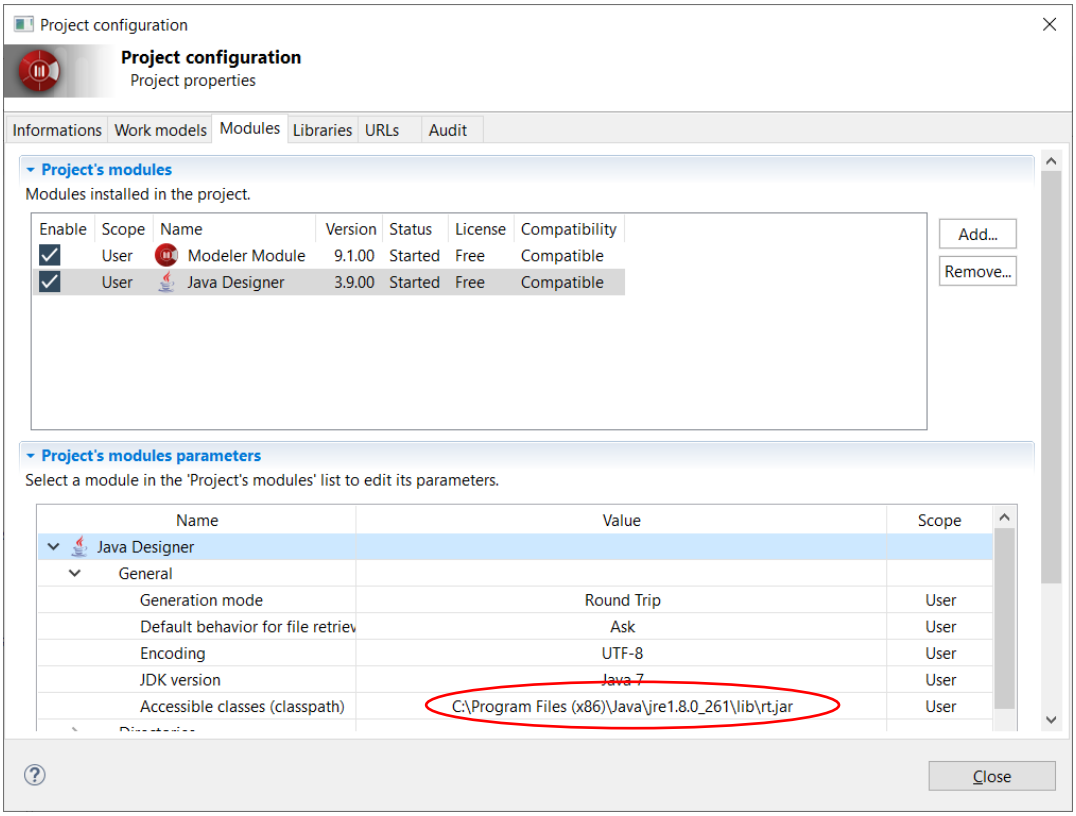
Webshop – Java forráskódból modell generálása (reverse engineering)

FELADAT:

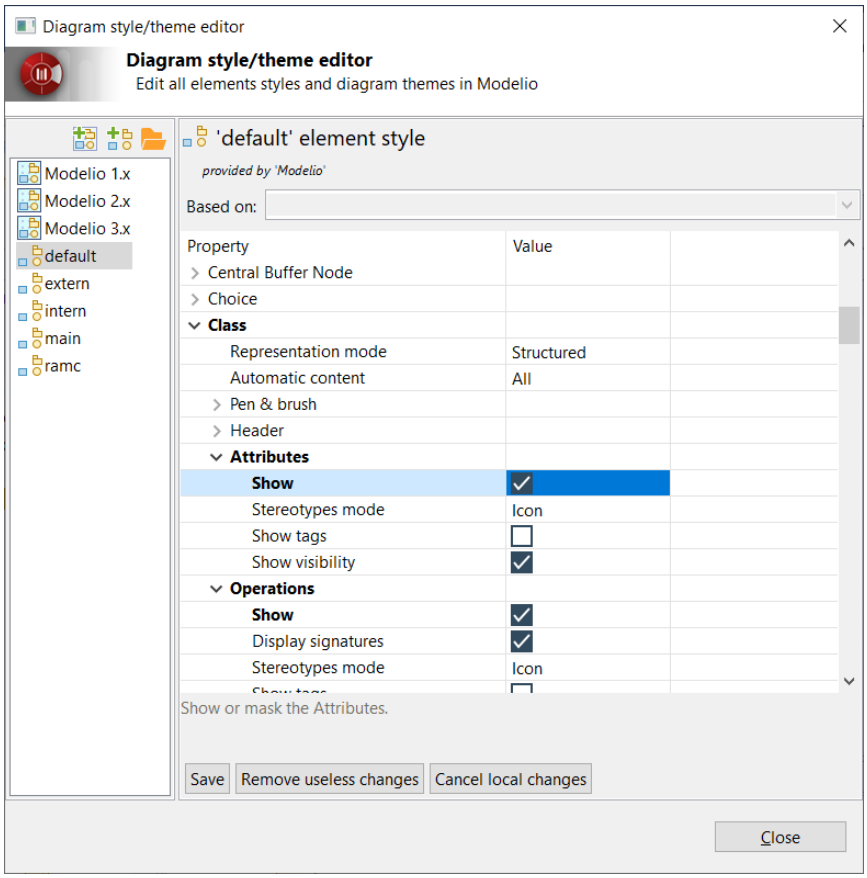
A feladat egy leegyszerűsített webshop elkészítése. Ennek első lépése, hogy a rendelkezésünkre álló Java skeleton kód alapján generáljuk le a bolt osztálydiagramját.

MEGOLDÁS:

- Először is importáljuk a Java forrásokat Eclipse-be! File > Import… > Existing projects into workspace > Select Root Directory, tallózuk be a Webshop mappáját > Finish
- Térjünk át a Modelio-ra és készítsünk egy új projektet!
- File > Create Project … > a projekt neve legyen *Webshop*
- **Pipáljuk be a Java project beállítást**!
- 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:



- A Webshop> Webshop > Webshop folderen 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 Webshop 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:

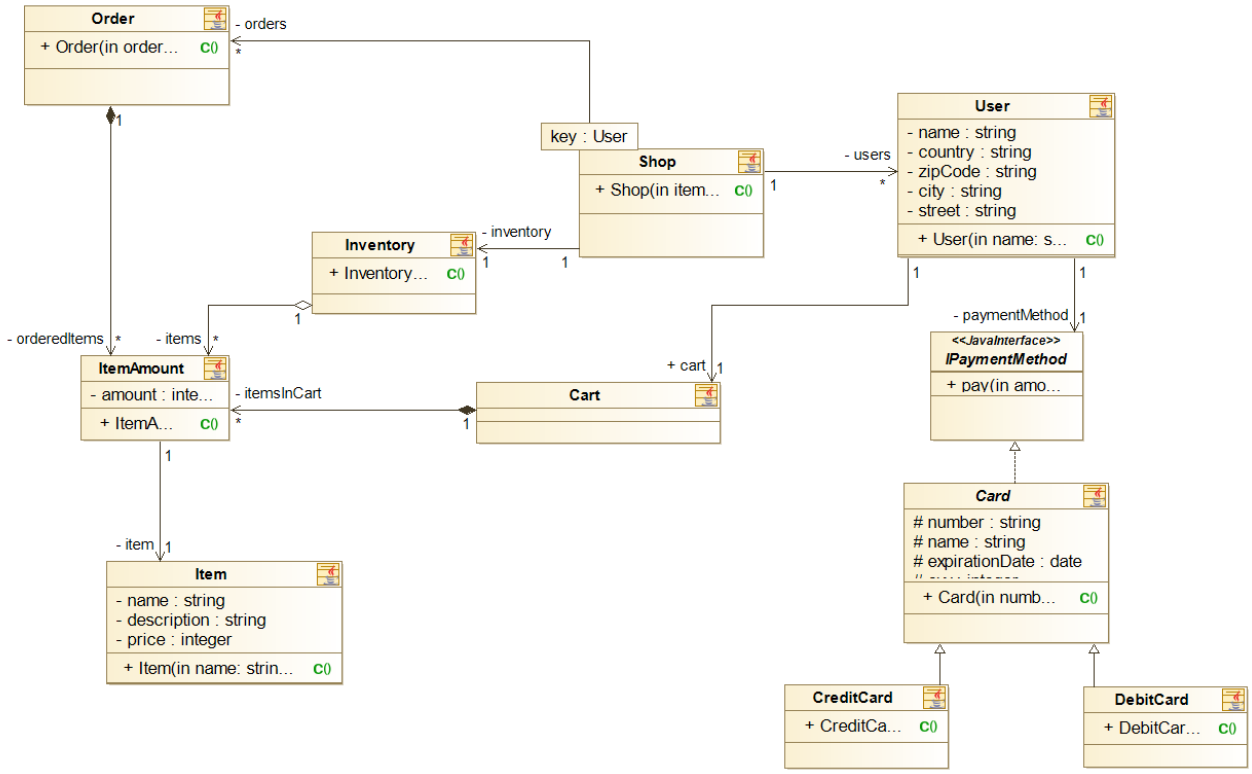


- **Javaslat2**: a műveleteket ne húzzuk be, mert azok úgyis csak getter/setterek. Szintén ne húzzuk rá a konstruktorokat, mert azok magukkal hozzák az összes műveletet. (Ctrl+Z-vel mindent visszavonhatunk, ha esetleg véletlenül bejöttek a getterek/setterek is.)
- Az egyes osztályok rádobása után mindig értelmezzük is az adott osztályt:
  - **Shop:** Láthatjuk, hogy három fő referenciát tartalmaz az osztály:
    - Az **inventory**-t, amely a megrendelhető termékekkel kapcsolatos adatokat és műveleteket tartalmazza.
    - A regisztrált felhasználók listáját (**users**).
    - A megrendelések map-jét (**orders**), ahol a kulcs az adott felhasználó, az érték pedig a felhasználó különböző rendelései.
  - **Inventory:** Egy **ItemAmount** listát tartalmaz, amely az egyes termékek raktáron levő mennyiségét tárolja.
  - **Item:** egy egyszerű adattárolóként szolgál, amely a termékekkel kapcsolatos információkat tárolja, mint név (**name**), leírás (**description**) és ár (**price**).
  - **User:** A felhasználók alapvető adatain felül (**name**, **country**, **zipCode**, **city**, **street**) az osztály tárolja a kosarat is, amelybe természetesen a vásárlás során helyezhetik el a termékeket a felhasználók, értelemszerűen akár többet is.
  - **IPaymentMethod:** az adott felhasználó fizetési módját is tároljuk, mely egyetlen metódus implementálását követeli meg, mégpedig a fizetését (**pay**). Ez a metódus átveszi a fizetendő összeget és visszaadja, hogy sikeres volt-e a művelet.
  - **CreditCard és DebitCard**:A webshop felhasználóinak lehetősége van kártyás fizetésre, amin belül hitelkártyát (**CreditCard**) vagy bankkártyát (**DebitCard**) használhatnak. Vegyük

észre, hogy a kártyás fizetés egy absztrakt ősosztály, amely megvalósítja az **IPaymentMethod** interfészt, de az implementálását a leszármazottjaira, azaz a bankkártyára és a hitelkártyára bízza. Ezzel az interfész-absztrakt ősosztály megközelítéssel egy könnyedén bővíthető fizetési rendszert hoztunk létre, amelyet így tetszőleges más fizetési móddal könnyedén kiegészíthetünk a meglévők módosítása nélkül.

- **Order:** Láthatjuk, hogy hasonlóan a kosárhoz, a rendelés is egy egyszerű listát tárol a megrendelt termékekről és azoknak a mennyiségéről.

- A map-et egy qualified association segítségével képezzük le két osztály között, melyet a Modelio-ban egy az asszociációra helyezett attribútum segítségével érhetünk el. Az attribútum típusa értelemszerűen a map kulcsának típusával egyezik meg, a map értékeinek típusa pedig az asszociáció másik vége által mutatott osztály. **Fontos!** Amikor behúzzuk az asszociációt a Shop és az Order között, a Modelio List[*] típusnak ismeri fel a cél osztályt. Kattintsunk az asszociációs nyílra és a Target tulajdonság To mezőjét állítsuk **Order**-re.
- A végeredmény hasonló lesz ehhez:



3. Önálló feladatok

Webshop – módosítás

Végezzünk el néhány módosítást a Java kódon majd vezessük vissza a változásokat a modellbe!

Első lépésben egészítsük ki a **Shop**-ot néhány hasznos metódussal. Adjuk hozzá a **bool registerUser(User user)** és **bool order(User user)** metódusokat, melyek a felhasználók regisztrálását és a rendelések véglegesítését fogják elvégezni.

Ezt követően egészítsük ki a **User** osztályt egy egyszerű **getCart()** getterrel, amely az adott felhasználó kosarát fogja visszaadni. Majd térjünk át a **Cart** osztályra és adjunk hozzá két új metódust:

1. **addToCart(Item item, int amount)**, mellyel a kosárhoz tudunk adott mennyiségű terméket hozzáadni.
2. **clear()**, mellyel a kosár tartalmát tudjuk teljesen kiüríteni.

Végezetül adjunk hozzá az **ItemAmount**-hoz egy **changeAmount(int amount)** metódust, amellyel az adott termék mennyiségét tudjuk ellenőrzött formában módosítani.

Vezessük vissza a módosításokat a modellbe! Ezt megtehetjük kézzel is, azonban a Modelio lehetőséget nyújt arra, hogy a forráskód alapján frissítse a modellt az új állapotra, használjuk a Java Designer -> Update model from sources if necessary lehetőséget. Sajnos előfordulat, hogy ez nem működik megfelelően. Ebben az esetben megoldás lehet, hogy a Java forrásokat újra beolvassuk a szokásos módon (Java Designer> Reverse >Reverse Java application from sources), és a bal oldali fából behúzzuk az új metódusokat.

Webshop – működő alkalmazás

A feladat a Webshop kiegészítése egy működő alkalmazássá. Néhány konstruktor kiegészítésre (adattagok inicicializálása) és a szöveges megjelenítéshez néhány metódusra van szükség.

A megoldásnak **mindenképpen** tartalmaznia kell az alábbi pontokat:

- A futtatáshoz kelleni fog egy **Main** metódus.
- A kipróbáláshoz elég, ha 2-3 termékkel inicializáljuk a webshopot, utána regisztrálunk egy felhasználót és rendelünk egy terméket (mindezt a **Main** metódusban néhány sorból meg lehet oldani).
- Inicializáció: **ItemAmount** lista készítése, majd új **Shop**, amely a konstruktorban paraméterül kapja ezt a listát.
- Regisztráció: új **User**, majd a **Shop**-on **registerUser**.
- Kosár kezelése: **User getCart** majd **addToCart** vagy esetleg **clear**
- Rendelés: **Shop order**

Szöveges kalandjáték (+ pontos feladat)

A feladat az alábbi specifikáció alapján elkészíteni a szoftver osztálydiagramját. Miután a modell elkészült, generáljunk belőle Java kódot és valósítsuk is meg a szükséges metódusokat, hogy a végén egy a specifikációnak megfelelő játék szülessen! Fontos: a játék elkészítéséhez nem szükséges kirajzolni semmit, tehát elegendő a megfelelő információkat szövegesen kiírni a képernyőre! A beadáshoz elegendő a forráskód és egy képernyőkép a diagramról!

Készítsünk egy szöveges kalandjátékot. A játék szobákból áll, ahol a játékosnak szörnyekkel kell megküzdenie. Minden szobában véletlenszerűen 1-5 szörny lehet. Miután a játékos legyőzte a szobában található szörnyeket, egy újabb szobába érkezik. A játékosnak van életereje és sebzése, amelyek minden teljesített szoba után egyel nőnek. A harc körökből áll. Minden körben 50% eséllyel a játékos kezd. Egy körben egyszer támadhat a játékos az egyik szörnyre, illetve egyszer támad véletlenszerűen az egyik szobában található szörny a játékosra. A játéknak akkor van vége, ha a játékosnak elfogy az élete vagy teljesített 10 szobát.

Tippek:

- Lehetséges osztályok: **Player**, **Room**, **Monster, Character ősosztály**
- A játék vezérléséhez esetleg egy **GameManager**-osztály is felhasználható
- A karakter tulajdonságai lehetnek például **health** és **damage**
- A körök szimulálásához elegendő egy egyszerű flag, hogy ki következik.