PA1482 Assignment Descriptions

Mikael Svahnberg[*]

2024-08-23

Contents

1 Introduction

This document describes the assignment submissions in the course PA1482 Applied Object Oriented Design.

The syllabus lists a single block "Laboratory Session" of 5 ECTS credits. Within the scope of the course, this is divided into submissions as follows:

1. **Early Design Sketch** (Feedback only). The purpose of this submission is to ensure that you are somewhat on the right track for the remaining assignments, and to provide you with early feedback to guide your work.
2. **Software Design**. In this assignment you create, describe, and reflect upon a software design for a given system as described below.
3. **Implementation**. In this assignment you implement the system according to your software design. This assignment is also further described below.

2 Assignment at a Glance: Tornsvala

The system you will be working with is a flight tracker software called *Tornsvala*. This system is intended for aircraft spotters to keep track of aircraft traffic within their region, and to take action depending on the type of aircraft.

[*]Mikael.Svahnberg@bth.se

An aircraft is added to the system with a tail-code (e.g. *G-CFGJ*) and a type. Depending on the *type*, the system decides how to keep track of it. For example:

Type	Action
Avro Lancaster	Do Nothing
Bristol Blenheim	Do Nothing
DH 98 Mosquito	Log flight code
Hawker Hurricane	Log flight code
Supermarine Spitfire	Log flight code
Messerschmitt BF109	Notify Fighter Command
Focke-Wulf FW190	Notify Fighter Command AND Log flight code
Dornier Do-217	Track route and predict destination
Heinkel HE-177	Track route and predict destination
Junker Ju-87 Stuka	Track route and predict destination AND Notify Fighter Command

Notes:

- As can be seen, some aircraft require a combintion of actions to be taken
- This is an example list of aircraft types; it shall be possible to add new types.
- This is a starting list of possible actions; it shall be possible to add new actions.

A typical session when using the system is as follows (a graphical user interface may also exist):

```
> seen G-CFGJ
This is a new aircraft.
Please add the type to your observation: "seen G-CFGJ «aircraft-type»"

> types
Possible types of aircraft are:
- (short name -- full name)
- Lancaster : Avro Lancaster
- Blenheim  : Bristol Blenheim
- Mosquito  : DH98 Mosquito
- Hurricane : Hawker Hurricane
- Spitfire  : Supermarine Spitfire
- Messershmitt : Messershmitt BF109
- Focke-Wulf : Focke-Wulf FW190
- Dornier   : Dornier DO-217
- Henkel    : Heinkel HE-177
- Junker    : Junker JU-87 Stuka

> seen G-CFGJ Spitfire
Added Supermarine Spitfire G-CFGJ to Collection
Logging G-CFGJ at August 26 14:32...
```

```
> info G-CFGJ
Supermarine Spitfire G-CFGJ
Last seen August 26 14:32

> seen D-FWME
Existing Aircraft Messerschmitt BG109 D-FWME
Alert! Notifying Fighter Command!

> seen RI-JK
Existing Aircraft Junker JU-97 Stuka RI-JK
Tracking route and predicting destination...
Alert! Notifying Fighter Command!
```

3 Early Design Sketch

Note: This submission is only intended for you to get early feedback to ensure that you are somewhat on the right track for the remaining assignments.

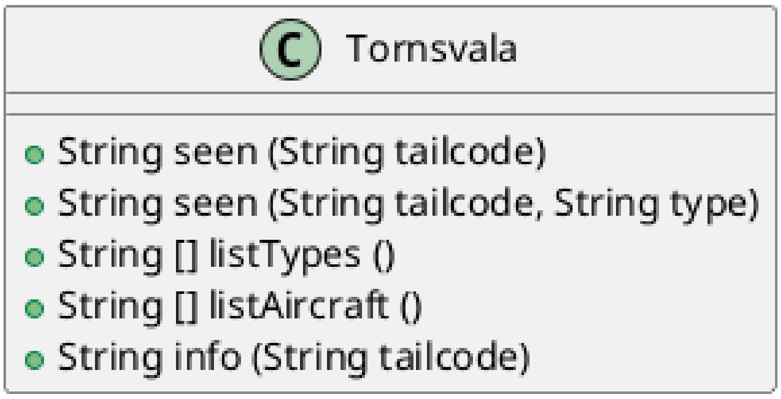
In this submission you will start reasoning about and using the basic design principles that have been introduced in the lectures up to this point. Specifically, the following principles have been covered:

- Program objects represent real world entities
- Low Coupling
- High Cohesion
- Encapsulation

Please consider these design principles and answer the following questions:

1. How should the system represent a single real world aircraft? Why (according to which principle or principles)?
2. What data are kept for each aircraft? Which class is best suited to contain this data? Why (according to which principle or principles)?
3. How should the system represent a collection of aircraft? Why (according to which principle or principles)?
4. How may you implement the system so that each aircraft may have different actions when seen? Reason about possible solutions and their strengths and weaknesses. Try to also think about this in terms of design principles.

Using your answers to these questions, please create a *class diagram* for the system. Try to make the class diagram consistent with your answers. Do not forget to add methods, attributes, and the necessary associations between the classes. Also, assume that there is one class "Tornsvala" that is the entry point for accessing the system, containing at least the following methods (more methods may be needed, e.g. to add and edit the list of aircraft types):


Tornsvala
String seen (String tailcode)
String seen (String tailcode, String type)
String [] listTypes ()
String [] listAircraft ()
String info (String tailcode)

Submit your answers to the questions and your class diagram to Canvas.

4 Software Design

In this assignment you continue working with your design and ensure that you make good use of *design patterns* to further improve your design. In addition to the previous design principles, you have now also been introduced to:

- Localise Change (High Cohesion)
- Favour Composition over Inheritance
- Open-Closed Principle

With this new knowledge, revisit your previous design and answer the following questions:

1. Can you encapsulate the different *actions* (do nothing, log, notify, and track route) using a design pattern? Which? How? Why is this good? How does this compare to your earlier solution?
2. Are there any design patterns that can help you deal with *combinations of actions*? Which? How?
3. Which class is best suited to ensure that the list of seen aircraft is not lost when the system is restarted? Will you need to create a new class for this? Why/Why not?

Please *update your class diagram* to make use of the design patterns and classes you identified and discussed in these questions, and any additional classes or updates that you deem necessary.

Some additional design patterns were introduced in lectures too close to the submission of this assignment, but you are welcome to reason about additional design patterns that you think may improve your design further. In particular, the creation of aircraft objects such that they get the desired behaviour depending on the type of aircraft can be encapsulated to a single place with the help of a design pattern, and the answer to question 3 above is actually an application of another design pattern.

Submit your answers to the questions and your class diagram to Canvas.

4.1 Assessment

The follwing is part of the assessment, and is weighed together to a grade on the assignment:

Completeness The extent to which all requested aspects of the assignment are accounted for.

- **3 points** Everything asked for in the assignment is delivered.
- **0 points** Some of the deliverables, as requested by the assignment description, are missing.

UML Notation Adherence to the standard UML notation.

- **3 points** All UML design elements are correctly used and described.
- **2 points** Most UML design elements are mostly used correctly and described.
- **1 points** (intermediate)
- **0 points** There are significant flaws with how UML design elements are used. Standard UML notation is not followed.

Workable Design The produced design artefact(s) will, if implemented as described, result in a workable system that addresses the right requirement(s).

- **5 points** This design can be directly implemented and will result in a workable system which is easy to maintain and extend. All relevant requirements are addressed.
- **4 points** (intermediate)
- **3 points** This design is likely to work with only minor changes. Most relevant requirements are addressed.
- **2 points** Significant changes are necessary to the design before it can be used to implement a working system. Changes are necessary before this design is able to address all the relevant requirements.
- **1 points** (intermediate)
- **0 points** Extensive work on the design is necessary before it can be used to implement a working system. Relevant requirements are not addressed.

Use and Description of Design Patterns The extent to which Design patterns are correctly used AND documented.

- **5 points** The right design patterns are used correctly in all relevant places. Each use is identified and correctly described.
- **4 points** (intermediate)
- **3 points** The right design patterns are used correctly in most relevant places. Each use is identified and correctly described.
- **2 points** The right design patterns are used mostly correct in most relevant places. Each use is identified and most descriptions are correct.

1 points (intermediate)

0 points The wrong design patterns are used, or no design patterns are used. Obvious applications of design patterns are missed. Usage of Design Patterns are not identified and/or documented.

5 Implementation

Impement your system according to your class diagram in Java or C++. Pay special attention to:

- Structure your classes into files, and your files into relevant packages/namespaces/file system directories according to your design.
- Use the appropriate type of reference (e.g. pointers) to address the objects.
- Use appropriate data structures for your collections. Use the data structures provided in the standard libraries for your programming language.
- Avoid any input/output to and from the user in all classes except those that actually have that responsibility.

Please document any deviations (new/changed/deleted classes, methods, attributes) from your design. Why are you introducing these deviations? Some re-naming of methods and changes to the parameter lists is ok, but any major deviations MUST be documented and explained. When asked, you are expected to be able to show the implementation of a method/class from the class diagrams.

You may get a starting point for your implementation from the following repository: `https://codeberg.org/mickesv/Tornsvala.git` . This is written in Java and provides two means of accessing the Tornsvala interface, i.e. one REPL and one GUI. You will notice that it is just as easy to instead attach a test harness to the same API endpoints. This is written in Java, but is easily translated to c++ if you prefer.

If you are unsure whether your class diagram is implementable, please contact the course manager to get a ready-made class diagram to use as a basis.

Submit your answers to the questions and your class diagram to Canvas.

5.1 Assessment

The follwing is part of the assessment, and is weighed together to a grade on the assignment:

Completeness The extent to which all requested aspects of the assignment are accounted for.

3 points Everything asked for in the assignment is delivered.

0 points Some of the deliverables, as requested by the assignment description, are missing.

Working Software The produced artefacts work or will lead to a working system.

3 points All produced artefacts work as intended. All relevant requirements are accounted for.

2 points Most produced artefacts work as intended, only a few of little importance to the overall system do not work fully. Most relevant requirements are accounted for.

1 points Some of the produced artefacts of lesser significance do not work as intended.

0 points Many of the produced artefacts do not work. Many significant requirements are not implemented.

Adherence to Design The extent to which the produced software artefacts implement the given software design.

5 Pts There is a clear and complete mapping between the design and the implementation. All deviations are documented.

4 Pts (intermediate)

3 Pts There is a clear mapping between the most of the design and the most of the implementation. Most deviations are documented.

2 Pts There is a mapping between the most of the design and the most of the implementation. Some deviations are documented.

1 Pts (intermediate)

0 Pts There are obvious gaps in the implementation where the design is not implemented as described. Deviations are not documented.