Name Id

advance Programming coursework report

Table of Contents

[Section 1 1](#_Toc5368973)

[Section 2 2](#_Toc5368974)

[Section 3 3](#_Toc5368975)

[3.1 Bugs - List plus brief description 3](#_Toc5368976)

[3.2 Weaknesses - List plus brief description 4](#_Toc5368977)

[Section 4 4](#_Toc5368978)

[Level 1 4](#_Toc5368979)

[Level 2 4](#_Toc5368980)

[a. UML class diagram for the inheritance hierarchy 4](#_Toc5368981)

[b. UML class diagram for the interfaces 5](#_Toc5368982)

[c. Brief Reflection 6](#_Toc5368983)

[Level 3 6](#_Toc5368984)

[Level 4 6](#_Toc5368985)

[Pattern 1 6](#_Toc5368986)

[Pattern 2 6](#_Toc5368987)

[Level 5 7](#_Toc5368988)

[Level 6 7](#_Toc5368989)

[Section 5 7](#_Toc5368990)

[Section 6 7](#_Toc5368991)

[Section 7 7](#_Toc5368992)

[Section 8 7](#_Toc5368993)

# Section 1

The application that we designed is a dashboard for a small air craft, which is fully working. We managed to go through all the levels, and, according to our own view, we succeeded in implementing all the requirements and made a functional software which delivers what is expected to do. We did not approach it from the first level to the last one in the given order, Level 5 and Level 2 were attempted in the end.

|  |  |
| --- | --- |
| Which type of dashboard application did you implement? | Train small air craft |
| 1.1 Circle the parts of the coursework you have fully completed and are fully working | 1a 1b 2a 2bi 2bii 3  4a 4bi 4bii 5a 5b 6a 6b |

# Section 2

As it can be noticed in our UML Class Diagram, Dashboard class is in the centre and it represents all the classes it interacts with, there are few aggregation relationships and plenty of simple associations with other classes that provide different functionality. The classes that came from the DashboardDemoProject we were given as a start were not included in the diagram as they were not created by us and we decided not to add them. There are 2 diagrams for this section, as it felt right to add one with class names only (Figure 1) so that the relationships between classes are easier to notice, and the second ones is a more in-detail diagram (Figure 2).

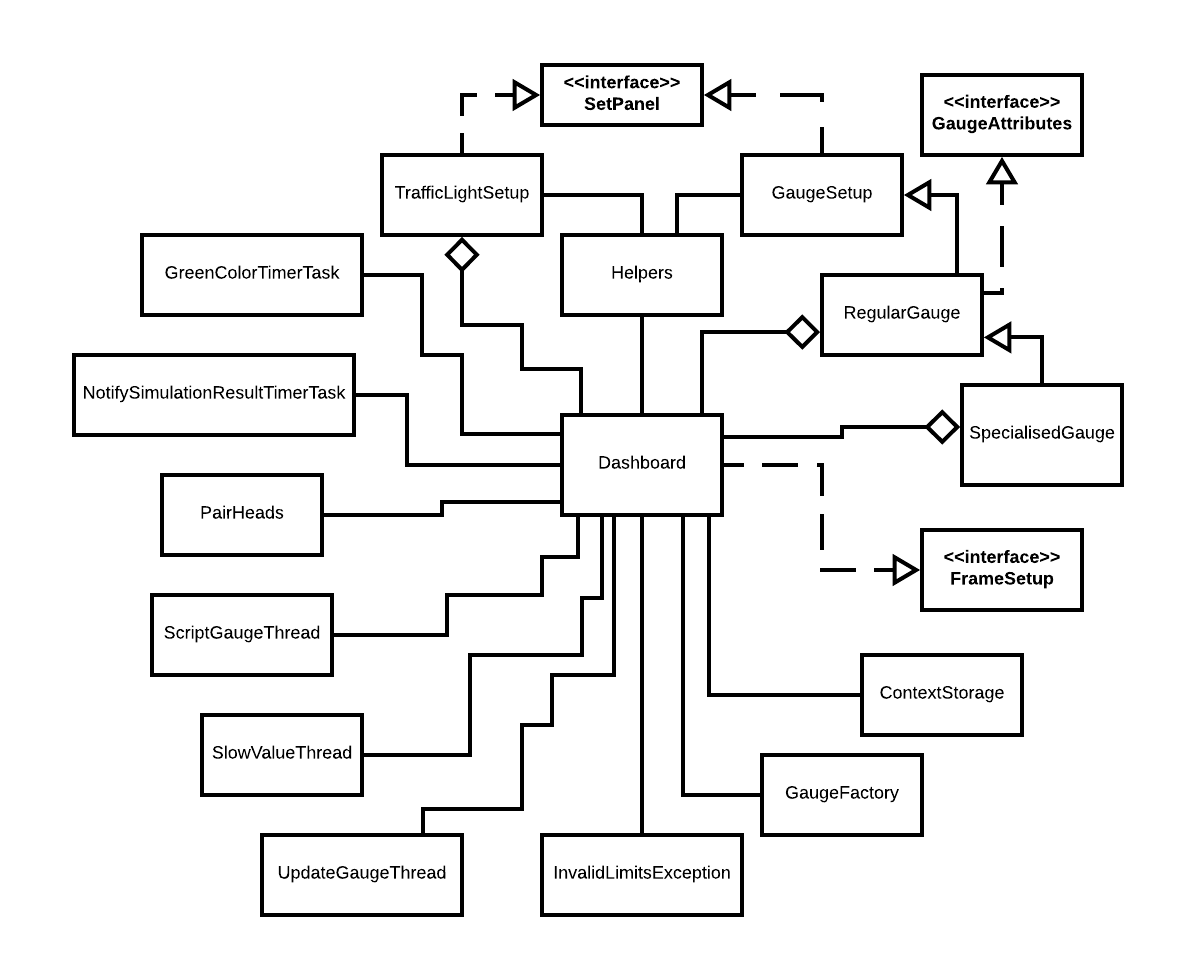


Figure 1 Simple UML Class Diagram

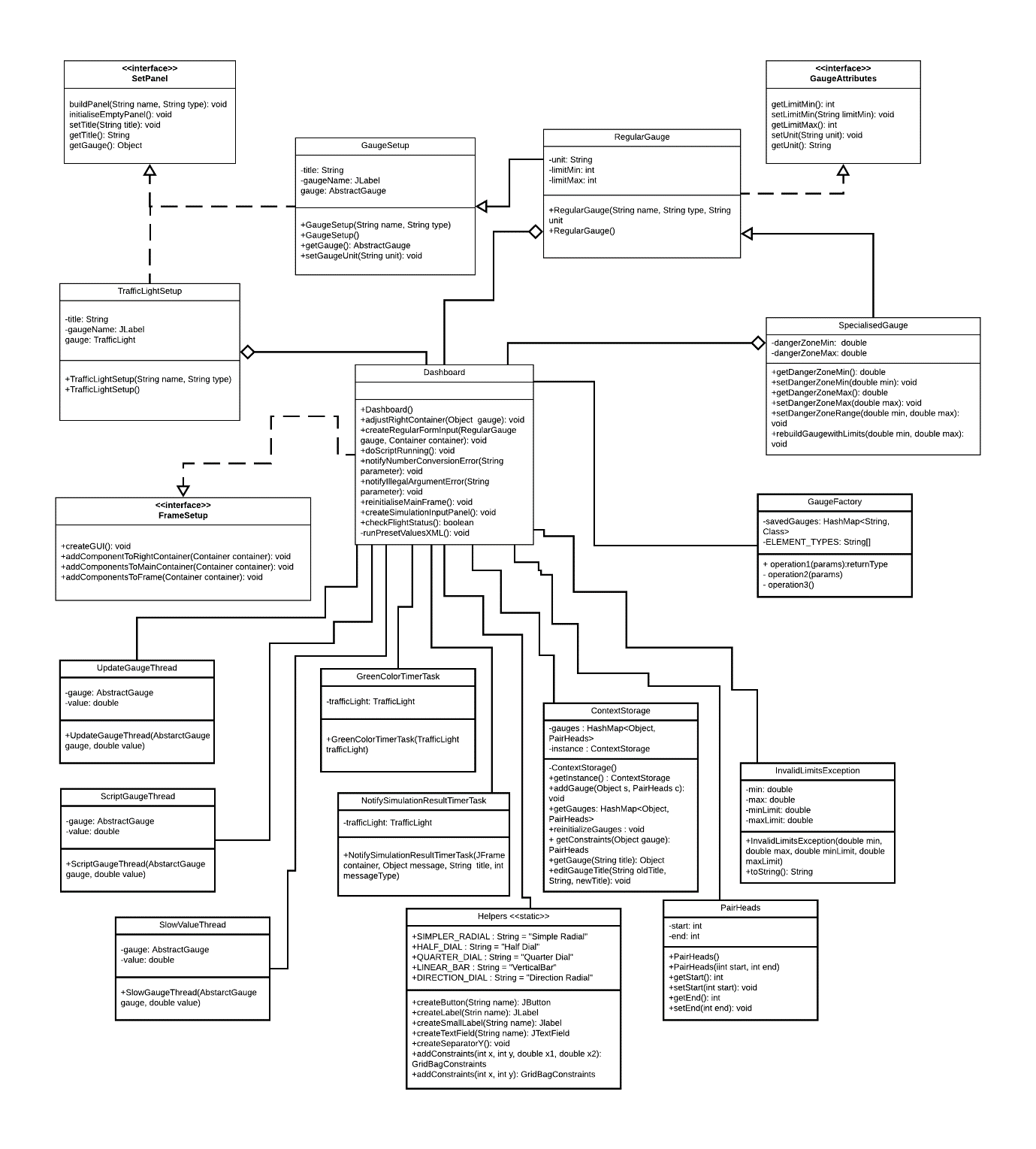


Figure 2 Detailed UML Class Diagrams

# Section 3

## 3.1 Bugs - List plus brief description

The graphical user interface has a strange behaviour once a user clicks the second time on the same gauge selected previous that panel that allows the user to edit different parameters of the dial disappears and only one text field remains, the one to change the value where the dial currently points at.

## 3.2 Weaknesses - List plus brief description

For the simulation game, when user inputs more than the maximum value, or less than minimum, there is no check to verify if the input exceeds the given range and technically user can add any value, also if overflow occurs, and the inserted number is larger than the variable type, no warning is given and an overflow value will be stored, which will be incorrect.

# Section 4

## Level 1

The code that we were provided with seemed to be simple and easy to use as well, however the solution we provided has more methods that simplify the creation of some of the GUI components, including containers with multiple text fields, labels and buttons. Additionally, it makes use of SteelSeries libraries which comes with various types of gauges, thus no need for writing the code to create a dial manually. Moreover, threading improves the overall performance of the application, which prevents it from freezing when executing lengthy tasks.

## Level 2

### UML class diagram for the inheritance hierarchy

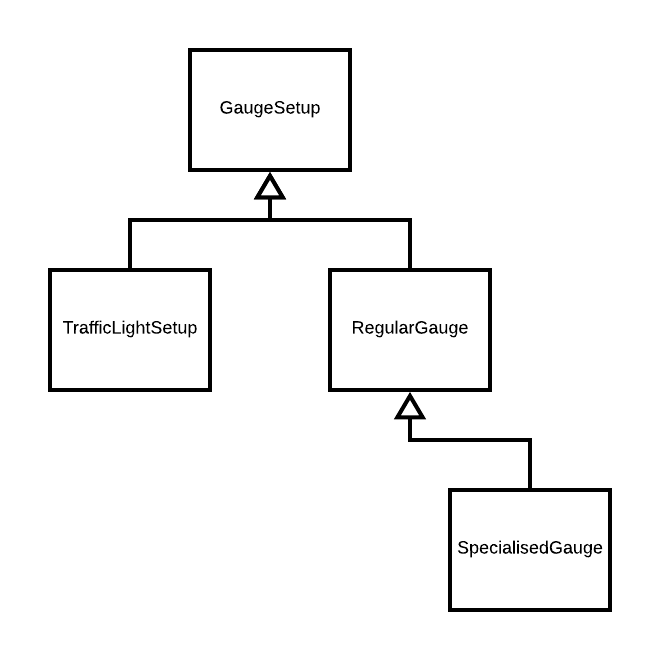


Figure 3 UML Inheritance Hierarchy

### UML class diagram for the interfaces

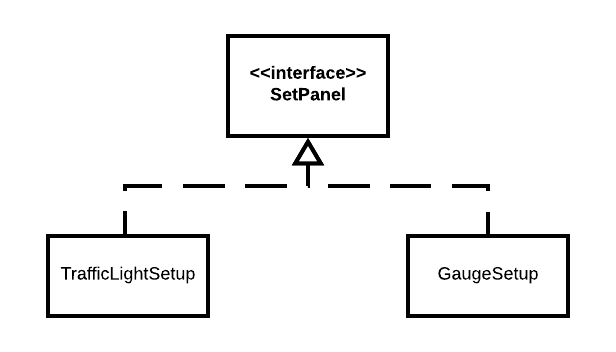


Figure 4 UML SetPanel interface

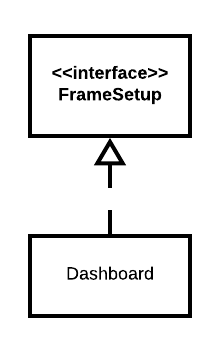


Figure 5 UML FrameSetup interface

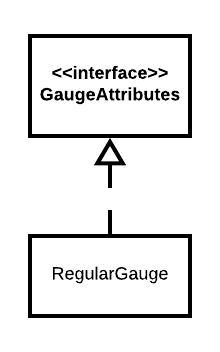


Figure 6UML GaugeAttributes interface

### Brief Reflection

The object-oriented features implemented for this level make the code cleaner and easier to maintain b3cause we are making use of inheritance and interfaces which allows a programmer that want to add new fields or methods to a super class then all the other classes will automatically have the same properties if the right visibility is set, same applies for interfaces, where no matter of the parameters of the different classes, functions will return what they are meant to.

## Level 3

For Junit testing the following classes where tested:

* GaugeSetup – 12 test cases;
* RegularGauge – 8 test cases;

## Level 4

### Pattern 1

Name: Singleton

Implementing the Singleton pattern adds limitations to the number of instances of the class that can exist. Additionally, since it has the same values throughout the app life-cycle, accessing it from different classes should return the same result, thus storing information could be considered, such as user settings.

### Pattern 2

Name: Factory

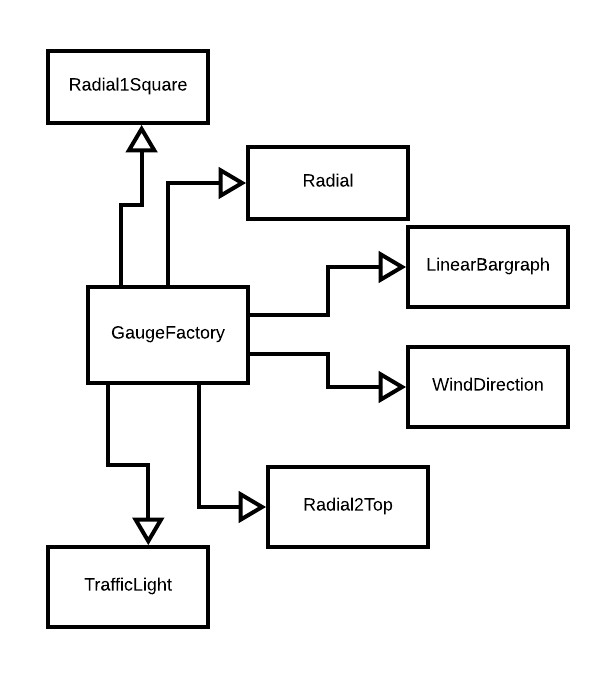


Figure 7 Factory Pattern

In general, the factory pattern is used to create instances of objects when there is uncertainty regarding the type of the object that should be instantiated, so instead of using a specific type, through casting the desired results can be achieved. In future, if the programmer decides to allow the user to dynamically instantiate objects at running time, this feature will prove very useful, and, in case more variety is to be added in matter of the dials that can be generated it will require adding only two lines of code for each new component.

### Pattern 3

Name: Servant

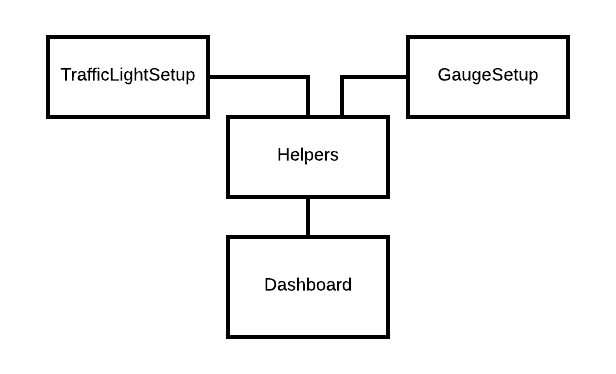


Figure 8 UML Helpers class

The Servant Pattern can provide a programmer with a lot of functionality if same functionality is needed in classes that have nothing in common, also constants can be kept inside this class. The servant does not need to know anything about the class that is using its functions, only the serviced class needs to know exactly what it needs from the Servant and asked for it. It is easy to maintain, and a programmer should not find it difficult to add extra functionality.

## Level 5

Class name: Converter

The created JavaBean is meant to serve as a converter for anything, user can set a scale so that every time calculus is required, the solution is based on the scale that was set.

There are four properties in total (4 getters, 4 setters), which allows changing of the units displayed next to the input fields, the name of the converter and the scale. The icon of our bean is also displayed in the pallets.

## Level 6

### Dials indicators move smoothly

In order to make the indicators of the dials move smoothly threads were required. Classes that extend the default Thread API class or implement Runnable interface were created so that the desired results can be achieved. Also, depending on the use of the Thread, thread-safety was added either on the instance of the object or on a given class. For example, class UpdateGaugeThread is meant to allow the user to set the value for a gauge, and thread safety is set for the entire class, so that no other dial will have its value updated until the one currently running finished the processing.

Weaknesses:

* User cannot update the values of other dials until the current one does not finish;
* A thread is sent to sleep in a for loop which might result in an InterruptedException and no proper feedback to the user of such error was implemented;

### Simple Game

The game is meant to simulate a take off and a landing of an airplane, although is relatively short. If the user selects the button “Play simulation”, a panel appears, and the user can fill in with values. The simulation is implemented by reading from a file predefined values, so it always runs the same way. The movements on the dials are smooth and based on given input an algorithm is used to calculate whether the flight will be successful, or the plane will crash.

Weaknesses:

* Since the simulation runs from a script file, it is always the same effects taking place on dials;
* There is no limitation for the number that can be inputted in a text field, thus if user decides to add a very large number, calculation errors may occur;

# Section 5

# Section 6

# Section 7

# Section 8