**ECC Academic Practice 2**

Context / Purpose:

The goal of the practice is to create a booking software application for a parking rent company. In this practice, we will focus on the rent price calculator.

There are two types of usage:

* park vehicles
* provide areas for exhibition stands during trade fairs

There are 3 types of vehicles:

* Car
* Truck
* Motorcycle

A vehicle shall store the following information:

* The number of kilometers

A car shall store the following information:

* The type of car (cabriolets, minivan, pickup…)
* The number of doors

A truck shall store the following information:

* The weight
* The length
* The height

A motorcycle shall store the following information:

* The type of motorcycle (roadster, custom, trail, racing…)

The value of a vehicle is calculated according to the number of kilometers:

* For a Truck :
  + < 100 000 km : costly
  + > 100 000 km and < 400 000 km : medium
  + > 400 000 km : cheap
* For a car :
  + < 10 000 km : costly
  + > 10 000 km and < 200 000 km : medium
  + > 200 000 km : cheap
* For a Motorcycle:
  + < 2 000 km : costly
  + > 2 000 km and < 20 000 km : medium
  + > 20 000 km : cheap

A motorcycle needs a small parking space.

A car needs a medium parking space.

A truck needs a large parking space.

An exhibition stand shall store the following information:

* The size (small, medium or large)
* The value of selling stuff (cheap, medium or costly)

A parking space shall store the following information:

* The size (small, medium or large)
* If it is available

The parking is composed of 30 parking spaces:

* 15 small spaces
* 10 medium spaces
* 5 large spaces

All vehicles and exhibition stand must implement the generic interface “ParkingUsage” which has two methods:

* getSize() : returns the size needed (small, medium or large).
* getValue() : returns the value of item parked (cheap, medium or costly).

The software shall provide a component (java object / not jacomo) called “ParkingManager”.

It shall provide the following functions:

* Return a price according to “ParkingUsage” interface passed in parameter.
* Take the booking into account (space booking)

The price computing must follow the following rules:

* Price = [size] X [value]
* Size:
  + Small : 10
  + Medium : 20
  + Large : 30
* Value :
  + Cheap : 5
  + Medium : 10
  + Costly : 20
* If all spaces are available, price is reduced from 10%.
* If only half spaces according to usage are available, price grows up 30%.
* If only one space according to usage is available, price is multiply by 2.

The “ParkingManager” aggregates the parking.

**1 – UML Conception:**

Draw a UML diagram representing the system described in context part.

When it’s done, scan the diagram and save it in the result folder of the practice. ([result-folder]/Ex1).

**2 – IDE:**

Open eclipse and create a new java project called “ParkingRentProject”

Open Java Build path and add JUnit library.

When it’s done, take a screenshot and save it in the result folder of the practice. ([result-folder]/Ex2).

**3 – Data Model:**

Create all interfaces and/or classes needed to represent all types of vehicles, exhibition stands and parking spaces.

*Technical expectations:*

* *Classical methods shall be implemented (equals, hasCode, toString…)*
* *The javaBeans check rules must be followed.*
* *Package organization must be understandable and logical.*
* *Visibility of methods and attributes must be correct.*

When it’s done, copy all java source files in the result folder of the practice. ([result-folder]/Ex3).

**4 – ParkingManager**

Create a Parking class with a collection of parking spaces according to the context description.

Create all interfaces and/or classes needed for the ParkingManager:

* It shall aggregate a Parking (attribute of class)
* It shall allow to compute and return the rent price according to parking usage
  + int computePrice(ParkingUsage parkingUsage);
* It shall allow to book parking spaces
  + void booking(ParkingUsage parkingUsage);
* It shall allow to get the list of available parking spaces
  + List<ParkingSpace> getAvailableParkingSpaces();
* It shall allow to verify if the parking is full
  + boolean isParkingFull();

*Technical expectations:*

* *The javaBeans check rules must be followed.*
* *Package organization must be understandable and logical.*
* *Visibility of methods and attributes must be correct.*

When it’s done, copy all java source files in the result folder of the practice. ([result-folder]/Ex4).

**5 – Unitary tests**

Create a JUnit unitary test to verify the good behaviors of the ParkingManager component:

* Test01 - Booking limits:
  + Create a car instance
  + Call the booking method of ParkingManager
  + Call the ParkingManager to verify if parking is full
  + Use assertion to check that’s false
  + Call the booking method of ParkingManager 9 more times
  + Call the ParkingManager to verify if parking is full
  + Use assertion to check that’s true
* Test02 - Price computing:
  + Create a motorcycle
  + Set 1 500 km as number of kilometers
  + Call the ParkingManager to compute the price
  + Use assertion to check the price is equals to 180$ (more or less 10$)
  + Explain in a code comment why the result must be equals to 180$

When it’s done, copy all java source files in the result folder of the practice. ([result-folder]/Ex5).

**6 – Debug**

Create a main class to launch the software.

Create an exhibition stand instance.

Call the method of the ParkingManager to compute rent price.

Go to ParkingManager class and add a breakpoint on the first line of compute rent price method.

Run in debug mode the software.

When the breakpoint is reached, take a screenshot and save it in the result folder of the practice. ([result-folder]/Ex6).

In variable debug tab, choose an attribute of the class and add on it a “Toggle Watchpoint”.

This new breakpoint must trigger only on modifications of the variable. (Modify parameters if needed)

Take a screenshot of the breakpoint tab and save it in the result folder of the practice. ([result-folder]/Ex6).

**7 – Swing**

Create a frame with the following widgets:

* A label named “Usage”.
* A combo box to choose the type of usage. (car, truck, motorcycle or exhibition stands)

According to the usage selection, the HMI will display only:

* For all types of vehicles :
  + A label named “kilometers”
  + A textfield to enter kilometers value
* For exhibition stands :
  + A label named “Size”
  + A combo box to choose the type of size (small, medium or large)
  + A label named “Value”
  + A combo box to choose value of selling stuff (cheap, medium or costly)
* A button to start price computing.
* A textArea to display results formatted like this: ex: “100$”

When the user clicks on “Calculate price” button, a usage instance is created (vehicles or exhibition stand) and details shall be getting from the good form and store in the object. This one shall be given to the price computing method of ParkingManager.

The result of the ParkingManager component price computing shall be displayed in the result textArea.

When it’s done, copy all java source files in the result folder of the practice. ([result-folder]/Ex7).