# Clean Code and Testing

## 1 Overview

The goal of this assignment is to improve the code quality, to test, and to version control a given system. This submission completes all of these requirements including the extension – using external tools (Checkstyle and Findbugs) to guide the refactoring. As an additional achievement, the given system has been extended to allow playing multiple games in succession, without having to restart it.

## 2 Code Style

In refactoring of the system, the guides from lecture 1 and Sun's Java Code Conventions[1] have been applied. The two do not contradict on any point. Sun's Conventions complement the lecture slides on points such as JavaDoc comment styles, structure of if-else and try-catch statements, and indentation. To support applying these conventions, IntelliJ IDEA and the Checkstyle plugin have been used.

### 2.1 Auto-reformatting

IntelliJ has an Auto-reformatting feature. It was used to apply simple conventions. Specifically, it transformed tab indentation to 4-spaces indentation, it put brackets on all one-line if-else, while, do-while, and for statements. It also ordered else, catch, and while statements, such that they are on the same line as the closing bracket for the if, try, or do before them. Some of the earlier commits are only applying the auto-reformat.

IntelliJ also 'optimizes' imports when auto-reformatting. That is, it replaces multiple imports from the same package with a star import (package.*). This goes against Sun's Java Code Conventions and imports were manually re-written (though there are some commits where they were unintentionally left).

### 2.2 Comments

As suggested by the lecture slides, end-of-line comments have been avoided as much as possible. Intentions have been expressed in code. JavaDoc comments have been written for every class, field, and method. Package-info files have also been provided.

## 3 Unit Tests

JUnit4 has been used to write a thorough set of unit tests. The classes, that handle command-line argument parsing and selecting a random phrase to play the game with, have been tested with the standard approach of invoking methods with various arguments and then asserting correctness or expecting exceptions.

For the testing of the classes, that store the game information and interact with the user through standard input, the following approach was taken. An utility class TestingUtils was created to simulate user I/O. It redirects the standard input to read from a string and standard output to write to a string. It also provides a method to compare the outputted string to the contents of a file. This way, every test is related to a file that contains the expected output. Therefore, test have the following structure: redirect input, play a game, compare output to contents of a file.

Tests have been written to account for all conceivable inputs – valid inputs, that the system works with; and invalid inputs, which are detected and handled appropriately. This ensures that the system is robust.

## 4 Design Decisions

The aim of the practical is to clean and test given code, rather than add features. The original design remained largely unchanged. Three major decisions were made.

### 4.1 Object-less implementation

The given code never handled multiple objects of the same class at the same time. In other words, there were never two or more 'GameStates' or 'CommandOpts' or others at the same time. All methods and fields were thus converted to static. This avoids object creation overhead and makes conceptually more sense to have only one set of command-line option, one game, one interface to the phrases for the game.

### 4.2 Phrases not Words

In a game of Hangman, the player often guesses a phrase made of multiple words (such as 'St Andrews' or 'Kingdom of Fife"). Variables, methods, and class names were all converted to refer to phrases when dealing with what the player is guessing.

### 4.3 Play again

The system has been extended to allow the player to play multiple consecutive games. After a game is over (regardless of win or lose), the player is asked if they wish to play another game. If so, a new game is started (the player again chooses a category). This is similar to restarting the system, except that the command-line arguments are already validated and the file with custom phrases (if one is provided) is already loaded.

## 5 Bibliography

[1] Java Code Conventions
Sun Microsystems
https://www.oracle.com/technetwork/java/codeconventions-150003.pdf