COSC 3P91
Assignment 3

Due date: March 24th, 2022 at 23:59 (11:59 pm).
Team: The assignment can be completed in groups of up to two students.
Delivery method: the student needs to delivery the assignment only through Sakai.
Delivery contents: document with answers and [Java, C, C++] codes if applicable (see Submission instructions).

Attention: check the Late Assignment Policy.

Assignment Overview

The assignment consists of modifying the current implementation of the Traffic Simulation Game. The game makes use of a console as an interface for the interactions of a player. The console is basically System.in/out for I/O communication with the player. Ideally, your current implementation, obtained from Assignment 2, will form the basis of your modifications. The modifications aim to benefit transparency, encapsulation, and extensibility. Even though the patterns and aspects should be contemplated in initial parts of the design, implementing them at this stage will demand efforts that require fluent comprehension of the 'application' and Design Patterns.

The Game

The **game** is a simplified traffic simulator where vehicles move around in a urban traffic network. The game is a turn-based simulator that updates the position of vehicles each "time step".

The **traffic network**, urban environment, is basically a map coded in the form of a graph, where edges are road segments and nodes are intersections. The map can be defined statically but is kept in memory as objects. A file can define the structure of the graph and be loaded at the beginning of the game (at its startup). Each road segment maybe composed of more than 0 lanes.

The **mobile entities** in the simulator are vehicles, which can be cars, trucks, or buses. As defined as a game, certain vehicles can be controlled by users while other are just controlled by the Game Engine.

The game follows a **time-stepped simulation**, where from time to time, the state of moving entities are updated accordingly, and the user is prompted for a some decision. Therefore, vehicles take movement/routing decisions, such as turning left, turning right, and going straight in intersections, whenever approaching a "decision position" in the map.

As the **main objective** of the game, **users** need to conduct their vehicle properly over the traffic network to avoid collisions.

Assignment Details

Redesign of the Traffic Simulation Game so that it contemplates a series of Design Patterns. All the game functionalities, requested from Assignment 2, should be still provided in the redesigned code of Assignment 3. The Game should allow *Looking*, *Moving*, *Asserting*, and *Gambling*; the movement decisions should be performed at intersections and middle of lanes, considering *Intersection Decision*, *Lane Changing*, *Challenge*, *Reputation*, *Reputation Value*, and *Damage*.

Since the Traffic Simulation Game is by definition an interactive system that involves the user, at least one player, it can greatly benefit from an Architectural Design Pattern: **Model View Controller** (MVC). Thus, ***(i)*** students will have modify and ensure that their program design follows an MVC Pattern.

Also, students will have to ***(ii)*** apply either **Factory** or **Abstract Factory Creational Design Pattern** in some parts of their code so that there is a more transparent handling of object instantiation. They will have to

(iii) apply the **Singleton Creational Design Pattern** and ***(iv)*** the **Composite Structural Design Pattern** in their code.

Finally, the execution of the simulator relies on a loaded traffic network (map). This map should be loaded from file and created/instantiated before the game starts. The elements of the map must be properly stored in an ***(v)*** **XML file**, created by the student. The XML file must follow a ***(vi)*** **XSD Schema**, which is defined by the student.

Important Notes

1. Applying the ***(i)*** MVC Pattern of the code may require modifications in the code. The amount of modifications depends on how the design and code have been performed in Assignment 2.
   Note: the student should explicitly and objectively explain in the **Description File** how their updated design now follows the MVC Pattern, pointing out how interactions and which classes below to the parts of MVC.
2. The student may choose to use either ***(ii)*** Factory or Abstract Factory to implement certain transparency when instantiating classes for the simulator. Please note that it is not necessary to apply such Creational Design Patterns all over the code. The student can choose which parts of the code are more suitable to use these Patterns; the patterns should be employed at least once in the code.
   Note: again, the student should indicate which parts of the code (tell which classes – the client requesting instantiation and classes being instantiated) are using Factory of Abstract Factory Patterns. The student should also explicitly and objectively explain in the **Description File** how part(s) of their updated design now follow(s) such Pattern, pointing out how interactions and which classes follow now the Pattern.
3. The student will need to modify some parts of the code in order to apply ***(iii)*** Singleton Creational Design Pattern. To make clear, please note that it is not necessary to apply such Creational Design Pattern all over the code: only on the parts that require a single instance running throughout the whole simulation. The student can choose which parts of the code are more suitable to use this Pattern; the pattern should be employed at least once in the code.
   Note: again, the student should indicate which parts of the code (tell which classes – the client requesting instantiation and classes being instantiated) are using Singleton Creational Design Pattern. The student should also explicitly and objectively explain in the **Description File** how part(s) of their updated design now follow(s) such Pattern, pointing out how interactions and which classes follow now the Pattern.
4. The student will need to modify some parts of the code in order to apply ***(iv)*** Composite Structural Design Pattern. To make clear, please note that it is not necessary to apply such Structural Design Pattern all over the code: only on the parts that can follow a recursive structural model. The student can choose which parts of the code are more suitable to use this Pattern; the pattern should be employed at least once in the code.
   Note: again, the student should indicate which parts of the code (tell which classes – the client requesting instantiation and classes being instantiated) are using Composite Structural Design Pattern. The student should also explicitly and objectively explain in the **Description File** how part(s) of their updated design now follow(s) such Pattern, pointing out how interactions and which classes follow now the Pattern.
5. The student will need to modify some initialization parts of the code in order to have the map of the traffic network ***(v)*** loaded up and parsed from an XML file. The XML file must follow an ***(vi)*** Schema that must also be defined in an XSD file.
   Note: again, the student should indicate which parts of the code (tell which classes – the client requesting instantiation and classes being instantiated) are loading the XML file and conducting the parsing. The student should also explicitly and objectively explain in the **Description File** how part(s) of their

updated design now follow(s) such modifications, Pattern, pointing out how interactions and which classes interact to load the map.

6. **Reiterating** – Recall that this course is about advanced object-oriented programming. Thus, you will want to ensure that your implementation makes use of object-oriented constructs, such as interfaces, inheritance, enumerations, and generics, where appropriate. Such constructs should have been identified during design.
7. **Reiterating** – Be sure to include sufficient comments which should consist of, at minimum, an appropriate comment for each file and method. Consider using Javadoc style comments. If you use NetBenas, Javadoc comments can be auto-created in NetBeans by placing the cursor above a method or a class that currently has no Javadoc comment, typing "/**", then pressing Enter. This will create a template Javadoc comment with some information automatically completed for you (e.g., parameters for a method).
    - Your comments, if done correctly, should facilitate writing the document that explains your code. The comments should match with what you have in your description document.
8. Besides the Object Orientation concepts, you must be able to show complete understanding of the following concepts and employ them in your code. **Adequately use them whenever possible and explain them, pointing out in your description file** (in other words, marking will consist checking the existence and proper use of each of these elements):
    (a) Factory or Abstract Factory Creational Design Pattern;
    (b) Singleton Creational Design Pattern;
    (c) Composite Structural Design Pattern;
    (d) Loading and Parsing XML files (using the parser provided in Java Standard library);
    (e) Defining the XML Schema in a XSD file.

Submission Material

The submission for this assignment will consist of two parts:

1. A ***Description document (PDF)***. A document succinctly describing your design and implementation decisions is necessary. Also, you will find it beneficial to justify your design choices such that the marker does not have to reason about why you have designed your system as you have. Make sure that the description and reasoning of your decisions are consistent with your Java code.
    - ***Latex template - a must for writing your assignment***. For writing your description file, use the Latex template enclosed in this assignment (update it accordingly!). You do not need to install Latex software in your computer. You can write it through Overleaf on your browser (it is a free tool). Just upload the latex template to your Overleaf project; it should compile/render the tex file gracefully.
2. The **Java code** implementing design in previous assignment or the one provided. The code should follow the UML class diagram. The code also must address the OOP concepts listed above. The classes and methods must be documented (commented). The code should compile and run properly. The compilation and execution of your code should not rely on any IDE.
    - ***Compilation***. Provide the command for compiling your source code from the command line.
    - ***Running***. Provide the command for running your compiled code from the command line.

You must guarantee that your code is legible, clear, and succinct. Keep in mind that any questionable implementation decision or copy from any source might have a negative effect on your mark. If you still have any questions regarding which file types are acceptable, please inquire prior to submission. Note that it is not the fault of the marker if they are unable to mark your assignment due to submitting an unreasonable or uncommon file format.

Marking Scheme

Marks will be awarded for completeness and demonstration of understanding of the material. It is important that you fully show your knowledge when providing solutions in a concise manner. Quality and conciseness of solutions are considered when awarding marks. Every code added to the originals should be well commented and explicitly indicated in the Java files; lack of clarity may lead you to loose marks, so keep it simple and clear.

Submission

Submission is to be a PDF (description document), and text Java files (Java code). All the submission should be performed electronically through Sakai.
You must guarantee that your code is legible, clear, and succinct. Keep in mind that any questionable implementation decision or copy from any source might have a negative effect on your mark. If you still have any questions regarding which file types are acceptable, please inquire prior to submission. Note that it is not the fault of the marker if they are unable to mark your assignment due to submitting an unreasonable or uncommon file format.

All content files should be organized and put together in a ZIP for the submission through Sakai.

* **Do not forget to include the names and student IDs of group members.**

Late Assignment Policy

A penalty of 25% will be applied on late assignments. Late assignments are accepted until the Late Assignment Date, three days after the Assignment Due Date. No excuses are accepted for missing deadlines. However, deadline extensions may be granted under extenuating circumstances, such as medical or physical conditions; please note that granting the extension is under the instructor's discretion. However, deadline extensions may be granted under extenuating circumstances, such as medical or physical conditions; please note that granting the extension is under the instructor's discretion.

Plagiarism

Students are expected respect academic integrity and deliver evaluation materials that are only produced by themselves. Any copy of content, text or code, from other students, books, web, or any other source is not tolerated. If there is any indication that an activity contains any part copied from any source, a case will be open and brought to a plagiarism committee's attention. In case plagiarism is determined, the activity will be cancelled, and the author(s) will be subject to the university regulations.
For further information on this sensitive subject, please refer to the document below:

`https://brocku.ca/academic-integrity/`