Software Engineering 2  
DEAD REPORT

Deadline Report

|  |  |
| --- | --- |
| **Team number:** | 0310 |

|  |  |
| --- | --- |
| **Team member 1** | |
| **Name:** | Christian Orlowski |
| **Student ID:** | 12024267 |
| **E-mail address:** | a12024267@unet.univie.ac.at |

|  |  |
| --- | --- |
| **Team member 2** | |
| **Name:** | Jonas Michael Speiser |
| **Student ID:** | 12012545 |
| **E-mail address:** | a12012545@unet.univie.ac.at |

|  |  |
| --- | --- |
| **Team member 3** | |
| **Name:** | Sandra Tomeschek |
| **Student ID:** | 01504450 |
| **E-mail address:** | a01504450@unet.univie.ac.at |

|  |  |
| --- | --- |
| **Team member 4** | |
| **Name:** | Leyla Durdyyeva |
| **Student ID:** | 01576824 |
| **E-mail address:** | a01576824@unet.univie.ac.at |

|  |  |
| --- | --- |
| **Team member 5** | |
| **Name:** | Felix Achim Hubert Lindau |
| **Student ID:** | 11948883 |
| **E-mail address:** | a11948883@unet.univie.ac.at |

Dear students,

This template document suggests an outline for the required contents of the DEAD report. The included descriptions and examples are supposed to help you write a clear report that documents and presents your actual solution well. Please remove this additional text (similarily written as this text) and exemplary material before you submit your report!

The actual DEAD report document can be based on this template, or can be written using an arbitrary text editing program such as Latex, LibreOffice and the like, as long as the required material (described in the assignment document) is contained.

In general, write the report in such a manner as to provide all information to a third party who is not involved in the design and development and unfamiliar with the exact tasks of the semester project assignment. Think of this third party as a company that roughly wants to offer a software product as described in the assignment document and commissioned your team to start this software engineering project and produce a viable project which follows best practices, is of high quality, and is worth to be funded further after the DEAD deadline.

**NOTE: In case you copy/paste material from the SUPD report, please make sure that it is updated and matches the final project state and results!**

The SE2 Team.

# Final Design

## Design Approach and Overview

Describe your design approach and how you arrived at your **final solution**. Typically you will start with an initial solution that you refine in an iterative way by means of re-factorization. Use at least class diagrams and the used technology stack for documenting:

* major design decisions (e.g., design alternatives). Discuss and explain your design decisions also regarding the overall class layout. Possibly include design descriptions/class diagrams showing (a part of) the design before a refactorization step and/or alternatively considered (parts of) designs.
* a design overview of the final state of your solution at DEAD. Note that used design patterns should be well visible in this design overview (e.g., in the class diagrams). Regarding design patterns you may cross-reference to Section 1.3 for the details.

Carefully check all your UML diagrams for syntactical and semantical correctness!

### Class Diagrams

Class diagrams (for example, see Figure 1) describe the structure of a design. Try to omitt unnecessary detail (e.g., getter and setter operations), and focus on giving a good structural overview. You may also use multiple levels of details (zoom levels), or present multiple class diagrams showing parts of the overall solution that are (mostly) independent from each other.

Ein Bild, das Text, Karte enthält.

Automatisch generierte Beschreibung  
Figure 1: Sample Class Diagram. TODO: remove in final submission!

### Technology Stack

Briefly describe the frameworks and/or libraries your solution is using or going to use (name, website link, version, 2-3 sentences describing the purpose of the framework/library in your solution). Also document considered alternatives and how you arrived at the decision to select one of the alternatives and disregarded the other(s). For your decision process you might want to pick a set of criteria (e.g., team experience, available documentation, convenience features, …).

## Major Changes Compared to SUPD

Give a list of major changes of your software project compared to the SUPD milestone (briefly summarize and possibly reference the SUPD report contents and/or source code paths). In case you have been given recommendations in the SUPD feedback, please briefly report on how you have implemented them.

Klare Struktur des MMVV Models🡪 Beispiele aus den Klassen

Data Binding und “leere” View Klasse

## Design Patterns

For each different design pattern make a subsection where you first briefly summarize the used pattern in general (in your own words!), then discuss how you applied the pattern in your solution. For each instance of the pattern in your solution

* give a (textual) argument how the pattern instance relates to one of the functional requirements,
* explain the particular problem that the pattern should solve,
* provide class diagrams and possibly other UML diagrams to give a high-level structural and possibly also behavioral description, and
* relevant code snippets from your implementation showing the implemented pattern

Focus on a detailed description for the first two pattern occurrences, but at least mention the remaining pattern instances along with references to your source code.

### Strategy Pattern (copied from SUPD)

The Strategy Pattern is useful when there are multiple algorithms which solve a problem, and they can be used interchangeably, according to the concrete context. It is an alternative to implementing behavioural logic in subclasses of a Context, which has the benefit of separating business logic from Context state and thus making the code easier to read and maintain. Moreover, conditional statements for selecting the behaviour are also avoided thanks to this pattern.

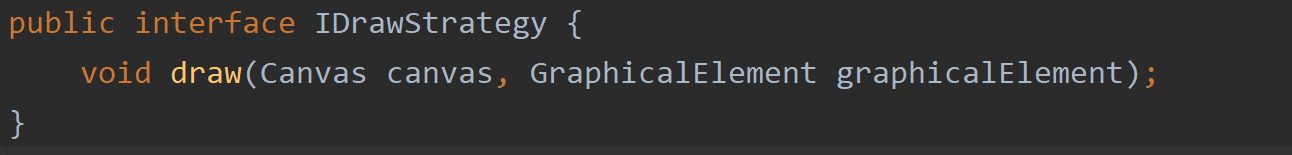
In our project, the Strategy Pattern turned out useful to avoid adding logic for drawing on the Canvas directly in our concrete subclasses of GraphicalElement. The logic for drawing is different for each graphical element, different Strategies have been implemented.

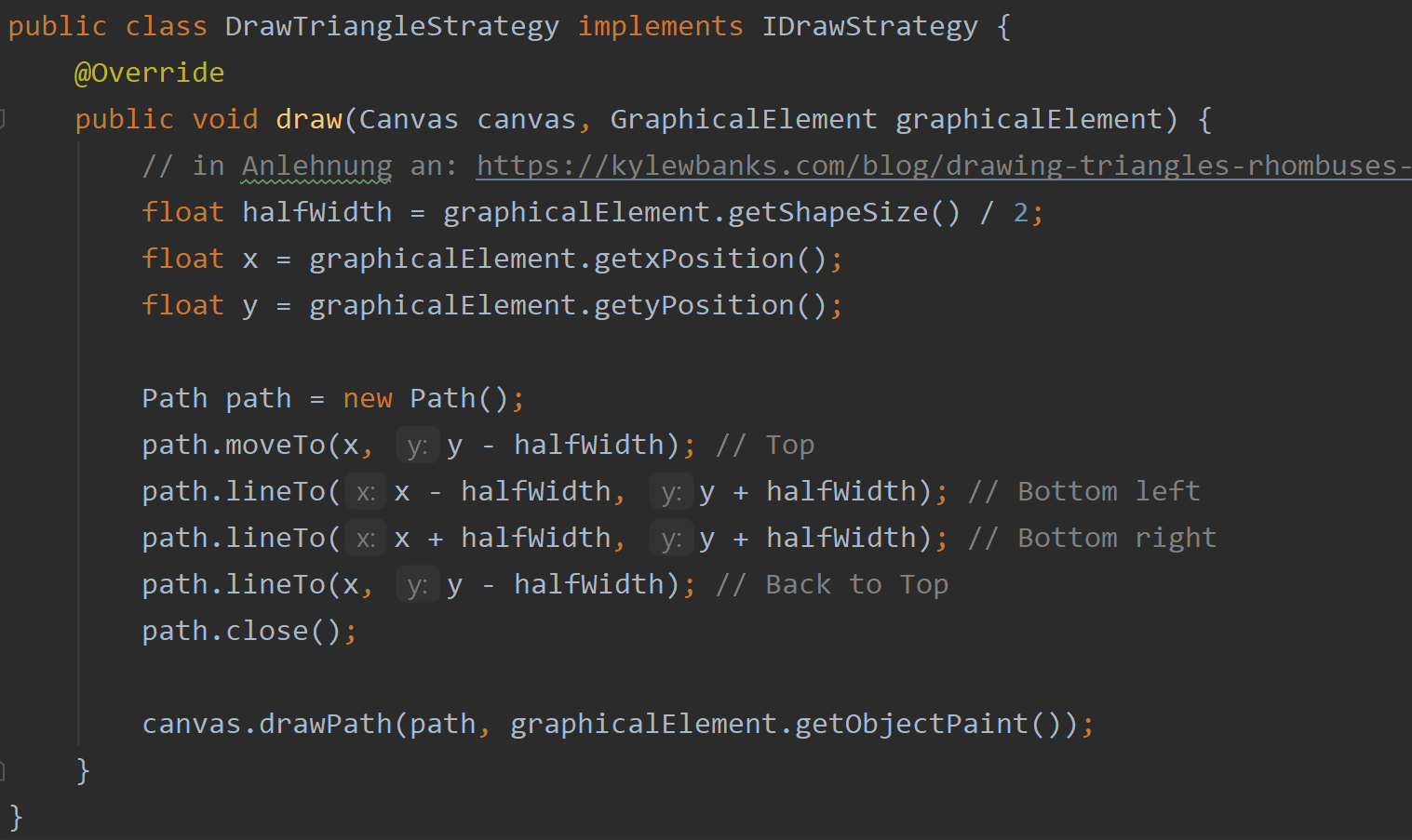
This design pattern relates to the Functional Requirements FR1-FR4, requiring new Graphical Elements to be created and displayed (drawn) on the View. The Strategy pattern is executed when the whole View is being drawn again, by going through each graphical element and invoking the corresponding Strategy object. The drawing is performed by the ViewModel and used to represent the data on the View.

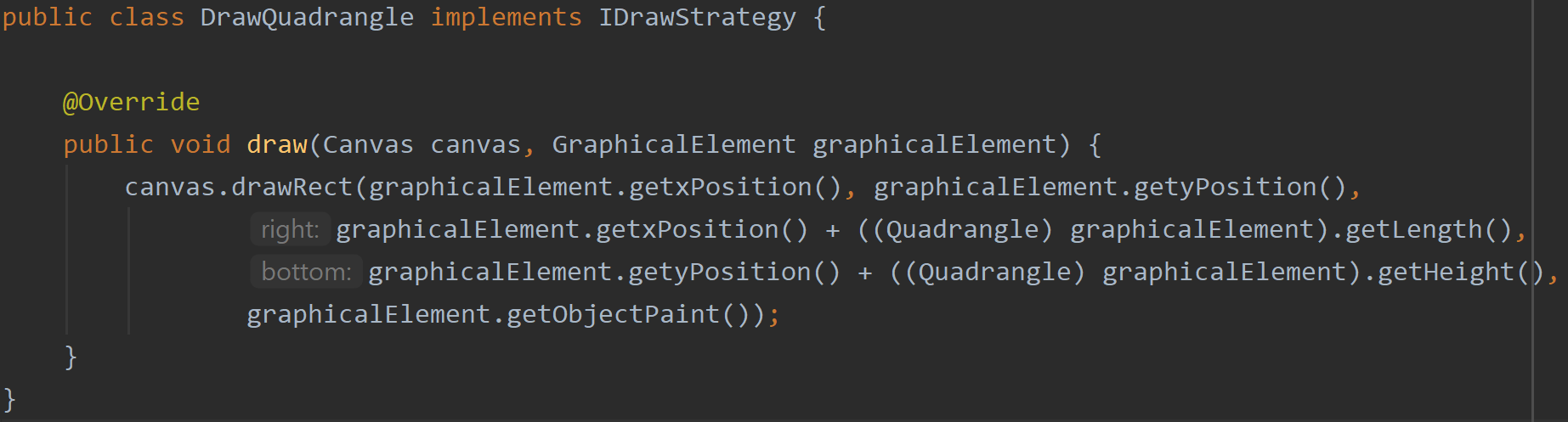
Diagram

Description automatically generated

Code snippet: (IDrawStrategy.java):







### Factory Pattern (Copied from SUBD)

The Factory Method Pattern is useful for delegating the creation of new objects to a dedicated class. This is useful in order to better split the code and avoid bugs. The required attributes and constructor of a concrete object might also change over time and having the creation logic in one determined class makes it easier to address the changes.

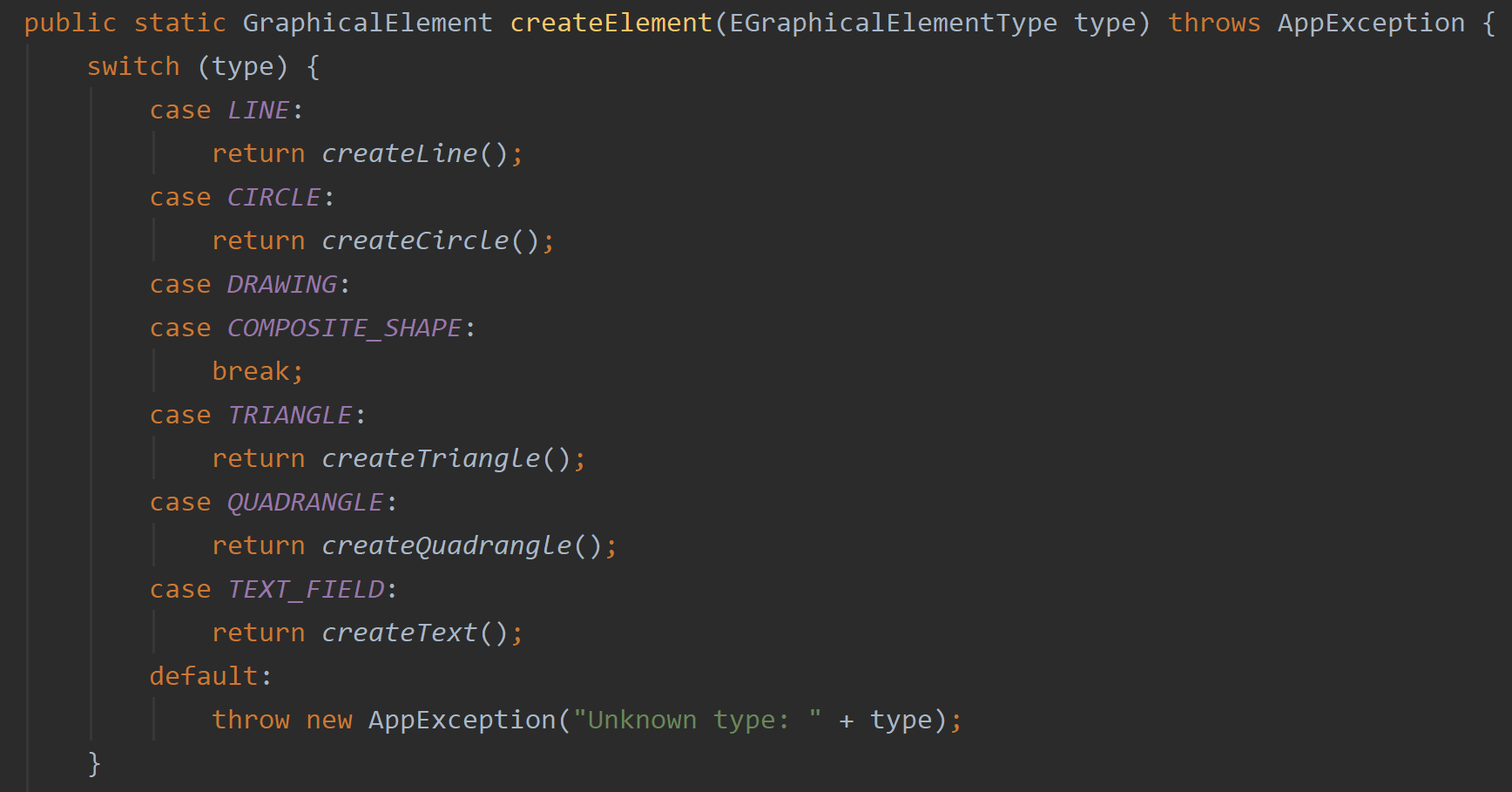
In our project we needed to implement the Factory Method pattern to create objects of the GraphicalElement type from our Model. These objects are being later displayed on the View.

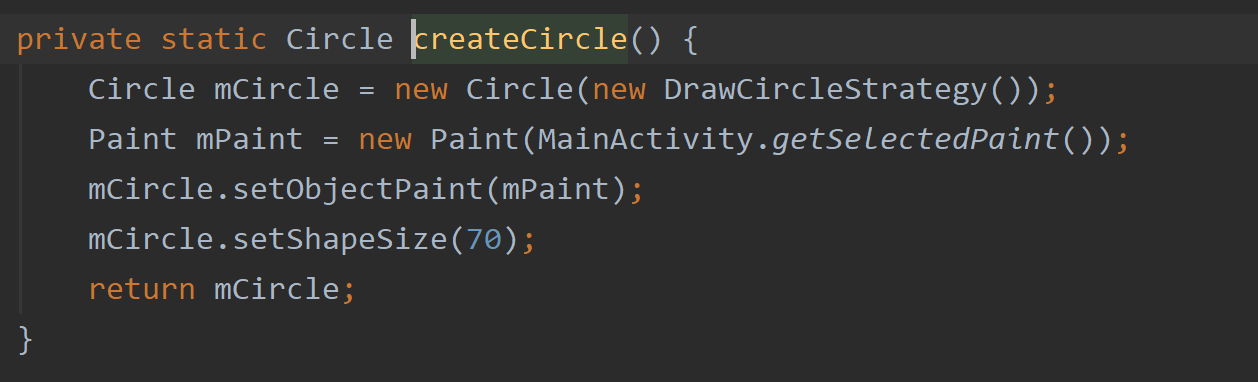
This design pattern relates to the Functional Requirements FR1-FR4, requiring new Graphical Elements to be created and displayed on the View. The creation happens after the user selects a graphical element. The creation is an intermediary step between the data saved in the Model and the representation of the data in the View.

Diagram

Description automatically generated

Code snippet (GraphicalElementFactory.java):





### Facade Pattern

Used to simply the communication with the graphical element creation factory

Facade zwischen view und viewmodel / viewmodel und model

### Decorator Pattern

Decorate our graphical elements with additional colour. We have a default colour and decorators for additional colours.

Or Decorate our text objects with additional styles / fonts

### Iterator

Create an iterator in order to iterate over the graphical elemnt list and the layers.

### Adapter

Sizebar should affect text size and graphical element size (round objects wird radius and square objects)

### Composite

combined graphical elements should be editable in the same way as single graphical elements.

**ALTERNATIVEN:**

### Abstract Factory Pattern

Used for creating two different patterns: One for filled objects and one for not filled objects (might not be necessary)

### Observer Pattern

Observer: daten von view Model auf view udn von model auf view model

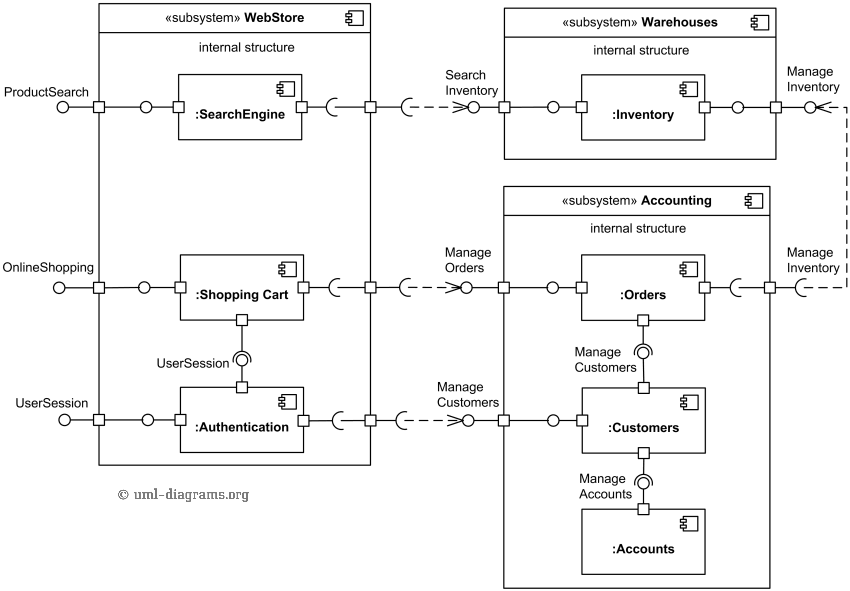
### Template Methode Pattern

Import and export different kinds of datatypes

# Implementation

## Overview of Main Modules and Components

Give a brief overview of the implementation of your design. Besides a textual description you may use UML component diagrams (see for example, Figure 2).

  
Figure 2: Sample Component Diagram from : <https://www.uml-diagrams.org/examples/online-shopping-uml-component-diagram-example.html> TODO: remove in final submission!

## Coding Practices

Discuss how and to what extent you have considered coding practice in your project implementation. Discuss and show examples from your code.

Naming: (Be intention revealing ♣ Avoid disinformation ♣ Make meaningful distinctions ♣ Be pronounceable ♣ Be searchable)

Comments: as discussed in qr1 and (internal documentation for anything that is not readily obvious ♣ Use comments to explain the intent of the code ♣ Comments can tell you why the code works / is written the way it is)

Form: Establish a standard size for indent ♣ Align open and close braces vertically ♣ Indent code along the lines of logical construction ♣ Use white space to provide organizational clues to source code

Creating Operations (Methodes?): The single most important reason to create an operation (method or function) is to reduce a program’s complexity ♣ Create an operation to hide information so that you won’t need to think about it

## Defensive Programming

Discuss how and to what extent you have considered defensive programming in your project implementation. Discuss and show examples from your code.

Purpose in general:♣ Leave in code that checks for important errors ♣ Remove code that checks for trivial errors ♣ Remove code that results in hard crashes ♣ Leave in code that helps the program crash gracefully ♣ Log errors for your technical support personnel ♣ Make sure that the error messages you leave in are friendly

Assertions: ( Assertions are used to handle errors that should never occur in the code)

Errorhanding techniques (Return a neutral value, Substitute the next piece of valid data., Return the same answer as the previous time, Substitute the closest legal value🡪 Log a warning message to a file ♣ Return an error code ♣ Call an error-processing routine/object ♣ Display an error message wherever the error is encountered ♣ Handle the error in whatever way works best locally ♣ Shut down

Exceptions: errors that can be ignored by the system

# Software Quality

## Code Metrics

Provide information on at least the number of packages, lines of code, comment lines of code, number of classes and code bugs with regard to your current state of the implementation. It is recommendable to use a static code analysis tool (e.g. SpotBugs). Compare the metric results with your SUPD results and discuss your findings.

## Testcases for Functional Requirements

Discuss for each functional requirement how one or more of your testcases cover/s it. In case of limitations document them!

## Quality Requirements Coverage

Discuss for each quality requirement what activties reported and/or what artifacts implemented in your deliverables cover it. Only summarize briefly and provide references to report sections, or source code files.

**QR1 Comment your code and provide code documentation in an appropriate manner:**

* Code snipplets with comments, Javadoc. Welche Methoden oder Klassen wollen wir mit JavaDoc kommentieren?

**QR2 Your implementation must be in compliance with a style guide:**

* Reasons for using a **style guide**: Be consistent within the team, outer teams or distributions are not important to us: “Within this article, we have briefly illustrated how to easily and quickly integrate the Google Java Style Guide in a Java project. Even though this convention is widely spread it’s not the only candidate, you may also consider the [Sun Code Conventions](https://checkstyle.org/styleguides/sun-code-conventions-19990420/CodeConvTOC.doc.html) and [Checkstyle](https://checkstyle.sourceforge.io/), even though it’s antiquated, verbose and quite difficult to integrate.”

<https://medium.com/@alexprut/integrate-google-java-style-guide-in-a-java-project-567abb6d7987>

https://github.com/autyzm-pg/friendly-plans/wiki/How-to-install-the-Google-Style-Guide-settings-in-Android-Studio

**QR3 Apply common coding practices. 🡪 Summarize chapter 2.2**

**QR4 Apply defensive programming**. 🡪 Summarize chapter 2.3

**QR5 Apply key design principles**

Abstraction means to focus on the essential features of a design element

Modularity is the property of a system that has been decomposed into a set of cohesive and loosely coupled modules

Information Hiding / Encapsulation: as follows: “Every module … is characterized by its knowledge of a design decision which it hides from all others

Separation of concerns: Different aspects of a problem should be separated from each other, and each aspect of the problem should be treated on its own

**QR6 Testing and other general quality measuring tools** (informal technical review, “formal” technical review, peere review – four eye principle)

**GENERAL DESIGN PRINCIPLES:**

Not important for this chapter but important in general: Did we make propper use of the general design Requirements:

• At least 1 interface (with one or more implementations)

• At least 5 classes (implementing the application logic)

• At least 1 abstract class (with subclasses)

• At least 1 custom exception class (with usages in your code)

• At least 1 level of depth in inheritance

GDR3 Apply principles of and follow best practices for object-oriented design, such as:

• Use data encapsulation

• Use inheritance, abstraction and polymorphism properly, e.g., follow SOLID6

• Favor object composition over class inheritance

• Use exception handling

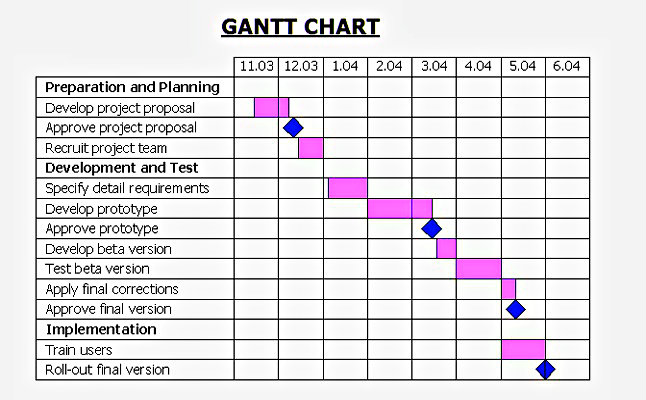
• Program to an interface, not an implementation

• Follow the principles of strong cohesion and loose coupling

# Team Contribution

## Project Tasks and Schedule

Try to break down the overall semester project in several tasks and schedule them appropriately and in accordance with the SUPD and DEAD milestones. Gantt charts like shown in Figure 3 may help to guide and visualize this process. However, feel free to use any visualizations, methods, and tools that support you in meeting the project deadline, and report this here.

  
Figure 3: Sample Gantt Chart from <http://diagramscharts.com/gantt-chart/>. TODO: remove in final submission!

## Distribution of Work and Efforts

Report in a table how you distributed the overall work among team members and how much time was spent by each team member on the tasks.