turo

January 7, 2019

Contents

```
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
```

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

```
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
```

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

```
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
```

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

```
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
```

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

```
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
```

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

```
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
```

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

```
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
```

Function or operation	Line	Coverage	Calls
Van	8	96.1%	1
calculatePricePerDay	28	100.0%	2
Van.vdmpp		97.7%	3

```
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
```

Function or operation	Line	Coverage	Calls
BikeRack	7	100.0%	1
BikeRack.vdmpp		100.0%	1

```
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
```

Function or operation	Line	Coverage	Calls
Bluetooth	7	100.0%	1
Bluetooth.vdmpp		100.0%	1

```
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
```

Function or operation	Line	Coverage	Calls
Convertible	7	100.0%	1
Convertible.vdmpp		100.0%	1

```
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
```

Function or operation	Line	Coverage	Calls
CustomFeature	5	100.0%	1
CustomFeature.vdmpp		100.0%	1

```
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
```

Function or operation	Line	Coverage	Calls
getDescription	12	100.0%	2
getName	9	100.0%	2
getValue	15	100.0%	4
Feature.vdmpp		100.0%	8

```
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
```

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

```
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
```

Function or operation	Line	Coverage	Calls
GPS	7	100.0%	1
GPS.vdmpp		100.0%	1

```
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
```

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

```
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
```

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

```
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
```

Function or operation	Line	Coverage	Calls
SkiRack	7	100.0%	1
SkiRack.vdmpp		100.0%	1

```
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
```

Function or operation	Line	Coverage	Calls
SnowTires	7	100.0%	1
SnowTires.vdmpp		100.0%	1

```
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
```

Function or operation	Line	Coverage	Calls
Sunroof	7	100.0%	1
Sunroof.vdmpp		100.0%	1

```
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
```

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

```
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
```

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

```
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
```

Function or operation	Line	Coverage	Calls
Date	16	100.0%	6
compare	37	73.6%	0
daysSinceStart	47	100.0%	22
daysToDate	53	100.0%	8
getDatesTo	82	59.0%	0
getDay	28	100.0%	2
getMonth	31	100.0%	1
getNextDay	59	64.1%	1
getText	40	100.0%	1
getYear	34	100.0%	1
Date.vdmpp		81.6%	42

```
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
```

Function or operation	Line	Coverage	Calls
create_date	9	100.0%	18
DateFactory.vdmpp		100.0%	18

```
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
```

Function or operation	Line	Coverage	Calls
develiveryOptions	7	100.0%	1
getDeliveryOptions	24	100.0%	1
DeliveryOptions.vdmpp		84.2%	2

```
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
```

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

```
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
```

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

```
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
```

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

```
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
```

Function or operation	Line	Coverage	Calls
Basic	6	100.0%	1

Basic.vdmpp		100.0%	1

```
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
```

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

```
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
```

Function or operation	Line	Coverage	Calls
Premium	6	100.0%	1
Premium.vdmpp		100.0%	1

```
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
```

Function or operation	Line	Coverage	Calls
getBenefits	8	100.0%	1
getRevenueSplit	11	100.0%	1
ProtectionPlan.vdmpp		100.0%	2

```
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
```

Function or operation	Line	Coverage	Calls
Standard	6	100.0%	1
Standard.vdmpp		100.0%	1

```
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
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

```
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
```

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

```
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
```

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

```
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
```

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

```
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
```

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

```
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
```

Function or operation	Line	Coverage	Calls

BookingRequestCancellationNotification	5	100.0%	1
BookingRequestCancellationNotification.vdmpp		100.0%	1

```
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
```

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

```
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
```

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

```
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
```

Function or operation	Line	Coverage	Calls
InstantBookNotification	10	100.0%	2
InstantBookNotification.vdmpp		100.0%	2

```
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
```

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

```
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
```

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

```
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
```

Function or operation	Line	Coverage	Calls
Renter	9	100.0%	1
addBooking	42	100.0%	1
getBookings	39	100.0%	1
getDriversLicenceID	33	100.0%	1
getInsuranceScore	30	100.0%	1
getPassportID	36	100.0%	1
getPaymentMethod	24	100.0%	4
setPaymentMethod	27	100.0%	1
Renter.vdmpp		100.0%	11

```
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
```

Function or operation	Line	Coverage	Calls
Review	9	100.0%	1
getReview	25	100.0%	1
getReviewScore	22	100.0%	1
getReviewer	19	100.0%	1
Review.vdmpp		100.0%	4

```
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
```

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