P6 - Turo

Final Report



Mestrado Integrado em Engenharia Informática e Computação

Sistemas de Informação

**Grupo:**

Diogo Reis - up201505472

Mariana Guimarães - up201307777

Tiago Magalhães - up201607931

Faculdade de Engenharia da Universidade do Porto  
Rua Roberto Frias, sn, 4200-465 Porto, Portugal

7 de Janeiro de 2019

Contents

[**System Description and Requirements** 4](#_Toc534584043)

[**UML Model** 5](#_Toc534584044)

[**Formal VDM++ Model** 8](#_Toc534584045)

[Bike Rack 8](#_Toc534584046)

[Bluetooth 8](#_Toc534584047)

[Convertible 8](#_Toc534584048)

[Custom Feature 9](#_Toc534584049)

[Heated Seats 9](#_Toc534584050)

[Pet Friendly 9](#_Toc534584051)

[4x4 10](#_Toc534584052)

[GPS 10](#_Toc534584053)

[USB 10](#_Toc534584054)

[Sunroof 11](#_Toc534584055)

[Snow Tires 11](#_Toc534584056)

[Ski Rack 11](#_Toc534584057)

[Feature 12](#_Toc534584058)

[Car 12](#_Toc534584059)

[Manufacturer 13](#_Toc534584060)

[Van 14](#_Toc534584061)

[Truck 14](#_Toc534584062)

[SUV 15](#_Toc534584063)

[Regular Car 16](#_Toc534584064)

[Mini Van 17](#_Toc534584065)

[Availability Calendar 18](#_Toc534584066)

[Date 18](#_Toc534584067)

[Date Factory 20](#_Toc534584068)

[Protection Plan 21](#_Toc534584069)

[Standard 21](#_Toc534584070)

[Basic 21](#_Toc534584071)

[Premium 21](#_Toc534584072)

[Commercial Plan 22](#_Toc534584073)

[Delivery Options 22](#_Toc534584074)

[Location 23](#_Toc534584075)

[Extra 23](#_Toc534584076)

[Listing 23](#_Toc534584077)

[Notification 25](#_Toc534584078)

[Instant Booking Notification 26](#_Toc534584079)

[Instant Booking Cancellation Notification 26](#_Toc534584080)

[Booking Request Notification 26](#_Toc534584081)

[Booking Request Cancelation Notification 27](#_Toc534584082)

[Inbox 27](#_Toc534584083)

[Booking 28](#_Toc534584084)

[Booking Request 29](#_Toc534584085)

[Review 31](#_Toc534584086)

[User 31](#_Toc534584087)

[Renter 32](#_Toc534584088)

[Lister 33](#_Toc534584089)

**HYPERLINK \l "\_Toc534584091" Model Verification** 35

HYPERLINK \l "\_Toc534584092" Code Generation 36

# System Description and Requirements

***Turo*** is a platform where you list cars for rent in a location and people who go on vacation or trips can rent them to use during the trip. As such our main requirement are to be able to login and register either as a renter or a lister. Renters Must be able to search and rent cars in a location and by the start and end date of his trip. Lister on the other hand must be able to list his cars for renting while the system must be responsible for calculating the price per day of renting a given car.

When a renter requests to rent a car the lister should receive a notification, he can then decide to accept or decline the booking request. If the request is accepted the request days must be removed from the set of days, the car is available to renting, and those days must be returned if the booking is cancelled.

As such our most important constraints are:

- There must not be two lister with the same username or two renters with the same username.

- The price per day of a car must not be zero.

- The system must always properly indicate whether a user is logged in or not and what type of user is logged in.

- Listings have extras, those extras must have a cost higher than 0.

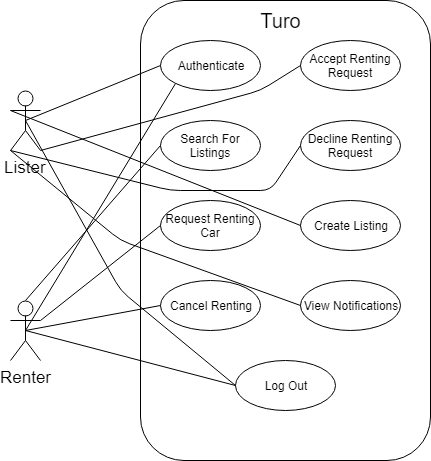
- Listings have dates, these days are made up for a day, a month and a year, the day must be between 1 and 30, the month must be between 1 and 12 and the year must be between larger than 0.

- When logging out you must first be logged in and after logging out the system must properly indicate you are logged out.

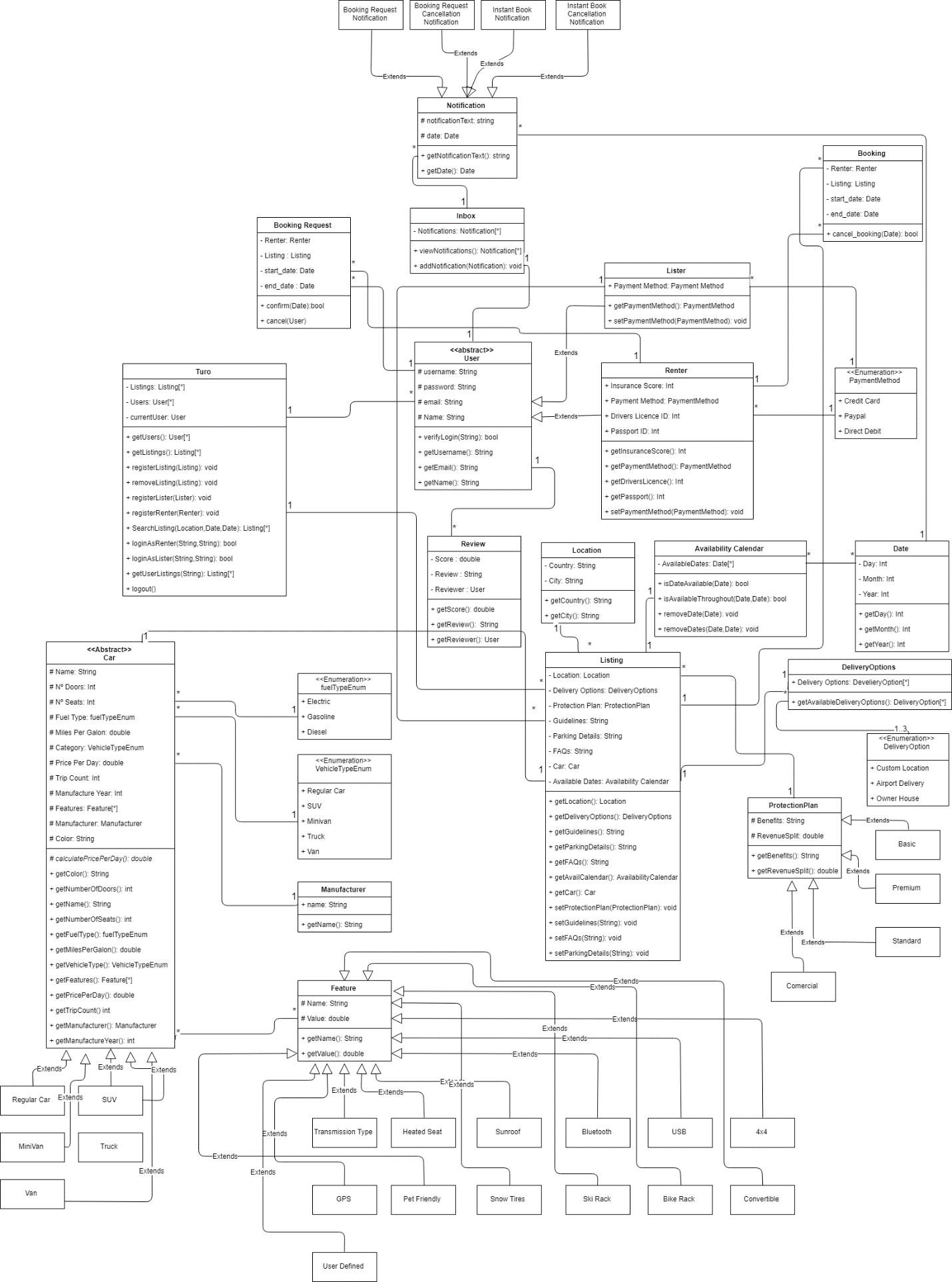
- When a lister attempts to confirm or decline a request for booking, that booking request must be active, meaning it has not been canceled and not been confirmed or declined yet.

|  |  |
| --- | --- |
| **Identifier** | **Requirement** |
| R01 | Users must be able to login either as a renter or a lister |
| R02 | Users must be able to search for trips by location and time frame |
| R03 | Renters must be able to send booking request for a car listing |
| R04 | Listers must be able to list their cars for renting |
| R05 | Lister should receive notifications and requests when someone attempts to book their car |
| R06 | There must be a logout feature |
| R07 | The lister must be able to accept or decline booking requests |
| R08 | Renters must be able to cancel bookings if it is not too late |
| R09 | The system must calculate the price per day of renting a car |

# UML Model



Here we can see the major use cases of the system as described earlier, the two main user types are the lister and the renter. The renter must be able to log into the system, search for listings, request car rentals and cancel those rentals and log out of the system. The lister must be able to log into the system, accept or decline rental requests, create rental listings, view and receive notifications regarding his listings and log out of the system.



Here we have the class diagram that represent the entire system, the represented classes are as follows:

* Car – this represents a car to be listed for renting, this is an abstract class.
* Regular Car, SUV, MiniVan, Truck and Van are classes that represent specific types of cars and descent from the car base class.
* Manufacturer – represent the manufacturer of a car.
* VehicleTypeEnum – is an enum class for the type of car.
* fuelTypeEnu, - is an enum class for the type of fuel the car uses.
* Feature – is an abstract class that represent a feature of a car, all descendent classes are specific features, here the idea is that we can very easily extend to system and add new features to it.
* User Defined – is a nonstandard feature class of the system and servers for the user to be able to give his car features that are no currently in the standard list of system features.
* Listing – this is a class that represent a listing for renting a car
* Delivery Options – This is a class that contains the options for delivering a car to the renter
* Location – This class represent a location where a listing is based on, this is a country and a city
* Availability Calendar – This class is a set of dates in which the car can be rented
* Protection Plan – this is an abstract class representing the protection plan the user has selected for the car
* Commercial, Basic, Standard and Premium – These are specific protection plans and are descendants of the Protection Plan Base class.
* User – This is an abstract class that represent a user independent of the type of user.
* Renter – This represents a user account that only servers for renting cars and is a descendent of the User base class.
* Lister – This represents a user account that only server for listing cars for renting and is a descendant class of the User base class.
* PaymentMethod – This is an enum class representing how a renter pays or how a lister gets payed
* Review - This is a class that represent a review of a user, this can be either a review of a lister or a renter.
* Booking – This class represent a booking that has been confirmed.
* Booking Request – This class represent a request from a renter to the lister of a car to rent the car.
* Inbox – This class represents an inbox of notifications of a user.
* Notification – This is a notification to a user based on his actions on the website and other users’ interactions with him, this is an abstract class and is not supposed to be initialized one of the specific notification classes should be used instead.
* Turo – This is the main class of the system, representing the system itself and is the primary interface the user has with the system.

# Formal VDM++ Model

## Bike Rack

**class** BikeRack **is subclass of** Feature

**values**

**private** NAME: string = "Bike Rack";

**private** DESCRIPTION: string = "This car has a bike rack";

**private** VALUE: **real** = 1.0;

**operations**

**public** BikeRack: () ==> BikeRack

BikeRack() ==

(

name := NAME;

description := DESCRIPTION;

value := VALUE;

return self;

);

**end** BikeRack

## Bluetooth

**class** Bluetooth **is subclass of** Feature

**values**

**private** NAME: string = "Bluetooth";

**private** DESCRIPTION: string = "This car supports bluetooth connectivity";

**private** VALUE: **real** = 1.0;

**operations**

**public** Bluetooth: () ==> Bluetooth

Bluetooth() ==

(

name := NAME;

description := DESCRIPTION;

value := VALUE;

return self

);

**end** Bluetooth

## Convertible

**class** Convertible **is subclass of** Feature

**values**

**private** NAME: string = "Convertible";

**private** DESCRIPTION: string = "This car is a convertible";

**private** VALUE: **real** = 1.0;

**operations**

**public** Convertible: () ==> Convertible

Convertible() ==

(

name := NAME;

description := DESCRIPTION;

value := VALUE;

return self;

);

**end** Convertible

## Custom Feature

**class** CustomFeature **is subclass of** Feature

**values**

**private** VALUE: **real** = 1.0;

**operations**

**public** CustomFeature: string \* string ==> CustomFeature

CustomFeature(m\_name,m\_desc) == (

name := m\_name;

description := m\_desc;

value := VALUE;

return self

);

**end** CustomFeature

## Heated Seats

**class** HeatedSeats **is subclass of** Feature

**values**

**private** NAME: string = "Heated Seat";

**private** DESCRIPTION: string = "This car has heated seats";

**private** VALUE: **real** = 1.0;

**operations**

**public** HeatedSeats: () ==> HeatedSeats

HeatedSeats() ==

(

name := NAME;

description := DESCRIPTION;

value := VALUE;

return self;

);

**end** HeatedSeats

## Pet Friendly

**class** PetFriendly **is subclass of** Feature

**values**

**private** NAME: string = "Pet Friendly";

**private** DESCRIPTION: string = "This car is pet friendly";

**private** VALUE: **real** = 1.0;

**operations**

**public** PetFriendly: () ==> PetFriendly

PetFriendly() ==

(

name := NAME;

description := DESCRIPTION;

value := VALUE;

return self;

);

**end** PetFriendly

## 4x4

**class** FourByFour **is subclass of** Feature

**values**

**private** NAME: string = "4x4";

**private** DESCRIPTION: string = "This car has 4 weel drive";

**private** VALUE: **real** = 1.0;

**operations**

**public** FourByFour: () ==> FourByFour

FourByFour() ==

(

name := NAME;

description := DESCRIPTION;

value := VALUE;

return self;

);

**end** FourByFour

## GPS

**class** GPS **is subclass of** Feature

**values**

**private** NAME: string = "GPS";

**private** DESCRIPTION: string = "This car has GPS connectivity";

**private** VALUE: **real** = 1.0;

**operations**

**public** GPS: () ==> GPS

GPS() ==

(

name := NAME;

description := DESCRIPTION;

value := VALUE;

return self;

);

**end** GPS

## USB

**class** USB **is subclass of** Feature

**values**

**private** NAME: string = "USB";

**private** DESCRIPTION: string = "This car has USB ports";

**private** VALUE: **real** = 1.0;

**operations**

**public** USB: () ==> USB

USB() ==

(

name := NAME;

description := DESCRIPTION;

value := VALUE;

return self

);

**end** USB

## Sunroof

**class** Sunroof **is subclass of** Feature

**values**

**private** NAME: string = "Sunroof";

**private** DESCRIPTION: string = "This car has a sunroof";

**private** VALUE: **real** = 1.0;

**operations**

**public** Sunroof: () ==> Sunroof

Sunroof() ==

(

name := NAME;

description := DESCRIPTION;

value := VALUE;

return self;

);

**end** Sunroof

## Snow Tires

**class** SnowTires **is subclass of** Feature

**values**

**private** NAME: string = "Snow Tires";

**private** DESCRIPTION: string = "This car has snow tires in case of need";

**private** VALUE: **real** = 1.0;

**operations**

**public** SnowTires: () ==> SnowTires

SnowTires() ==

(

name := NAME;

description := DESCRIPTION;

value := VALUE;

return self;

);

**end** SnowTires

## Ski Rack

**class** SkiRack **is subclass of** Feature

**values**

**private** NAME: string = "Ski Rack";

**private** DESCRIPTION: string = "This car has a ski rack";

**private** VALUE: **real** = 1.0;

**operations**

**public** SkiRack: () ==> SkiRack

SkiRack() ==

(

name := NAME;

description := DESCRIPTION;

value := VALUE;

return self;

);

**end** SkiRack

## Feature

**class** Feature

**types**

**public** string = **seq** **of** **char**;

**instance variables**

**protected** name: string := "";

**protected** description: string := "";

**protected** value: **real** := 0.0;

**operations**

**public** getName: () ==> string

getName() == (return self.name);

**public** getDescription: () ==> string

getDescription() == (return self.description);

**public** getValue: () ==> **real**

getValue() == (return self.value);

**end** Feature

## Car

**class** Car

**types**

**public** string = **seq** **of** **char**;

**public** fuelTypeEnum = **<ELECTRIC>** | **<GASOLINE>** | **<DIESEL>**;

**public** vehicleTypeEnum = **<REGULAR>** | **<SUV>** | **<MINIVAN>** | **<TRUCK>** | **<VAN>**;

**public** featureSet = **set** **of** Feature;

**instance variables**

**protected** name: string := "";

**protected** numberOfDoors: **nat** := 0;

**protected** numberOfSeats: **nat** := 0;

**protected** milesPerGalon: **real** := 0.0;

**protected** pricePerDay: **real** := 0.0;

**protected** tripCount: **nat** := 0;

**protected** manufactureYear: **nat** := 0;

**protected** color: string := "";

**protected** fuelType: fuelTypeEnum := <DIESEL>;

**protected** vehicleType: vehicleTypeEnum := <REGULAR>;

**protected** manufacturer: Manufacturer;

**protected** features: featureSet := {};

**operations**

**public** calculatePricePerDay: () ==> ()

calculatePricePerDay() == is **subclass responsibility**

**post** pricePerDay > 0.0;

**public** getName: () ==> string

getName() == (return self.name);

**public** getNumberOfDoors: () ==> **nat**

getNumberOfDoors() == (return self.numberOfDoors);

**public** getNumberOfSeats: () ==> **nat**

getNumberOfSeats() == (return self.numberOfSeats);

**public** getMilesPerGalon: () ==> **real**

getMilesPerGalon() == (return self.milesPerGalon);

**public** getPricePerDay: () ==> **real**

getPricePerDay() == (return self.pricePerDay);

**public** getTripCount: () ==> **nat**

getTripCount() == (return self.tripCount);

**public** getManufactureYear: () ==> **nat**

getManufactureYear() == (return self.manufactureYear);

**public** getColor: () ==> string

getColor() == (return self.color);

**public** getFuelType: () ==> fuelTypeEnum

getFuelType() == (return self.fuelType);

**public** getVehicleType: () ==> vehicleTypeEnum

getVehicleType() == (return self.vehicleType);

**public** getManufacturer: () ==> Manufacturer

getManufacturer() == (return self.manufacturer);

**public** getFeatures: () ==> featureSet

getFeatures() == (return self.features);

**public** addFeatureSet: (featureSet) ==> ()

addFeatureSet(m\_features) ==

(

features := features union m\_features;

);

**public** addFeature: (Feature) ==> ()

addFeature(m\_feature) ==

(

features := features union {m\_feature};

);

**end** Car

## Manufacturer

**class** Manufacturer

**types**

**public** string = **seq** **of** **char**;

**instance variables**

**private** name: string := "";

**operations**

-- constructor, takes in string representing the name of the manufacturer

**public** Manufacturer: string ==> Manufacturer

Manufacturer(m\_name) == (

name := m\_name;

return self

);

-- getter for the name

**public** getName: () ==> string

getName() == (return self.name);

**end** Manufacturer

## Van

**class** Van **is subclass of** Car

**values**

**private** NUMBER\_OF\_DOORS: **nat** = 2;

**private** NUMBER\_OF\_SEATS: **nat** = 2;

**private** BASE\_VALUE: **real** = 1.0;

**private** CAR\_TYPE: vehicleTypeEnum = <VAN>;

**operations**

**public** Van: string \* **real** \* **nat** \* **nat** \* string \* fuelTypeEnum \* Manufacturer \* featureSet ==> Van

Van(m\_name,m\_milespergalon,m\_trip\_count,m\_manufacture\_year,m\_color,m\_fuel\_type,m\_manufacturer,m\_features) ==

(

name := m\_name;

milesPerGalon := m\_milespergalon;

tripCount := m\_trip\_count;

manufactureYear := m\_manufacture\_year;

color := m\_color;

fuelType := m\_fuel\_type;

manufacturer := m\_manufacturer;

features := m\_features;

numberOfDoors := NUMBER\_OF\_DOORS;

numberOfSeats := NUMBER\_OF\_SEATS;

vehicleType := CAR\_TYPE;

self.calculatePricePerDay();

return self

);

**public** calculatePricePerDay: () ==> ()

calculatePricePerDay() ==

(

**dcl** dayPrice: **real** := BASE\_VALUE;

for **all** feature **in set** features **do**

(

dayPrice := dayPrice + feature.getValue();

);

pricePerDay := dayPrice;

)

**post** pricePerDay > 0.0;

**end** Van

## Truck

**class** Truck **is subclass of** Car

**values**

**private** NUMBER\_OF\_DOORS: **nat** = 2;

**private** NUMBER\_OF\_SEATS: **nat** = 2;

**private** BASE\_VALUE: **real** = 1.0;

**private** CAR\_TYPE: vehicleTypeEnum = <TRUCK>;

**operations**

**public** Truck: string \* **real** \* **nat** \* **nat** \* string \* fuelTypeEnum \* Manufacturer \* featureSet ==> Truck

Truck(m\_name,m\_milespergalon,m\_trip\_count,m\_manufacture\_year,m\_color,m\_fuel\_type,m\_manufacturer,m\_features) ==

(

name := m\_name;

milesPerGalon := m\_milespergalon;

tripCount := m\_trip\_count;

manufactureYear := m\_manufacture\_year;

color := m\_color;

fuelType := m\_fuel\_type;

manufacturer := m\_manufacturer;

features := m\_features;

numberOfDoors := NUMBER\_OF\_DOORS;

numberOfSeats := NUMBER\_OF\_SEATS;

vehicleType := CAR\_TYPE;

self.calculatePricePerDay();

return self

);

**public** calculatePricePerDay: () ==> ()

calculatePricePerDay() ==

(

**dcl** dayPrice: **real** := BASE\_VALUE;

for **all** feature **in set** features **do**

(

dayPrice := dayPrice + feature.getValue();

);

pricePerDay := dayPrice;

)

**post** pricePerDay > 0.0;

**end** Truck

## SUV

**class** SUV **is subclass of** Car

**values**

**private** NUMBER\_OF\_DOORS: **nat** = 4;

**private** NUMBER\_OF\_SEATS: **nat** = 5;

**private** BASE\_VALUE: **real** = 1.0;

**private** CAR\_TYPE: vehicleTypeEnum = <SUV>;

**operations**

**public** SUV: string \* **real** \* **nat** \* **nat** \* string \* fuelTypeEnum \* Manufacturer \* featureSet ==> SUV

SUV(m\_name,m\_milespergalon,m\_trip\_count,m\_manufacture\_year,m\_color,m\_fuel\_type,m\_manufacturer,m\_features) ==

(

name := m\_name;

milesPerGalon := m\_milespergalon;

tripCount := m\_trip\_count;

manufactureYear := m\_manufacture\_year;

color := m\_color;

fuelType := m\_fuel\_type;

manufacturer := m\_manufacturer;

features := m\_features;

numberOfDoors := NUMBER\_OF\_DOORS;

numberOfSeats := NUMBER\_OF\_SEATS;

vehicleType := CAR\_TYPE;

self.calculatePricePerDay();

return self

);

**public** calculatePricePerDay: () ==> ()

calculatePricePerDay() ==

(

**dcl** dayPrice: **real** := BASE\_VALUE;

for **all** feature **in set** features **do**

(

dayPrice := dayPrice + feature.getValue();

);

pricePerDay := dayPrice;

)

**post** pricePerDay > 0.0;

**end** SUV

## Regular Car

**class** RegularCar **is subclass of** Car

**values**

**private** NUMBER\_OF\_DOORS: **nat** = 4;

**private** NUMBER\_OF\_SEATS: **nat** = 5;

**private** BASE\_VALUE: **real** = 1.0;

**private** CAR\_TYPE: vehicleTypeEnum = <REGULAR>;

**operations**

**public** RegularCar: string \* **real** \* **nat** \* **nat** \* string \* fuelTypeEnum \* Manufacturer \* featureSet ==> RegularCar

RegularCar(m\_name,m\_milespergalon,m\_trip\_count,m\_manufacture\_year,m\_color,m\_fuel\_type,m\_manufacturer,m\_features) ==

(

name := m\_name;

milesPerGalon := m\_milespergalon;

tripCount := m\_trip\_count;

manufactureYear := m\_manufacture\_year;

color := m\_color;

fuelType := m\_fuel\_type;

manufacturer := m\_manufacturer;

features := m\_features;

numberOfDoors := NUMBER\_OF\_DOORS;

numberOfSeats := NUMBER\_OF\_SEATS;

vehicleType := CAR\_TYPE;

self.calculatePricePerDay();

return self

);

**public** calculatePricePerDay: () ==> ()

calculatePricePerDay() ==

(

**dcl** dayPrice: **real** := BASE\_VALUE;

for **all** feature **in set** features **do**

(

dayPrice := dayPrice + feature.getValue();

);

pricePerDay := dayPrice;

)

**post** pricePerDay > 0.0;

**end** RegularCar

## Mini Van

**class** MiniVan **is subclass of** Car

**values**

**private** NUMBER\_OF\_DOORS: **nat** = 4;

**private** NUMBER\_OF\_SEATS: **nat** = 5;

**private** BASE\_VALUE: **real** = 1.0;

**private** CAR\_TYPE: vehicleTypeEnum = <MINIVAN>;

**operations**

**public** MiniVan: string \* **real** \* **nat** \* **nat** \* string \* fuelTypeEnum \* Manufacturer \* featureSet ==> MiniVan

MiniVan(m\_name,m\_milespergalon,m\_trip\_count,m\_manufacture\_year,m\_color,m\_fuel\_type,m\_manufacturer,m\_features) ==

(

name := m\_name;

milesPerGalon := m\_milespergalon;

tripCount := m\_trip\_count;

manufactureYear := m\_manufacture\_year;

color := m\_color;

fuelType := m\_fuel\_type;

manufacturer := m\_manufacturer;

features := m\_features;

numberOfDoors := NUMBER\_OF\_DOORS;

numberOfSeats := NUMBER\_OF\_SEATS;

vehicleType := CAR\_TYPE;

self.calculatePricePerDay();

return self

);

**public** calculatePricePerDay: () ==> ()

calculatePricePerDay() ==

(

**dcl** dayPrice: **real** := BASE\_VALUE;

for **all** feature **in set** features **do**

(

dayPrice := dayPrice + feature.getValue();

);

pricePerDay := dayPrice;

)

**post** pricePerDay > 0.0;

**end** MiniVan

## Availability Calendar

**class** AvailabilityCalendar

**types**

**public** dates = **set** **of** Date;

**instance variables**

**private** availableDates: dates := {};

**operations**

**public** AvailabilityCalendar: () ==> AvailabilityCalendar

AvailabilityCalendar() == (return self);

**public** AvailabilityCalendar: dates ==> AvailabilityCalendar

AvailabilityCalendar(t\_dates) ==

(

availableDates := t\_dates;

return self

);

**public** AvailabilityCalendar: Date ==> AvailabilityCalendar

AvailabilityCalendar(t\_date) ==

(

availableDates := {t\_date};

return self

);

**public** getDates: () ==> dates

getDates() == (return self.availableDates);

**public** isDateAvailable: Date ==> **bool**

isDateAvailable(t\_date) == (return t\_date in **set** availableDates);

**public** areDatesAvailable: dates ==> **bool**

areDatesAvailable(t\_dates) == (return t\_dates subset availableDates);

**public** removeDate: Date ==> ()

removeDate(t\_date) == (availableDates := availableDates \ {t\_date});

**public** removeDates: dates ==> ()

removeDates(t\_dates) == (availableDates := availableDates \ t\_dates);

**public** addDates: dates ==> ()

addDates(t\_dates) == (availableDates := availableDates union t\_dates);

**public** availableThrough : Date \* Date ==> **bool**

availableThrough(t\_start\_date,t\_end\_date) ==

(

**dcl** wanted\_dates: dates := t\_start\_date.getDatesTo(t\_end\_date);

return wanted\_dates subset availableDates;

);

**end** AvailabilityCalendar

## Date

**class** Date

**types**

**public** string = **seq** **of** **char**;

**values**

**private** days\_in\_month: **nat** = 30;

**private** months\_in\_year: **nat** = 12;

**private** day\_in\_year: **nat** = days\_in\_month \* months\_in\_year;

**instance variables**

**private** year:**nat** := 0;

**private** month:**nat** := 0;

**private** day:**nat** := 0;

**operations**

**protected** Date: **nat** \* **nat** \* **nat** ==> Date

Date(t\_day,t\_month,t\_year) ==

(

year := t\_year;

month := t\_month;

day := t\_day;

return self

)

**pre** t\_day > 0 and t\_day < 31

and t\_month > 0 and t\_month < 13

and t\_year > 0;

**public** getDay: () ==> **nat**

getDay() == (return self.day);

**public** getMonth: () ==> **nat**

getMonth() == (return self.month);

**public** getYear: () ==> **nat**

getYear() == (return self.year);

**public** compare: Date ==> **bool**

compare(d2) == (return (d2.day = self.day and d2.month = self.month and d2.year = self.year));

**public** getText: () ==> string

getText() ==

(

**dcl** ret: string := "";

return ret

);

**public** daysSinceStart: () ==> **nat**

daysSinceStart() ==

(

return self.day + self.month \* days\_in\_month + self.year \* day\_in\_year;

);

**public** daysToDate: Date ==> **nat**

daysToDate(t\_date) ==

(

return t\_date.daysSinceStart() - self.daysSinceStart();

);

**public** getNextDay: () ==> Date

getNextDay() ==

(

**dcl** n\_day: **nat** := self.day + 1;

**dcl** n\_month: **nat** := self.month;

**dcl** n\_year: **nat** := self.year;

**dcl** fac: DateFactory := new DateFactory();

if(n\_day > days\_in\_month) **then**

(

n\_day := 1;

n\_month := n\_month + 1;

);

if(n\_month > months\_in\_year) **then**

(

n\_month := 1;

n\_year := n\_year + 1;

);

return fac.create\_date(n\_day,n\_month,n\_year)

);

**public** getDatesTo: Date ==> **set** **of** Date

getDatesTo(t\_end\_date) ==

(

**dcl** curr\_date: Date := self;

**dcl** wanted\_dates: **set** **of** Date := {self};

**dcl** date\_diff : **nat** := self.daysToDate(t\_end\_date);

for i=2 **to** date\_diff **by** 1 **do**

(

curr\_date := curr\_date.getNextDay();

wanted\_dates := wanted\_dates union {curr\_date};

);

return wanted\_dates

);

**end** Date

## Date Factory

**class** DateFactory **is subclass of** Date

**types**

**private** dateTuple = **nat** \* **nat** \* **nat**;

**private** datemap = **map** dateTuple **to** Date;

**instance variables**

**private** **static** dates: datemap := {|->};

**operations**

**public** **static** create\_date: **nat** \* **nat** \* **nat** ==> Date

create\_date(t\_day,t\_month,t\_year) ==

(

if({mk\_(t\_day,t\_month,t\_year)} subset dom dates) **then**

(

return dates(mk\_(t\_day,t\_month,t\_year));

)**else**

(

**dcl** dt: Date := new Date(t\_day,t\_month,t\_year);

dates := dates munion {mk\_(t\_day,t\_month,t\_year) |-> dt};

return dates(mk\_(t\_day,t\_month,t\_year));

);

)

**pre** t\_day > 0 and t\_day < 31

and t\_month > 0 and t\_month < 13

and t\_year > 0;

**end** DateFactory

## Protection Plan

**class** ProtectionPlan

**types**

**public** string = **seq** **of** **char**;

**instance variables**

**protected** benefits: string := "";

**protected** revenueSplit: **real** := 0.0;

**operations**

**public** getBenefits: () ==> string

getBenefits() == (return self.benefits);

**public** getRevenueSplit: () ==> **real**

getRevenueSplit() == (return self.revenueSplit);

**end** ProtectionPlan

## Standard

**class** Standard **is subclass of** ProtectionPlan

**values**

**private** BENEFITS: string = "";

**private** REVENUE\_SPLIT: **real** = 0.8;

**operations**

**public** Standard: () ==> Standard

Standard() ==

(

benefits := BENEFITS;

revenueSplit := REVENUE\_SPLIT;

return self;

)

**end** Standard

## Basic

**class** Basic **is subclass of** ProtectionPlan

**values**

**private** BENEFITS: string = "";

**private** REVENUE\_SPLIT: **real** = 0.85;

**operations**

**public** Basic: () ==> Basic

Basic() ==

(

benefits := BENEFITS;

revenueSplit := REVENUE\_SPLIT;

return self;

)

**end** Basic

## Premium

**class** Premium **is subclass of** ProtectionPlan

**values**

**private** BENEFITS: string = "";

**private** REVENUE\_SPLIT: **real** = 0.8;

**operations**

**public** Premium: () ==> Premium

Premium() ==

(

benefits := BENEFITS;

revenueSplit := REVENUE\_SPLIT;

return self;

)

**end** Premium

## Commercial Plan

**class** ComercialPlan **is subclass of** ProtectionPlan

**values**

**private** BENEFITS: string = "";

**private** REVENUE\_SPLIT: **real** = 0.9;

**operations**

**public** ComercialPlan: () ==> ComercialPlan

ComercialPlan() ==

(

benefits := BENEFITS;

revenueSplit := REVENUE\_SPLIT;

return self;

)

**end** ComercialPlan

## Delivery Options

**class** DeliveryOptions

**types**

**public** develiveryOption = **<CUSTOM\_LOCATION>** | **<AIRPORT>** | **<OWNER\_HOUSE>**;

**instance variables**

**public** deliveryOptions: **set** **of** develiveryOption := {};

**operations**

**public** develiveryOptions: () ==> DeliveryOptions

develiveryOptions() == (return self);

**public** develiveryOptions: **set** **of** develiveryOption ==> DeliveryOptions

develiveryOptions(t\_options) ==

(

deliveryOptions := t\_options;

return self

);

**public** develiveryOptions: develiveryOption ==> DeliveryOptions

develiveryOptions(t\_option) ==

(

deliveryOptions := {t\_option};

return self

);

**public** getDeliveryOptions: () ==> **set** **of** develiveryOption

getDeliveryOptions() == (return self.deliveryOptions);

**end** DeliveryOptions

## Location

**class** Location

**types**

**public** string = **seq** **of** **char**;

**instance variables**

**private** country: string := "";

**private** city: string := "";

**operations**

**public** Location: string \* string ==> Location

Location(t\_country,t\_city) ==

(

country := t\_country;

city := t\_city;

return self

);

**public** getCountry: () ==> string

getCountry() == (return self.country);

**public** getCity: () ==> string

getCity() == (return self.city);

**end** Location

## Extra

**class** Extra

**types**

**public** string = **seq** **of** **char**;

**instance variables**

**private** name: string := "";

**private** description: string := "";

**private** cost: **real** := 0.0;

**operations**

**public** Extra: string \* string \* **real** ==> Extra

Extra(t\_name,t\_description,t\_value) ==

(

name := t\_name;

description := t\_description;

cost := t\_value;

return self;

)

**pre** t\_value > 0.0;

**public** getName: () ==> string

getName() == (return self.name);

**public** getDescription: () ==> string

getDescription() == (return self.description);

**public** getCost: () ==> **real**

getCost() == (return self.cost);

**end** Extra

## Listing

**class** Listing

**types**

**public** string = **seq** **of** **char**;

**public** extraSet = **set** **of** Extra;

**instance variables**

**private** location: Location;

**private** develiveryOptions: DeliveryOptions;

**private** protectionPlan: ProtectionPlan;

**private** guidelines: string := "";

**private** parkingDetails: string := "";

**private** faqs: string := "";

**private** car: Car;

**private** availableDates: AvailabilityCalendar;

**private** hasInstantBook: **bool** := false;

**private** Lister: Lister;

**private** extras: extraSet := {};

**operations**

**public** Listing: Location \* DeliveryOptions \* ProtectionPlan \* string \* string \* string \* Car \* AvailabilityCalendar \* **bool** \* Lister \* extraSet==> Listing

Listing(t\_location,t\_options,t\_plan,t\_guidelines,t\_park\_details,t\_faqs,t\_car,t\_dates,t\_instant\_book, t\_owner,t\_extras) ==

(

location := t\_location;

develiveryOptions := t\_options;

protectionPlan := t\_plan;

guidelines := t\_guidelines;

parkingDetails := t\_park\_details;

faqs := t\_faqs;

car := t\_car;

availableDates := t\_dates;

hasInstantBook := t\_instant\_book;

Lister := t\_owner;

extras:= t\_extras;

return self;

);

**public** getLocation: () ==> Location

getLocation() == (return self.location);

**public** getDeliveryOptions: () ==> DeliveryOptions

getDeliveryOptions() == (return self.develiveryOptions);

**public** getProtectionPlan: () ==> ProtectionPlan

getProtectionPlan() == (return self.protectionPlan);

**public** getGuidelines: () ==> string

getGuidelines() == (return self.guidelines);

**public** getParkingDetails: () ==> string

getParkingDetails() == (return self.parkingDetails);

**public** getFAQS: () ==> string

getFAQS() == (return self.faqs);

**public** getCar: () ==> Car

getCar() == (return self.car);

**public** getAvailableDates: () ==> AvailabilityCalendar

getAvailableDates() == (return self.availableDates);

**public** getLister: () ==> Lister

getLister() == (return self.Lister);

**public** getExtras: () ==> extraSet

getExtras() == (return self.extras);

**public** requestBooking: Renter \* Date \* Date \* Date \* extraSet ==> **bool**

requestBooking(t\_booker,t\_curr\_date,t\_start\_date,t\_end\_date,t\_extras) ==

(

**dcl** request: BookingRequest := new BookingRequest(t\_booker,self,t\_start\_date,t\_end\_date,t\_extras);

**dcl** wanted\_dates: **set** **of** Date := t\_start\_date.getDatesTo(t\_end\_date);

**dcl** notification: Notification := new BookingRequestNotification(t\_curr\_date,t\_booker,self.car,wanted\_dates,t\_extras,request);

**dcl** inbox: Inbox := self.Lister.getInbox();

if(self.availableDates.availableThrough(t\_start\_date,t\_end\_date)) **then**

(

inbox.registerNotification(notification);

self.Lister.addRequest(request);

t\_booker.addRequest(request);

return true;

)**else**

(

return false;

);

);

**public** instantBook: Renter \* Date \* Date \* Date \* extraSet==> **bool**

instantBook(t\_booker,t\_curr\_date,t\_start\_date,t\_end\_date,t\_extras) ==

(

**dcl** booking: Booking := new Booking(t\_booker,self,t\_start\_date,t\_end\_date,t\_extras);

**dcl** wanted\_dates: **set** **of** Date := t\_start\_date.getDatesTo(t\_end\_date);

**dcl** notification: Notification := new InstantBookNotification(t\_curr\_date,t\_booker,self.car,wanted\_dates,t\_extras);

**dcl** inbox: Inbox := self.Lister.getInbox();

if(self.availableDates.availableThrough(t\_start\_date,t\_end\_date)) **then**

(

inbox.registerNotification(notification);

availableDates.removeDates(wanted\_dates);

t\_booker.addBooking(booking);

return true;

)**else**

(

return false;

);

)

**pre** hasInstantBook = true;

## **end** Listing

## Notification

**class** Notification

**types**

**public** string = **seq** **of** **char**;

**instance variables**

**protected** notificationText: string := "";

**protected** date: Date;

**protected** viewed: **bool** := false;

**operations**

**public** getNotificationText: () ==> string

getNotificationText() == (return self.notificationText);

**public** getNotificationDate: () ==> Date

getNotificationDate() == (return self.date);

**public** wasViewed: () ==> **bool**

wasViewed() == (return self.viewed);

**public** setAsViewed: () ==> ()

setAsViewed() == (viewed := true);

**end** Notification

## Instant Booking Notification

**class** InstantBookNotification **is subclass of** Notification

**types**

**public** extraSet = **set** **of** Extra;

**values**

**private** base\_string\_1: string = " has instantly booked the car ";

**private** base\_string\_2: string = " on the days:";

**instance variables**

**private** extras: extraSet := {};

**operations**

**public** InstantBookNotification: Date \* User \* Car \* **set** **of** Date \* extraSet==> InstantBookNotification

InstantBookNotification(t\_date,t\_user,t\_car,t\_dates,t\_extras) ==

(

notificationText := ":" ^ t\_user.getName() ^ base\_string\_1 ^ t\_car.getName() ^ base\_string\_2;

date := t\_date;

return self;

)

**end** InstantBookNotification

## Instant Booking Cancellation Notification

**class** InstantBookCancellationNotification **is subclass of** Notification

**values**

**private** base\_string\_1: string = " has cancelled his instant book of the car ";

**operations**

**public** InstantBookCancellationNotification: Date \* User \* Car ==> InstantBookCancellationNotification

InstantBookCancellationNotification(t\_date,t\_user,t\_car) ==

(

notificationText := ":" ^ t\_user.getName() ^ base\_string\_1 ^ t\_car.getName();

date := t\_date;

return self;

)

**end** InstantBookCancellationNotification

## Booking Request Notification

**class** BookingRequestNotification **is subclass of** Notification

**types**

**public** extraSet = **set** **of** Extra;

**values**

**private** base\_string\_1: string = " has request to book the car ";

**private** base\_string\_2: string = " on the following days:";

**instance variables**

**private** extras: extraSet := {};

**private** request: BookingRequest;

**operations**

**public** BookingRequestNotification: Date \* User \* Car \* **set** **of** Date \* extraSet \* BookingRequest==> BookingRequestNotification

BookingRequestNotification(t\_date,t\_user,t\_car,t\_dates,t\_extras, t\_request) ==

(

notificationText := ":" ^ t\_user.getName() ^ base\_string\_1 ^ t\_car.getName() ^ base\_string\_2;

extras := t\_extras;

request := t\_request;

date := t\_date;

return self;

);

**public** getRequest: () ==> BookingRequest

getRequest() == (return self.request);

**end** BookingRequestNotification

## Booking Request Cancelation Notification

**class** BookingRequestCancellationNotification **is subclass of** Notification

**values**

**private** base\_string\_1: string = " has cancelled his request to book the car ";

**operations**

**public** BookingRequestCancellationNotification: Date \* User \* Car ==> BookingRequestCancellationNotification

BookingRequestCancellationNotification(t\_date,t\_user,t\_car) ==

(

notificationText := ":" ^ t\_user.getName() ^ base\_string\_1 ^ t\_car.getName();

date := t\_date;

return self;

)

**end** BookingRequestCancellationNotification

## Inbox

**class** Inbox

**types**

**public** notificationSet = **set** **of** Notification;

**instance variables**

**private** notifications: notificationSet := {}

**operations**

**public** getAllNotifications: () ==> notificationSet

getAllNotifications() == (return self.notifications);

**public** getNotViewedNotifications: () ==> notificationSet

getNotViewedNotifications() ==

(

**dcl** retNotifications: notificationSet := {};

for **all** notification **in set** retNotifications **do**

(

if(notification.wasViewed() = false) **then**

(

retNotifications := retNotifications union {notification};

);

);

return retNotifications

);

**public** viewNotifications: () ==> notificationSet

viewNotifications() ==

(

for **all** notification **in set** notifications **do**

(

notification.setAsViewed();

);

return self.notifications;

);

**public** registerNotification: Notification ==> ()

registerNotification(t\_notification) ==

(

notifications := notifications union {t\_notification};

);

**end** Inbox

## Booking

**class** Booking

**types**

**public** extraSet = **set** **of** Extra;

**instance variables**

**private** renter: Renter;

**private** listing: Listing;

**private** start\_date: Date;

**private** end\_date: Date;

**private** active: **bool** := true;

**private** extras: extraSet := {};

**operations**

**public** Booking: Renter \* Listing \* Date \* Date \* extraSet==> Booking

Booking(t\_renter,t\_listing,t\_start\_date,t\_end\_date,t\_extras) ==

(

renter := t\_renter;

listing := t\_listing;

start\_date := t\_start\_date;

end\_date := t\_end\_date;

extras := t\_extras;

return self;

);

**public** cancel: User \* Date ==> **bool**

cancel(t\_user,t\_curr\_date) ==

(

**dcl** wanted\_dates: **set** **of** Date := start\_date.getDatesTo(end\_date);

if(t\_user.getUsername() = renter.getUsername() and t\_curr\_date.daysSinceStart() < start\_date.daysSinceStart() and active) **then**

(

listing.getAvailableDates().addDates(wanted\_dates);

active := false;

return true;

)**else**

(

return false;

);

);

**public** getRenter: () ==> Renter

getRenter() == (return self.renter);

**public** getListing: () ==> Listing

getListing() == (return self.listing);

**public** getStartDate: () ==> Date

getStartDate() == (return self.start\_date);

**public** getEndDate: () ==> Date

getEndDate() == (return self.end\_date);

**public** isActive: () ==> **bool**

isActive() == (return self.active);

**public** getTotalPrice: () ==> **real**

getTotalPrice() ==

(

**dcl** total: **real** := 0.0;

total := total + listing.getCar().getPricePerDay();

for **all** extra **in set** extras **do**

(

total := total + extra.getCost();

);

return total;

);

**end** Booking

## Booking Request

**class** BookingRequest

**types**

**public** extraSet = **set** **of** Extra;

**instance variables**

**private** renter: Renter;

**private** listing: Listing;

**private** start\_date: Date;

**private** end\_date: Date;

**private** active: **bool** := true;

**private** extras: extraSet := {};

**operations**

**public** BookingRequest: Renter \* Listing \* Date \* Date \* extraSet==> BookingRequest

BookingRequest(t\_renter,t\_listing,t\_start\_date,t\_end\_date,t\_extras) ==

(

renter := t\_renter;

listing := t\_listing;

start\_date := t\_start\_date;

end\_date := t\_end\_date;

extras := t\_extras;

return self;

);

**public** confirm: User \* Date ==> **bool**

confirm(t\_user,t\_curr\_date)==

(

**dcl** wanted\_dates: **set** **of** Date := start\_date.getDatesTo(end\_date);

if(t\_user.getUsername() = listing.getLister().getUsername() and t\_curr\_date.daysSinceStart() < start\_date.daysSinceStart() and active) **then**

(

-- send notification

listing.getAvailableDates().removeDates(wanted\_dates);

active := false;

renter.addBooking(new Booking(renter,listing,start\_date,end\_date,extras));

return true;

)**else**

(

return false;

);

)

**pre** active = true;

**public** decline: User \* Date ==> **bool**

decline(t\_user,t\_curr\_date)==

(

if(t\_user.getUsername() = listing.getLister().getUsername() and t\_curr\_date.daysSinceStart() < start\_date.daysSinceStart() and active) **then**

(

-- send notification

active := false;

return true;

)**else**

(

return false;

);

)

**pre** active = true;

**public** cancel: User \* Date ==> **bool**

cancel(t\_user,t\_curr\_date)==

(

if(t\_user.getUsername() = renter.getUsername() and t\_curr\_date.daysSinceStart() < start\_date.daysSinceStart() and active) **then**

(

-- send notificaion

active := false;

return true;

)**else**

(

return false;

);

)

**pre** active = true;

**public** getRenter: () ==> Renter

getRenter() == (return self.renter);

**public** getListing: () ==> Listing

getListing() == (return self.listing);

**public** getStartDate: () ==> Date

getStartDate() == (return self.start\_date);

**public** getEndDate: () ==> Date

getEndDate() == (return self.end\_date);

**public** isActive: () ==> **bool**

isActive() == (return self.active);

**public** getTotalPrice: () ==> **real**

getTotalPrice() ==

(

**dcl** total: **real** := 0.0;

total := total + listing.getCar().getPricePerDay();

for **all** extra **in set** extras **do**

(

total := total + extra.getCost();

);

return total;

);

**end** BookingRequest

## Review

**class** Review

**types**

**public** string = **seq** **of** **char**;

**instance variables**

**private** reviewer: User;

**private** reviewScore: **real** := 0.0;

**private** reviewDescription: string := "";

**operations**

**public** Review : User \* **real** \* string ==> Review

Review(t\_reviewer,t\_score,t\_review) ==

(

reviewer := t\_reviewer;

reviewScore := t\_score;

reviewDescription := t\_review;

return self;

)

**pre** t\_score >= 0.0 and t\_score <= 5.0;

**public** getReviewer: () ==> User

getReviewer() == (return self.reviewer);

**public** getReviewScore: () ==> **real**

getReviewScore() == (return self.reviewScore);

**public** getReview: () ==> string

getReview() == (return self.reviewDescription);

**end** Review

## User

**class** User

**types**

**public** string = **seq** **of** **char**;

**public** paymentMethod = **<DEBIT>** | **<CREDIT>** | **<PAYPAL>**;

**public** reviewSet = **set** **of** Review;

**instance variables**

**protected** username: string := "";

**protected** password: string := "";

**protected** email: string := "";

**protected** name: string := "";

**private** notifications: Inbox := new Inbox();

**private** requests: **set** **of** BookingRequest := {};

**private** reviews: reviewSet := {};

**operations**

**public** **pure** getUsername: () ==> string

getUsername() == (return self.username);

**public** getEmail: () ==> string

getEmail() == (return self.email);

**public** getName: () ==> string

getName() == (return self.name);

**public** verifyLogin: string ==> **bool**

verifyLogin(pass) == (return pass = self.password);

**public** getInbox: () ==> Inbox

getInbox() == (return self.notifications);

**public** getRequests: () ==> **set** **of** BookingRequest

getRequests() == (return self.requests);

**public** addRequest: BookingRequest ==> ()

addRequest(t\_request) == (requests := requests union {t\_request});

**public** addReview: Review ==> ()

addReview(t\_review) == (reviews := reviews union {t\_review});

**public** getReviews: () ==> reviewSet

getReviews() == (return self.reviews);

**end** User

## Renter

**class** Renter **is subclass of** User

**instance variables**

**private** insuranceScore: **nat** := 0;

**private** driversLicenceID: **nat** := 0;

**private** passportId: **nat** := 0;

**private** PaymentMethod: paymentMethod;

**private** bookings: **set** **of** Booking := {};

**operations**

**public** Renter: string \* string \* string \* string \* **nat** \* **nat** \* **nat** \* paymentMethod ==> Renter

Renter(m\_username,m\_password,m\_email,m\_name,m\_insurance\_score,m\_drivers\_licence,m\_passport,m\_pay\_type) ==

(

username := m\_username;

password := m\_password;

email := m\_email;

name := m\_name;

PaymentMethod := m\_pay\_type;

insuranceScore := m\_insurance\_score;

driversLicenceID := m\_drivers\_licence;

passportId := m\_passport;

return self

);

**public** getPaymentMethod: () ==> paymentMethod

getPaymentMethod() == (return self.PaymentMethod);

**public** setPaymentMethod: paymentMethod ==> ()

setPaymentMethod(pay\_method) == (PaymentMethod := pay\_method);

**public** getInsuranceScore: () ==> **nat**

getInsuranceScore() == (return self.insuranceScore);

**public** getDriversLicenceID : () ==> **nat**

getDriversLicenceID() == (return self.driversLicenceID);

**public** getPassportID: () ==> **nat**

getPassportID() == (return self.passportId);

**public** getBookings: () ==> **set** **of** Booking

getBookings() == (return self.bookings);

**public** addBooking: Booking ==> ()

addBooking(t\_booking) == (bookings := bookings union {t\_booking});

**end** Renter

## Lister

**class** Lister **is subclass of** User

**instance variables**

**private** PaymentMethod: paymentMethod;

**operations**

**public** Lister: string \* string \* string \* string \* paymentMethod ==> Lister

Lister(m\_username,m\_password,m\_email,m\_name,m\_pay\_type) ==

(

username := m\_username;

password := m\_password;

email := m\_email;

name := m\_name;

PaymentMethod := m\_pay\_type;

return self

);

**public** getPaymentMethod: () ==> paymentMethod

getPaymentMethod() == (return self.PaymentMethod);

**public** setPaymentMethod: paymentMethod ==> ()

setPaymentMethod(pay\_method) == (PaymentMethod := pay\_method);

**end** Lister

**class** Turo

**types**

**public** string = **seq** **of** **char**;

**public** usersSet = **set** **of** User;

**public** listingSet = **set** **of** Listing;

**instance variables**

**private** Renters: **set** **of** Renter := {};

**private** Listers: **set** **of** Lister := {};

**private** users: usersSet := {};

**private** listings: listingSet := {};

**static** **public** currUser: User := new User();

**static** **public** userType: **nat** := 0;

**inv** userType = 1 or userType = 2 or userType = 0; -- 1 -> renter 2 -> lister 0 -> not logged in

**operations**

**public** getUsers: () ==> usersSet

getUsers() == (return self.users);

**public** getListings: () ==> listingSet

getListings() == (return self.listings);

**public** registerListing: Listing ==> ()

registerListing(t\_listing) == (listings := listings union {t\_listing});

**public** removeListing: Listing ==> ()

removeListing(t\_listing) == (listings := listings \ {t\_listing});

**public** registerLister: Lister ==> ()

registerLister(t\_lister) == (Listers := Listers union {t\_lister})

**pre** not listerExists(t\_lister.getUsername());

**public** registerRenter: Renter ==> ()

registerRenter(t\_renter) == (Renters := Renters union {t\_renter})

**pre** not renterExists(t\_renter.getUsername());

**public** SearchListing: Location \* Date \* Date ==> listingSet

SearchListing(t\_location, t\_start\_date, t\_end\_date) ==

(

**dcl** found: listingSet := {};

for **all** listing **in set** listings **do**

(

**dcl** lstLoc: Location := listing.getLocation();

if(lstLoc.getCity() = t\_location.getCity() and lstLoc.getCountry() = t\_location.getCountry()) **then**

(

if(listing.getAvailableDates().availableThrough(t\_start\_date,t\_end\_date)) **then**

(

found := found union {listing};

);

);

);

return found;

);

**public** renterLogin: string \* string ==> **bool**

renterLogin(t\_username,t\_password) ==

(

**dcl** ok: **bool**:=false;

for **all** renter **in set** Renters **do**

(

if(renter.getUsername() = t\_username)**then**

(

if(renter.verifyLogin(t\_password))**then**

(

currUser := renter;

userType := 1;

return true;

)**else**

(

return false;

);

);

);

return ok;

);

**public** listerLogin: string \* string ==> **bool**

listerLogin(t\_username,t\_password) ==

(

**dcl** ok: **bool**:=false;

for **all** lister **in set** Listers **do**

(

if(lister.getUsername() = t\_username)**then**

(

if(lister.verifyLogin(t\_password))**then**

(

currUser := lister;

userType := 2;

return true;

)**else**

(

return false;

);

);

);

return ok;

);

**public** getUserListings: string ==> listingSet

getUserListings(t\_user) ==

(

**dcl** lists: listingSet := {};

for **all** listing **in set** listings **do**

(

if(listing.getLister().getUsername() = t\_user)**then**

(

lists := lists union {listing};

);

);

return lists;

);

**public** logout: () ==> ()

logout() ==

(

currUser := new User();

userType := 0;

)

**pre** userType = 1 or userType = 2

**post** userType = 0;

**public** **pure** listerExists: string ==> **bool**

listerExists(t\_user) ==

(

for **all** lister **in set** Listers **do**

(

if(lister.getUsername() = t\_user)**then**

(

return true;

);

);

return false;

);

**public** **pure** renterExists: string ==> **bool**

renterExists(t\_user) ==

(

for **all** renter **in set** Renters **do**

(

if(renter.getUsername() = t\_user)**then**

(

return true;

);

);

return false;

)

**functions**

-- TODO Define functiones here

**traces**

-- TODO Define Combinatorial Test Traces here

**end** Turo

# Model Validation

## Test Class

class TuroTests

instance variables

Turo : Turo := new Turo();

lister1 : Lister;

renter1: Renter;

review1: Review;

inbox1: Inbox;

requests: set of BookingRequest;

bookings: set of Booking;

booking1: Booking;

bookingrequest1: BookingRequest;

location1: Location;

listing1: Listing;

van1: Van;

minivan1: MiniVan;

regularcar1: RegularCar;

suv1: SUV;

truck1: Truck;

manufacturer1: Manufacturer;

basic1: Basic;

comercialplan1: ComercialPlan;

premium1: Premium;

standard1: Standard;

availabilityCalendar1: AvailabilityCalendar;

bikerack1: BikeRack;

bluetooth1 : Bluetooth;

convertible1: Convertible;

customFeature1: CustomFeature;

fourbyfour1: FourByFour;

gps1: GPS;

heatedSeats1: HeatedSeats;

skirack1: SkiRack;

snowtires1: SnowTires;

sunroof1: Sunroof;

usb1: USB;

deliveryoption1 : DeliveryOptions;

extra1: Extra;

petfriendly1: PetFriendly;

notification1: Notification;

notification2: Notification;

operations

public TuroTests: () ==> TuroTests

TuroTests() == (

return self

);

private assertTrue: bool ==> ()

assertTrue(cond) == return

pre cond;

public testCreateTuro: () ==> ()

testCreateTuro () ==

(

dcl Turo: Turo := new Turo();

assertTrue(card Turo.getUsers() = 0);

assertTrue(card Turo.getListings() = 0);

);

public testCreateRenter: () ==> ()

testCreateRenter () ==

(

renter1 := new Renter("10","20","30","40", 50, 60, 70, <DEBIT>);

);

public testGetsFromRenter: () ==> ()

testGetsFromRenter () ==

(

assertTrue(renter1.getPaymentMethod() = <DEBIT>);

assertTrue(renter1.getUsername() = "10");

assertTrue(renter1.getEmail() = "30");

assertTrue(renter1.getName() = "40");

assertTrue(renter1.verifyLogin("20"));

assertTrue(renter1.getInsuranceScore() = 50);

assertTrue(renter1.getDriversLicenceID() = 60);

assertTrue(renter1.getPassportID() = 70);

bookings := renter1.getBookings();

);

public testSetsFromRenter: () ==> ()

testSetsFromRenter () ==

(

renter1.setPaymentMethod(<CREDIT>);

assertTrue(renter1.getPaymentMethod() = <CREDIT>);

);

public testCreateLister: () ==> ()

testCreateLister () ==

(

lister1 := new Lister("1","2","3","4",<DEBIT>);

);

public testGetsFromLister: () ==> ()

testGetsFromLister () ==

(

assertTrue(lister1.getPaymentMethod() = <DEBIT>);

assertTrue(lister1.getUsername() = "1");

assertTrue(lister1.getEmail() = "3");

assertTrue(lister1.getName() = "4");

assertTrue(lister1.verifyLogin("2"));

inbox1 := lister1.getInbox();

assertTrue(card inbox1.getNotViewedNotifications() = 0);

assertTrue(card inbox1.viewNotifications() = 0);

requests := lister1.getRequests();

);

public testSetsFromLister: () ==> ()

testSetsFromLister () ==

(

lister1.setPaymentMethod(<CREDIT>);

assertTrue(lister1.getPaymentMethod() = <CREDIT>);

);

public testGetUser: () ==> ()

testGetUser() ==

(

assertTrue(card Turo.getUsers() = 0);

);

public testCreateReview: () ==> ()

testCreateReview() ==

(

review1 := new Review(lister1, 5.0, "Good Review");

assertTrue(review1.getReviewer() = lister1);

assertTrue(review1.getReviewScore() = 5.0);

assertTrue(review1.getReview() = "Good Review");

);

public testGetSetReview: () ==> ()

testGetSetReview() ==

(

lister1.addReview(review1);

assertTrue(card lister1.getReviews() = 1);

);

public testCreateManufacturer: () ==> ()

testCreateManufacturer() ==

(

manufacturer1 := new Manufacturer("Diogo");

assertTrue(manufacturer1.getName() = "Diogo");

);

public testCreateCars: () ==> ()

testCreateCars() ==

(

comercialplan1 := new ComercialPlan();

bikerack1 := new BikeRack();

assertTrue(bikerack1.getName() = "Bike Rack");

assertTrue(bikerack1.getDescription() = "This car has a bike rack");

petfriendly1 := new PetFriendly();

assertTrue(petfriendly1.getName() = "Pet Friendly");

assertTrue(petfriendly1.getDescription() = "This car is pet friendly");

bluetooth1 := new Bluetooth();

convertible1 := new Convertible();

customFeature1 := new CustomFeature("hey", "desc");

fourbyfour1 := new FourByFour();

gps1 := new GPS();

heatedSeats1 := new HeatedSeats();

skirack1 := new SkiRack();

snowtires1 := new SnowTires();

sunroof1 := new Sunroof();

usb1 := new USB();

van1:= new Van("Van", 100.0,0,1990,"BLUE",<ELECTRIC>,manufacturer1,{bikerack1, usb1});

van1.calculatePricePerDay();

minivan1 := new MiniVan("MiniVan", 100.0,0,1990,"BLUE",<DIESEL>,manufacturer1,{});

minivan1.calculatePricePerDay();

suv1 := new SUV("SUV", 100.0,0,1990,"BLUE",<ELECTRIC>,manufacturer1,{});

suv1.calculatePricePerDay();

truck1 := new Truck("Truck", 100.0,0,1990,"BLUE",<GASOLINE>,manufacturer1,{});

truck1.calculatePricePerDay();

regularcar1 := new RegularCar("RegularCar", 100.0,0,1990,"BLUE",<GASOLINE>,manufacturer1,{});

regularcar1.calculatePricePerDay();

);

public testGetFromCars: () ==> ()

testGetFromCars() ==

(

assertTrue(van1.getName() = "Van");

assertTrue(van1.getNumberOfDoors() = 2);

assertTrue(van1.getNumberOfSeats() = 2);

assertTrue(van1.getMilesPerGalon() = 100.0);

assertTrue(van1.getPricePerDay() = 3);

assertTrue(van1.getTripCount() = 0);

assertTrue(van1.getManufactureYear() = 1990);

assertTrue(van1.getColor() = "BLUE");

assertTrue(van1.getFuelType() = <ELECTRIC>);

assertTrue(van1.getVehicleType() = <VAN>);

assertTrue(van1.getManufacturer() = manufacturer1);

van1.addFeatureSet({sunroof1, snowtires1});

van1.addFeature(skirack1);

assertTrue(van1.getVehicleType() = <VAN>);

assertTrue(card van1.getFeatures() = 5);

);

public test\_dates: () ==> ()

test\_dates() ==

(

dcl DateFactory : DateFactory := new DateFactory();

dcl date : Date := DateFactory.create\_date(1,1,2019);

dcl date2 : Date := DateFactory.create\_date(1,2,2019);

dcl nextday : Date := DateFactory.create\_date(2,1,2019);

assertTrue(date.getDay() = 1);

assertTrue(date.getMonth() = 1);

assertTrue(date.getYear() = 2019);

assertTrue(date.getText() = "");

assertTrue(date.getNextDay() = nextday);

assertTrue(date.compare(date2) = false);

);

public testCreateListing: () ==> ()

testCreateListing() ==

(

dcl op: DeliveryOptions := new DeliveryOptions();

dcl DateFactory : DateFactory := new DateFactory();

dcl date : Date := DateFactory.create\_date(1,1,2019);

dcl start\_date1 : Date := DateFactory.create\_date(1,1,2019);

dcl start\_date2 : Date := DateFactory.create\_date(2,1,2019);

deliveryoption1 := op.develiveryOptions(<AIRPORT>);

deliveryoption1 := op.develiveryOptions({<AIRPORT>});

assertTrue(card op.getDeliveryOptions() = 1);

location1 := new Location("Portugal", "Porto");

basic1 := new Basic();

premium1 := new Premium();

standard1 := new Standard();

availabilityCalendar1 := new AvailabilityCalendar();

availabilityCalendar1 := new AvailabilityCalendar({start\_date1, start\_date2});

availabilityCalendar1 := new AvailabilityCalendar(start\_date1);

availabilityCalendar1.removeDate(start\_date1);

availabilityCalendar1.addDates({start\_date1, start\_date2});

assertTrue(availabilityCalendar1.areDatesAvailable({start\_date1, start\_date2}) = true);

assertTrue(availabilityCalendar1.isDateAvailable(start\_date1) = true);

assertTrue(card availabilityCalendar1.getDates() = 2);

extra1 := new Extra("fire", "hot", 1.0);

listing1 := new Listing(location1,deliveryoption1,basic1,"t","t","t",van1,availabilityCalendar1,true, lister1,{extra1});

);

public testListingGetsSets: () ==> ()

testListingGetsSets() ==

(

dcl DateFactory : DateFactory := new DateFactory();

dcl date : Date := DateFactory.create\_date(1,1,2019);

dcl start\_date : Date := DateFactory.create\_date(1,1,2019);

dcl end\_date : Date := DateFactory.create\_date(2,1,2019);

assertTrue(listing1.requestBooking(renter1,date,start\_date,end\_date,{}) = true);

assertTrue(listing1.instantBook(renter1,date,start\_date,end\_date,{}) = true);

assertTrue(listing1.getDeliveryOptions() = deliveryoption1);

assertTrue(listing1.getProtectionPlan() = basic1);

assertTrue(listing1.getGuidelines() = "t");

assertTrue(listing1.getFAQS() = "t");

assertTrue(listing1.getCar() = van1);

assertTrue(card listing1.getExtras() = 1);

assertTrue(extra1.getName() = "fire");

assertTrue(extra1.getDescription() = "hot");

assertTrue(basic1.getBenefits() = "");

assertTrue(basic1.getRevenueSplit() = 0.85);

);

public testCreateBooking: () ==> ()

testCreateBooking() ==

(

dcl DateFactory : DateFactory := new DateFactory();

dcl start\_date : Date := DateFactory.create\_date(5,1,2019);

dcl end\_date : Date := DateFactory.create\_date(6,1,2019);

booking1 := new Booking(renter1, listing1, start\_date, end\_date, {extra1});

);

public testBookingGetsSets: () ==> ()

testBookingGetsSets() ==

(

dcl DateFactory : DateFactory := new DateFactory();

dcl date : Date := DateFactory.create\_date(1,1,2019);

dcl start\_date : Date := DateFactory.create\_date(5,1,2019);

dcl end\_date : Date := DateFactory.create\_date(6,1,2019);

assertTrue(booking1.getRenter() = renter1);

assertTrue(booking1.getListing() = listing1);

assertTrue(booking1.getStartDate() = start\_date);

assertTrue(booking1.getEndDate() = end\_date);

assertTrue(booking1.isActive() = true);

assertTrue(booking1.getTotalPrice() = 4.0);

assertTrue(booking1.cancel(renter1, date) = true);

assertTrue(booking1.cancel(renter1, start\_date) = false);

);

public testCreateBookingRequest: () ==> ()

testCreateBookingRequest() ==

(

dcl DateFactory : DateFactory := new DateFactory();

dcl start\_date : Date := DateFactory.create\_date(5,1,2019);

dcl end\_date : Date := DateFactory.create\_date(6,1,2019);

bookingrequest1 := new BookingRequest(renter1, listing1, start\_date, end\_date, {extra1});

lister1.addRequest(bookingrequest1);

);

public testBookingRequestGetsSets: () ==> ()

testBookingRequestGetsSets() ==

(

dcl DateFactory : DateFactory := new DateFactory();

dcl date : Date := DateFactory.create\_date(1,1,2019);

dcl date2 : Date := DateFactory.create\_date(20,1,2019);

dcl start\_date : Date := DateFactory.create\_date(5,1,2019);

dcl end\_date : Date := DateFactory.create\_date(6,1,2019);

assertTrue(bookingrequest1.confirm(renter1, date) = false);

assertTrue(bookingrequest1.decline(renter1, date) = false);

assertTrue(bookingrequest1.getRenter() = renter1);

assertTrue(bookingrequest1.getListing() = listing1);

assertTrue(bookingrequest1.getStartDate() = start\_date);

assertTrue(bookingrequest1.getEndDate() = end\_date);

assertTrue(bookingrequest1.isActive() = true);

assertTrue(bookingrequest1.getTotalPrice() = 4.0);

assertTrue(bookingrequest1.cancel(renter1, date) = true);

);

public testTuroSets: () ==> ()

testTuroSets() ==

(

dcl DateFactory : DateFactory := new DateFactory();

dcl start\_date : Date := DateFactory.create\_date(5,1,2019);

dcl end\_date : Date := DateFactory.create\_date(6,1,2019);

Turo.registerListing(listing1);

assertTrue(card Turo.getUserListings("1") = 1);

Turo.registerLister(lister1);

Turo.registerRenter(renter1);

assertTrue(Turo.listerExists("1") = true);

assertTrue(Turo.listerExists("0") = false);

assertTrue(Turo.renterExists("10") = true);

assertTrue(Turo.renterExists("00") = false);

assertTrue(card Turo.SearchListing(location1, start\_date,end\_date) =1);

Turo.removeListing(listing1);

);

public testTuroLoginLogout: () ==> ()

testTuroLoginLogout() ==

(

assertTrue(Turo.renterLogin("10","20") = true);

assertTrue(Turo.listerLogin("1","2") = true);

Turo.logout();

assertTrue(Turo.renterLogin("6","6") = false);

assertTrue(Turo.listerLogin("5","5") = false);

assertTrue(Turo.renterLogin("10","6") = false);

assertTrue(Turo.listerLogin("1","5") = false);

);

public testCreateNotification: () ==> ()

testCreateNotification() ==

(

dcl DateFactory : DateFactory := new DateFactory();

dcl start\_date : Date := DateFactory.create\_date(5,1,2019);

notification1 := new InstantBookNotification(start\_date,lister1,van1,{start\_date},{});

notification1 := new InstantBookCancellationNotification(start\_date,lister1,van1);

notification1 := new BookingRequestNotification(start\_date,lister1,van1,{start\_date},{},bookingrequest1);

notification2 := new BookingRequestCancellationNotification(start\_date,lister1,van1);

assertTrue(notification2.getNotificationText() = ":4 has cancelled his request to book the car Van");

assertTrue(notification2.getNotificationDate() = start\_date);

assertTrue(notification2.wasViewed() = false);

notification2.setAsViewed();

);

public static main: () ==> ()

main() ==

(

dcl TuroTests: TuroTests := new TuroTests();

TuroTests.testCreateTuro();

TuroTests.testCreateLister();

TuroTests.testGetUser();

TuroTests.testGetsFromLister();

TuroTests.testSetsFromLister();

TuroTests.testCreateReview();

TuroTests.testGetSetReview();

TuroTests.testCreateRenter();

TuroTests.testGetsFromRenter();

TuroTests.testSetsFromRenter();

TuroTests.testCreateManufacturer();

TuroTests.testCreateCars();

TuroTests.testGetFromCars();

TuroTests.testCreateListing();

TuroTests.testListingGetsSets();

TuroTests.testCreateBooking();

TuroTests.testBookingGetsSets();

TuroTests.testCreateBookingRequest();

TuroTests.testBookingRequestGetsSets();

TuroTests.testTuroSets();

TuroTests.testTuroLoginLogout();

TuroTests.test\_dates();

TuroTests.testCreateNotification();

);

end TuroTests

|  |  |  |  |
| --- | --- | --- | --- |
| Function or operation | Line | Coverage | Calls |
| addFeature | 67 | 100.0% | 1 |
| addFeatureSet | 61 | 100.0% | 1 |
| calculatePricePerDay | 21 | 25.0% | 0 |
| getColor | 46 | 100.0% | 1 |
| getFeatures | 58 | 100.0% | 1 |
| getFuelType | 49 | 100.0% | 1 |
| getManufactureYear | 43 | 100.0% | 2 |
| getManufacturer | 55 | 100.0% | 1 |
| getMilesPerGalon | 34 | 100.0% | 2 |
| getName | 25 | 100.0% | 7 |
| getNumberOfDoors | 28 | 100.0% | 2 |
| getNumberOfSeats | 31 | 100.0% | 1 |
| getPricePerDay | 37 | 100.0% | 3 |
| getTripCount | 40 | 100.0% | 1 |
| getVehicleType | 52 | 100.0% | 2 |
| Car.vdmpp |  | 95.9% | 26 |

|  |  |  |  |
| --- | --- | --- | --- |
| Function or operation | Line | Coverage | Calls |
| Manufacturer | 8 | 100.0% | 1 |
| getName | 15 | 100.0% | 1 |
| Manufacturer.vdmpp | 100.0% | 2 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Function or operation | Line | Coverage | Calls |
| Manufacturer | 8 | 100.0% | 1 |
| getName | 15 | 100.0% | 1 |
| Manufacturer.vdmpp | 100.0% | 2 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Function or operation | Line | Coverage | Calls |
| RegularCar | 8 | 96.1% | 1 |
| calculatePricePerDay | 28 | 60.0% | 2 |
| RegularCar.vdmpp | 84.4% | 3 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Function or operation | Line | Coverage | Calls |
| SUV | 8 | 96.1% | 1 |
| calculatePricePerDay | 28 | 60.0% | 2 |
| SUV.vdmpp | 84.4% | 3 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| getVehicleType | 52 | 100.0% | 2 |

|  |  |  |  |
| --- | --- | --- | --- |
| Car.vdmpp | 95.9% | 26 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Function or operation | Line | Coverage | Calls |
| SUV | 8 | 96.1% | 1 |
| calculatePricePerDay | 28 | 60.0% | 2 |
| SUV.vdmpp | 84.4% | 3 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Function or operation | Line | Coverage | Calls |
| SUV | 8 | 96.1% | 1 |
| calculatePricePerDay | 28 | 60.0% | 2 |
| SUV.vdmpp | 84.4% | 3 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Function or operation | Line | Coverage | Calls |
| SUV | 8 | 96.1% | 1 |
| calculatePricePerDay | 28 | 60.0% | 2 |
| SUV.vdmpp | 84.4% | 3 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Function or operation | Line | Coverage | Calls |
| SUV | 8 | 96.1% | 1 |
| calculatePricePerDay | 28 | 60.0% | 2 |
| SUV.vdmpp | 84.4% | 3 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Function or operation | Line | Coverage | Calls |
| SUV | 8 | 96.1% | 1 |
| calculatePricePerDay | 28 | 60.0% | 2 |
| SUV.vdmpp | 84.4% | 3 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Function or operation | Line | Coverage | Calls |
| SUV | 8 | 96.1% | 1 |
| calculatePricePerDay | 28 | 60.0% | 2 |
| SUV.vdmpp | 84.4% | 3 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Function or operation | Line | Coverage | Calls |
| SUV | 8 | 96.1% | 1 |
| calculatePricePerDay | 28 | 60.0% | 2 |
| SUV.vdmpp | 84.4% | 3 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Function or operation | Line | Coverage | Calls |
| FourByFour | 7 | 100.0% | 1 |
| FourByFour.vdmpp | 100.0% | 1 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Function or operation | Line | Coverage | Calls |
| FourByFour | 7 | 100.0% | 1 |
| FourByFour.vdmpp | 100.0% | 1 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Function or operation | Line | Coverage | Calls |
| HeatedSeats | 7 | 100.0% | 1 |
| HeatedSeats.vdmpp | 100.0% | 1 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Function or operation | Line | Coverage | Calls |
| PetFriendly | 7 | 100.0% | 1 |
| PetFriendly.vdmpp | 100.0% | 1 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Function or operation | Line | Coverage | Calls |
| PetFriendly | 7 | 100.0% | 1 |
| PetFriendly.vdmpp | 100.0% | 1 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Function or operation | Line | Coverage | Calls |
| PetFriendly | 7 | 100.0% | 1 |
| PetFriendly.vdmpp | 100.0% | 1 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Function or operation | Line | Coverage | Calls |
| PetFriendly | 7 | 100.0% | 1 |
| PetFriendly.vdmpp | 100.0% | 1 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Function or operation | Line | Coverage | Calls |
| USB | 7 | 100.0% | 1 |
| USB.vdmpp | 100.0% | 1 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Function or operation | Line | Coverage | Calls |
| AvailabilityCalendar | 7 | 100.0% | 1 |
| addDates | 39 | 100.0% | 2 |
| areDatesAvailable | 30 | 100.0% | 1 |
| availableThrough | 42 | 100.0% | 9 |
| getDates | 24 | 100.0% | 1 |
| isDateAvailable | 27 | 100.0% | 1 |
| removeDate | 33 | 100.0% | 1 |
| removeDates | 36 | 100.0% | 1 |
| AvailabilityCalendar.vdmpp | 100.0% | 17 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Function or operation | Line | Coverage | Calls |
| AvailabilityCalendar | 7 | 100.0% | 1 |
| addDates | 39 | 100.0% | 2 |
| areDatesAvailable | 30 | 100.0% | 1 |
| availableThrough | 42 | 100.0% | 9 |
| getDates | 24 | 100.0% | 1 |
| isDateAvailable | 27 | 100.0% | 1 |
| removeDate | 33 | 100.0% | 1 |
| removeDates | 36 | 100.0% | 1 |
| AvailabilityCalendar.vdmpp | 100.0% | 17 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Function or operation | Line | Coverage | Calls |
| create date | 9 | 100.0% | 18 |
| DateFactory.vdmpp | 100.0% | 18 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Function or operation | Line | Coverage | Calls |
| create date | 9 | 100.0% | 18 |
| DateFactory.vdmpp | 100.0% | 18 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Function or operation | Line | Coverage | Calls |
| Extra | 9 | 100.0% | 1 |
| getCost | 25 | 100.0% | 2 |
| getDescription | 22 | 100.0% | 1 |
| getName | 19 | 100.0% | 1 |
| Extra.vdmpp | 100.0% | 5 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Function or operation | Line | Coverage | Calls |
| Listing | 18 | 100.0% | 1 |
| getAvailableDates | 57 | 100.0% | 4 |
| getCar | 54 | 100.0% | 3 |
| getDeliveryOptions | 39 | 100.0% | 1 |
| getExtras | 63 | 100.0% | 1 |
| getFAQS | 51 | 100.0% | 2 |
| getGuidelines | 45 | 100.0% | 2 |
| getLister | 60 | 100.0% | 3 |
| getLocation | 36 | 100.0% | 1 |
| getParkingDetails | 48 | 0.0% | 0 |
| getProtectionPlan | 42 | 100.0% | 1 |
| instantBook | 86 | 93.0% | 1 |
| requestBooking | 66 | 90.2% | 0 |
| Listing.vdmpp |  | 92.8% | 20 |

|  |  |  |  |
| --- | --- | --- | --- |
| Function or operation | Line | Coverage | Calls |
| Location | 8 | 100.0% | 1 |
| getCity | 19 | 100.0% | 4 |
| getCountry | 16 | 100.0% | 2 |
| Location.vdmpp | 100.0% | 7 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Function or operation | Line | Coverage | Calls |
| Basic | 6 | 100.0% | 1 |
| Basic,vdmpp |  | 100.0% | 1 |

|  |  |  |  |
| --- | --- | --- | --- |
| Function or operation | Line | Coverage | Calls |
| ComercialPlan | 6 | 100.0% | 1 |
| ComercialPlan.vdmpp | 100.0% | 1 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Function or operation | Line | Coverage | Calls |
| ComercialPlan | 6 | 100.0% | 1 |
| ComercialPlan.vdmpp | 100.0% | 1 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Function or operation | Line | Coverage | Calls |
| ComercialPlan | 6 | 100.0% | 1 |
| ComercialPlan.vdmpp | 100.0% | 1 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Function or operation | Line | Coverage | Calls |
| ComercialPlan | 6 | 100.0% | 1 |
| ComercialPlan.vdmpp | 100.0% | 1 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Function or operation | Line | Coverage | Calls |
| TuroTests | 40 | 100.0% | 1 |
| assertTrue | 45 | 100.0% | 192 |
| main | 215 | 100.0% | 1 |
| testBookingGetsSets | 222 | 100.0% | 1 |
| testBookingRequestGetsSets | 245 | 100.0% | 1 |
| testCreateBooking | 196 | 100.0% | 1 |
| testCreateBookingGetsSets | 245 | 100.0% | 1 |
| testCreateBookingRequest | 204 | 100.0% | 1 |
| testCreateCars | 139 | 100.0% | 1 |
| testCreateLister | 84 | 100.0% | 1 |
| testCreateListing | 185 | 100.0% | 3 |
| testCreateManufacturer | 133 | 100.0% | 1 |
| testCreateNotification | 347 | 100.0% | 1 |
| testCreateRenter | 57 | 100.0% | 1 |
| testCreateReview | 116 | 100.0% | 1 |
| testCreateTuro | 49 | 100.0% | 1 |
| testGetFromCars | 166 | 100.0% | 1 |
| testGetSetReview | 126 | 100.0% | 1 |
| testGetUser | 109 | 100.0% | 1 |
| testGetsFromLister | 90 | 100.0% | 1 |
| testGetsFromRenter | 63 | 100.0% | 1 |
| testListingGets | 197 | 100.0% | 1 |
| testListingGetsSets | 197 | 100.0% | 1 |
| testSetsFromLister | 102 | 100.0% | 1 |
| testSetsFromRenter | 77 | 100.0% | 1 |
| testTuroLoginLogout | 228 | 100.0% | 1 |
| testTuroSets | 214 | 100.0% | 1 |
| test dates | 189 | 100.0% | 1 |
| TuroTests.vdmpp | 100.0% | 221 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Function or operation | Line | Coverage | Calls |
| SearchListing | 38 | 100.0% | 1 |
| getListings | 21 | 100.0% | 1 |
| getUserListings | 105 | 100.0% | 1 |
| getUsers | 18 | 100.0% | 4 |
| listerExists | 130 | 100.0% | 2 |
| listerLogin | 81 | 100.0% | 1 |
| logout | 121 | 100.0% | 1 |
| registerLister | 30 | 100.0% | 2 |
| registerListing | 24 | 100.0% | 1 |
| registerRenter | 34 | 100.0% | 2 |
| removeListing | 27 | 100.0% | 1 |
| renterExists | 144 | 100.0% | 2 |
| renterLogin | 58 | 100.0% | 1 |
| Turo.vdmpp |  | 100.0% | 20 |

|  |  |  |  |
| --- | --- | --- | --- |
| Function or operation | Line | Coverage | Calls |
| Booking | 12 | 100.0% | 2 |
| cancel | 23 | 100.0% | 2 |
| getEndDate | 47 | 100.0% | 1 |
| getListing | 41 | 100.0% | 1 |
| getRenter | 38 | 100.0% | 1 |
| getStartDate | 44 | 100.0% | 1 |
| getTotalPrice | 53 | 100.0% | 1 |
| isActive | 50 | 100.0% | 1 |
| Booking.vdmpp | 100.0% | 10 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Function or operation | Line | Coverage | Calls |
| BookingRequest | 12 | 100.0% | 2 |
| cancel | 56 | 88.4% | 2 |
| confirm | 23 | 45.0% | 0 |
| decline | 41 | 55.5% | 0 |
| getEndDate | 80 | 100.0% | 1 |
| getListing | 74 | 100.0% | 1 |
| getRenter | 71 | 100.0% | 1 |
| getStartDate | 77 | 100.0% | 1 |
| getTotalPrice | 86 | 100.0% | 1 |
| isActive | 83 | 100.0% | 1 |
| BookingRequest.vdmpp | 74.6% | 10 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Function or operation | Line | Coverage | Calls |
| getAllNotifications | 7 | 0.0% | 0 |
| getNotViewedNotifications | 10 | 33.3% | 0 |
| registerNotification | 37 | 100.0% | 2 |
| viewNotifications | 26 | 75.0% | 1 |
| Inbox.vdmpp | 51.3% | 3 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Function or operation | Line | Coverage | Calls |
| BookingRequestCancellationNotification | 5 | 100.0% | 1 |
| BookingRequestCancellationNotification.vdmpp |  | 100.0% | 1 |

|  |  |  |  |
| --- | --- | --- | --- |
| Function or operation | Line | Coverage | Calls |
| BookingRequestNotification | 11 | 100.0% | 2 |
| getRequest | 21 | 0.0% | 0 |
| BookingRequestNotification.vdmpp | 85.7% | 2 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Function or operation | Line | Coverage | Calls |
| InstantBookCancellationNotification | 5 | 100.0% | 1 |
| InstantBookCancellationNotification.vdmpp | 100.0% | 1 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Function or operation | Line | Coverage | Calls |
| InstantBookCancellationNotification | 5 | 100.0% | 1 |
| InstantBookCancellationNotification.vdmpp | 100.0% | 1 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Function or operation | Line | Coverage | Calls |
| getNotificationDate | 12 | 100.0% | 2 |
| getNotificationText | 9 | 100.0% | 1 |
| setAsViewed | 18 | 100.0% | 1 |
| wasViewed | 15 | 100.0% | 1 |
| Notification.vdmpp | 100.0% | 5 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Function or operation | Line | Coverage | Calls |
| Lister | 5 | 100.0% | 1 |
| getPaymentMethod | 17 | 100.0% | 2 |
| setPaymentMethod | 20 | 100.0% | 1 |
| Lister.vdmpp | 100.0% | 4 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Function or operation | Line | Coverage | Calls |
| Lister | 5 | 100.0% | 1 |
| getPaymentMethod | 17 | 100.0% | 2 |
| setPaymentMethod | 20 | 100.0% | 1 |
| Lister.vdmpp | 100.0% | 4 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Function or operation | Line | Coverage | Calls |
| Lister | 5 | 100.0% | 1 |
| getPaymentMethod | 17 | 100.0% | 2 |
| setPaymentMethod | 20 | 100.0% | 1 |
| Lister.vdmpp | 100.0% | 4 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Function or operation | Line | Coverage | Calls |
| addRequest | 33 | 100.0% | 3 |
| addReview | 36 | 100.0% | 1 |
| getEmail | 18 | 100.0% | 2 |
| getInbox | 27 | 100.0% | 3 |
| getName | 21 | 100.0% | 8 |
| getRequests | 30 | 100.0% | 2 |
| getReviews | 39 | 100.0% | 1 |
| getUsername | 15 | 100.0% | 25 |
| verifyLogin | 24 | 100.0% | 6 |
| User.vdmpp | 100.0% | 51 |  |

# Model Verification

To observe a pre-condition in action we merely need to try to create a Date with an invalid date, we can use the code DateFactory`create\_date(0,1,1); this date is invalid due to the day number being under 1, as such the pre-condition in this operation should kick into gears and now allow the operation to go through.

There is no operation that currently that can break an invariant, however we can add code to the Turo class that would break an invariant and not be allowed, we can add code that sets userType to something other than 0,1 or 2, that would break the invariant present in the class and not be allowed.

# Code Generation

To generate our java code, we merely must select the project with the right mouse button and press code generation the select generate java code.

The generated java code was however not of the quality we expected, union types or enums are translated into quote types in java instead of a union class as expected and would be correct, on top of that all generated references to those union types or enums we incorrectly imported.

Besides that the only other major problems we had was with the design of the VDM set data structure, instead of being translated into a HashSet with the correct type for the java generic it is instead translated into a VDMSet class that is an extension of the LinkedHashSet but offer no useful method and only stores objects, we believe this is very bad code and the correct translation would be one of the java set generic class initialized to use the correct type of the data it was holding.

We were also no happy with the way natural numbers and real numbers were translated into a generic Number class instead of using the data types that are base to java that they represent such as integer and double.

All these things together make the generated code harder to read and work with, this also meant that we had to write code that went around the issues with the generated code. Despite this the code execution was correct and is working as expected.

# Conclusions

We were able to reach the results we wanted, creating a somewhat faithful reproduction systems behind Turo within what is possible with VDM++ and our knowledge of it. The core functionality is all present and we were able to add some non-core functionality such as the review system and the notification system.

Some things we believe can be improved is the price per day calculation system that is currently very basic despite being setup to be much more complex and intricate, as well as creating a better set of notifications that represent more actions in the system. Regarding code we believe the only improvements that can be made are adding more pre-conditions that make sure parameters of say constructors are safe and improving some our variable names and overall code layout. Despite all this we believe we have achieved a good product given our resources.

While we do believe we would have been able to create a better product in a more conventional language such as Java or C++ we are still proud of what we have achieved.

Work was distributed as evenly as possible with the contributing of each team member being 33.3% as per our own evaluation.