Computer Science 2A

Practical Assignment 07

Assignment date:	2023-05-02
Deadline	2023-05-09 12h00
Marks: 130	

This practical assignment must be uploaded to eve.uj.ac.za **before** 2023-05-09 12h00. Late[1] or incorrect submissions **will not be accepted**, and will therefore not be marked. You are **not allowed to collaborate** with any other student.

Good coding practices include a proper coding convention and a good use of documentation. Marks will be deducted if these are not present. Every submission **must** include a batch file unless stated otherwise.

The **reminder page** includes details for submission. Please ensure that **ALL** submissions follow the guidelines. The reminder page can be found on the last page of this practical.

This practical aims to familiarise you with ***Abstract Factory Design Pattern***.

The **Milky Way Space Communication Board (MWSCB)**[2] would like to make an upgrade on the vehicles used for interplanetary travel. They would like you as a contractor to build a tool that interfaces with the different factories that manufacture Military and Civilian vehicles, both **`Spaceship`**s and **`Rover`**s.

Since you are only dealing with factories your implementation will not include a complex GUI or work with files. You have rather been provided with a *Main.java* class that makes use of the factory that you will create, and an E_PLANET enum that contains all necessary planets.

The factory creates two types of Vehicles, **`Spaceship`** and **`Rover`**, which are either *Military* or *Civilian*. Create *Military* and *Civilian* Vehicle factories (concrete implementations) that will manufacture **`Spaceship`**s and **`Rover`**s.

The interface below describes the factory interface:

```
1 public interface VehicleFactory {
2   RoverVehicle createRover(E_PLANET planet, boolean hasArmourPlating, boolean
    ↪  hasWeaponMounts);
3   SpaceshipVehicle createSpaceship(String type, boolean manned);
4 }
```

[1] Alternate arrangements for exceptional circumstances will been posted on eve.

[2] Disclaimer - This series of problem statements are a work of fiction. Names, characters, businesses, places, events and incidents are either the products of the author's imagination or used in a fictitious manner. Any resemblance to actual persons, living or dead, or actual events is purely coincidental.

The **RoverVehicle** interface:

```
1 public interface RoverVehicle {
2   void drive();
3 }
```

The **SpaceshipVehicle** interface:

```
1 public interface SpaceshipVehicle {
2   void fly();
3 }
```

Create a **Java** application with the following:

- Create an Abstract class for **Rover** that has the following properties and methods:
  - *hasWeaponMounts* and *hasArmourPlating* both booleans
  - planet of type E_PLANET
  - all the necessary accessor and mutator methods [3]
- Create an Abstract class for **Spaceship** that has the following properties and methods:
  - *manned* as a boolean (the **Spaceship** is either manned (true) or unmanned(false))
  - all the necessary accessor and mutator methods [3]
- Create classes for each **Rover** type that will also implement the **RoverVehicle** interface (i.e. **EarthTraveller**, **MarsRover**, **MercuryExplorationRover** and **VenusPathfinder**). The additional features are:
  - **EarthTraveller** - ATVClass of type integer (level of all-terrain-vehicle)
  - **MarsRover** - numWheels and numArms both of type integer
  - **MercuryExplorationRover** - temp and numMinerals both of type integer
  - **VenusPathfinder** - temp of type integer and atmosphericPressure of type double
  - accessors and mutators where you deem necessary [3]
  - the ***drive*** method prints a message stating the type of **Rover** being driven, and its properties
- Create classes for each **Spaceship** type that will also implement the **SpaceshipVehicle** interface (i.e. **Atmospheric**, **Orbiter**, **Passenger** and **RoverCarrier**). The additional features are:
  - **Atmospheric** - numSensors of type integer and planet of type E_PLANET
  - **Orbiter** - planet of type E_PLANET
  - **Passenger** - numPassengers of type integer
  - **RoverCarrier** - rovers of type *ArrayList<Rover>*
  - accessors and mutators where you deem necessary [3]
  - the ***fly*** method prints a message stating the type of **Spaceship** being flown, and its properties
- Create classes for each Factory [4] concrete implementation of the **VehicleFactory** interface.
  - Each factory must create be able to create both **RoverVehicle** and **SpaceshipVehicle**

[3] Refer to Main.java to match method names
[4] Hint - CivilianFactory and MilitaryFactory

- **`Main`**
  - Should make use of the classes you have created to create either Military or Civilian Vehicles
  - Do not add code (only comment out). You are free to comment out a section that stops your code from compiling or running (*Note*: You could possibly loose marks as a result)

Finally the provided **`Main`** will allow the creation of the correct vehicles based on user input and always refer to the *Main.java* to ensure method names match. Remember to place the relevant classes into the **`csc2a.models.rover`**, **`csc2a.models.spaceship`** and **`csc2a.factory`** sub-packages[5]

[5] Hint: **`Rover`** related classes should appear in the **`csc2a.models.rover`**, **`Spaceship`** related classes should appear in the **`csc2a.models.spaceship`**, and **`Factory`** related classes should appear in the **`csc2a.factory`** sub-package.

Mark sheet

1. UML class diagrams for all classes. **[15]**
2. Abstract **`Rover`** and **`Spaceship`** **[06]**
3. Model classes per **`Rover`** (2 marks per type of **`Rover`**) **[08]**
4. ***`drive`*** method for **`Rover`** types (2 marks per type of **`Rover`**) **[08]**
5. Model classes per **`Spaceship`** (2 marks per type of **`Spaceship`**) **[08]**
6. ***`fly`*** method for **`Spaceship`** types (2 marks per type of **`Spaceship`**) **[08]**
7. **`MilitaryFactory`** concrete implementation **[16]**
8. **`CivilianFactory`** concrete implementation **[16]**
9. Packages **[05]**
10. Coding convention (structure, layout, OO design) **[05]**
11. Commenting (normal and JavaDoc commenting). **[05]**
12. Correct execution **[30]**

NB

Submissions which **do not compile** will be capped at 40%!

Practical marks are awarded subject to the student's ability to explain the concepts and decisions made in preparing the practical assignment solution. (Inability to explain code = inability to be given marks.)

Execution marks are awarded for a correctly functioning application and not for having related code.

Reminder

Your submission must follow the naming convention below.

`SURNAME_INITIALS_STUDENTNUMBER_SUBJECTCODE_YEAR_PRACTICALNUMBER`

Example

Surname	Berners-Lee	**Module Code**	CSC02A2
Initials	TJ	**Current Year**	2023
Student number	209912345	**Practical number**	P07

`Berners-Lee_TJ_209912345_ CSC02A2_2023_P07`

Your submission must include the following folders:

Folder	State	Purpose
`bin`	*Required*	Should be empty at submission but will contain runnable binaries when your submission is compiled.
`docs`	*Required*	Contains the batch file to compile your solution, UML diagrams, and any additional documentation files. All files must be in **PDF** format. Your details must be included at the top of any **PDF** files submitted. **Do not include generated JavaDoc.**
`src`	*Required*	Contains all relevant source code. Source code must be places in relevant sub-packages! Your details must be included at the top of the source code.
`data`	*Optional*	Contains all data files needed to run your solution.
`lib`	*Optional*	Contains all libraries needed to compile and run your solution.

NB

Every submission **must** include a batch file that contains commands which will:

- Compile your Java application source code.

- Compile the associated application JavaDoc.
- Run the application.

Do not include generated JavaDoc in your submission. All of the classes/methods which were created/updated need to have JavaDoc comments.

Multiple uploads

Note that only **one** submission is marked. If you already have submitted once and want to upload a newer version then submit a newer file with the same name as the uploaded file in order to overwrite it.