**Team Adas**

**Refactoring documentation**

# Overview

The document describes the refactoring process of the game Bulls and cows. Main purpose of the project is to provide high quality code following the best practices of the programming practices introduced in the course "High-Quality Programming Code".

On the basis of the source code provided the team has made refactoring and improvements of the code while following the best practices. As a result the obtained source code is readable, maintainable and testable. All the refactoring steps will be described and will be explained their role for the final result.

# Refactoring process

## Redesigned the code structure

1. Renaming the files with appropriate names according to their function for the game logic. For example:
   1. Program to BullsAndCowsGame
   2. BullsAndCowsNumber to SecretNumberProcessor
   3. rezultat to PlayerResult
   4. gameScore to AnswerProvider
2. Extracting all the constants (strings and game commands) in a static class with appropriate name.

## Code reformatting

1. Moved the using directives inside the namespace in all files.
2. Added line spaces when needed – for example after all block enclosed by curly braces.
3. Removed extra empty lines where needed – (for example in the file gameScore.cs).
4. Rename class members according to the appropriate code convention using intuitive names. For example - public int cheats ot public int HintsUsed; public string GetCheat() to public string GetHint();

## Design patterns

Implemented design patterns:

• Structural patterns – **composite**, **extensibility**, **façade**;

• Behavior patterns – **chain of responsibility,** **command**;

• Creational patterns – **singleton;**

**Unit tests**

We designed and implemented unit tests covering the entire project functionality. To ensure the project works correctly according to the requirements and behaves correctly in all possible use cases. In order to make the code testable the code was redesigned but not the program logic.

Tests cover normal expected behavior (correct data) and possible expected failures (incorrect data). Special attention was put to the border cases. The code coverage of the unit tests is 50%.

**Principles followed**

We tried to follow the idea of **strong cohesion** and **loose coupling**. Mainly we followed the SOLID and DRY principles.

**SOLID** – ever class maintains only its own functionality. For instance HintProvider returns info only when a hint is requested.