COS 214 Practical Assignment 3

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

- Date Issued: **14 September 2021**
- Date Due: **28 September 2021** at **11:00 am**
- Submission Procedure: **Upload via ClickUP**
- Submission Format: **archive (zip or tar.gz)**

1 Introduction

1.1 Objectives

In this practical you will:

- implement the Composite design pattern in conjunction with the Decorator design pattern
- implement the Observer design pattern
- draw an Object diagram showing a snapshot of the system objects at a given time
- draw a State diagram to highlight the events that trigger changes in the object state
- draw a Sequence diagram to highlight the interaction between objects

1.2 Outcomes

When you have completed this practical you should:

- understand the Composite and Decorator design patterns and how they can be used together
- understand the Observer design pattern
- understand Object, State, and Sequence diagrams and how to draw them to represent the various aspects of the design of your program.

2 Constraints

1. You must complete this assignment individually.
2. You may ask the Teaching Assistants for help but they will not be allowed to give you the solutions.

3 Submission Instructions

You are required to upload all your source files (that is `.h` and `.cpp`), your Makefile, and UML diagrams as individual or a single PDF document and any data files you may have created, in a single archive to ClickUP before the deadline. **Strictly no executables or object files should be uploaded.**

4 Mark Allocation

Task	**Marks**
Race Track Design	20
Race Track Decoration	15
Simulate crashes	15
Main	10
Object Diagram	10
State Diagram	10
Sequence Diagram	10
TOTAL	90

5 Assignment Instructions

A racing company has approached you to design their new race circuit. You will have to use your knowledge about the Composite and Decorator design patterns in order to design their race track.
After you have designed the race track you will use the Observer pattern to simulate what will happen when there is a crash on the track.

Task 1: Race Track Design ..... (20 marks)

A race track consists of various intricate sections that can be combined in order to form a single racetrack. Examples of sections are:

- Straights
- 1/4 Turns
- Chicanes

Notes:

i Putting 4 1/4 turns after one another creates a circular track. You are required to use the Composite design pattern to design a racetrack composed from the various sections available.

ii A 1/4 turn is a 90° turn to the right. Therefore if you need a left turn you can rotate it through 180°.

1.1 Create class `Section` which will be the **component** for both the Composite and the Decorator pattern. It should have the following functions: (5)

- *void print()* - pure virtual function
- *void add(Section*)* - virtual function to add a section to the current end of the track (Note: this is not a pure virtual function)
- *void remove()* - virtual function to remove the last created section of the track (Note: this is not a pure virtual function)

1.2 Create the leaf classes for the Composite pattern. These classes need to **override** the print function to specify what type of section it is. Each class should print out the following: (5)

- `Straight` - prints out "Straight"
- `Turn` - prints out "Right Turn" if a 1/4 turn is rotated with 0°
- `Turn` - prints out "Left Turn" if a 1/4 turn is rotated with 180°
- `Chicane` - prints out "Chicane"

The `Turn` class has a private int variable which gets set through the constructor and is used to check how much the turn is rotated.

1.3 Create the `Track` class that will be used as your composite participant. You have to add the following attributes and functions for the class: (10)

- sections - a vector of Section objects
- void print() - implement the print function
- void add(Section*) - implement the add function to add a new track section
- void remove() - implement the remove function to remove a track section

The **print function** should print out all the sections that are stored in the vector (by calling the print function on each section in the vector - have a look at vector iterators) on a new line so that you can see the logical layout of the track.
You are simply required to print out the details for each section in table-like format as follows (ignore the decorations for now):

Section	Added decorations
Straight	Starting line
Right Turn	Barriers
Straight	
Straight	
Right Turn	Gravel traps
Chicane	Barriers
Right Turn	Barriers
Straight	Pit Stop
Straight	
Right Turn	Gravel Traps

The **add function** has to add a new Section object to the back of the sections vector.

The **remove function** has to remove the last added Section in the sections vector.

Task 2: Race Track Decoration ..... (15 marks)

In addition to building a racetrack you will need to use the Decorator design pattern to add additional detail to your track sections. You have to for instance add the following to your sections:

- Barriers
- Gravel traps
- Pit stop
- Start line (which is also the finish line)

2.1 Create the `Decorator` class. You should add the following attributes and functions: (10)

- section - reference to a Section object
- constructor() - constructor for the `Decorator` class
- void print() - implement the print function
- void add(Section*) - implement the add function
- void remove() - implement the remove function

The **constructor**, initializes the section reference to a null pointer whenever a new instance of the `Decorator` class is made.

The **print function** simply calls the section object's print function.

The **add function**, checks if the section reference is null or not.
If the section reference is null:
    set the section reference to the passed in parameter.
If the section reference is not null:
    call the section reference's add operation with the passed in parameter.

The **remove function**, checks if the section reference is not null, and set it to a null pointer if it is not null.

2.2 Add the additional concrete decorator classes that will all implement the print function to specify what decoration was added to each section of the track. Each class prints out the following: (5)

- `Barrier` - prints out "Barriers"
- `GravelTrap` - prints out "Gravel traps"
- `Pitstop` - prints out "Pit stop"

- `Startline` - prints out "Starting line"

The **print function** first calls the `Decorator`'s print function and then prints out the needed text.

Task 3: Simulate crashes ..... (15 marks)

Your next task is to simulate what will happen when there was a crash on the track. You will need to implement the Observer pattern so that flags are waving when there is a crash and to stop waving them once it is safe to drive.

3.1 You need to update the `Section` class by adding the following attributes and functions: (5)

- observerList - vector of Observer objects
- void attach() - function to add observers to the observerList
- void detach() - function to remove observers from the observerList
- void notify() - function that should notify all the observers in the observerList

The **attach function** should take an Observer object as parameter and add it to the back of the observerList vector.
The **detach function** should take an Observer object as parameter and remove it from the observerList vector.
The **notify function** should call each object in the observerList's update function.

3.2 You need to update the `Track` class by adding the following attributes and functions: (3)

- bool crash - state of the track to signal if there is a crash or not
- bool hasCrash() - function that returns the crash attribute
- void setCrash(bool) - function that will change the state of the track

The **hasCrash function** returns the crash attribute.
The **setCrash function** takes a bool parameter and sets the race tracks state to that value.

3.3 You need to create the `Observer` class with the following attributes and functions: (2)

- *void update()* - pure virtual function
- *void print()* - pure virtual function

3.4 You need to create the `FlagObserver` class with the following attributes and functions: (5)

- raceTrack - reference to a Track object
- waving - bool to signal the state of the observer
- constructor(Track*) - to initialize the attributes of the class
- void update() - implement the update function
- void print() - implement the print function

The **constructor** takes an object of type Track. It sets both the attributes of the class to default values.
The **update function** checks the state of the Track and then sets its own state accordingly.
The **print function** prints out the state of the class to show what it is currently doing.

Task 4: Main ..... (10 marks)

4.1 You have to make a main that will outline what you have done for this practical. In your main you are required to make a race track that consists of various sections where some sections have decorations added to them. You will need to print out your race tack. Next you will need to simulate a crash by changing the race tracks state and changing it back to the original state. (10)
Note: making a main while coding can help you find errors and debug your code.

Task 5: Object Diagram ..... (10 marks)

5.1 You are required to draw an Object diagram for the Composite design pattern. For this you are required to use the `Composite` and `Leaf` classes for the pattern. You are required to use the objects you made in the main.cpp for the diagram. (10)

Task 6: State Diagram ..... (10 marks)

6.1 You are required to draw a State diagram for the Track object in your main. You will need to show all the possible states of the race track object. (10)

Task 7: Sequence Diagram ..... (10 marks)

7.1 You are required to draw a Sequence diagram for the Observer pattern. You have to specify all the important points of interaction between the FlagObserver and the Track objects. (10)