# Refactoring Documentation for Project “Hangman ”

Team “Hangman-5”

1. Redesigned the project structure:

* Renamed the project to “HangmanGame”.
* Renamed the main class Program to Hangman.
* Extracted each class in a separate file with a good name.
* Implemented 6 design patterns:
  1. Singleton Pattern for the scoreboard:

public sealed class Scoreboard

{

private const int **TopScoreMaxRecords** = 5;

private static readonly List<KeyValuePair<int, string>> GameTopScore = new List<KeyValuePair<int, string>>();

...

* 1. Factory Pattern:

public static IMessage GetMessage(MessageType messageId)

{

switch (messageId)

{

case MessageType.**welcome**:

return new WelcomeMessage();

case MessageType.**exit**:

...

* 1. Extensibility Pattern:

public static class ClassExtensions

{

public static void ThrowIfArgumentIsNull<T>(this T obj, string text) where T : class

{

if (obj == null)

{

throw new ArgumentNullException(text + " not allowed to be null");

}

}

...

* 1. Adapter Pattern:

public class Words : Word, IPrintable // extends Word, without changing Word's Methods

{

private static readonly string[] WordsArray = new string[] { "computer", "programmer", "software", "debugger", "compiler", "developer", "algorithm", "array", "method", "variable" };

public Words(string word) : base(word)

{

}

...

* 1. Command Pattern:

public class GetUserInputCommand : ICommand

{

private readonly IUserInputHandler handler;

public GetUserInputCommand(IUserInputHandler handler)

{

this.handler = handler;

}

public void Execute()

{

this.handler.GetUserInput();

}

}

}

...

* 1. Template Method Pattern:

public abstract class Game

{

/// <summary>

/// The Template Method

/// </summary>

public void Play()

{

this.Initialize();

while (!this.IsWon())

{

this.Update();

}

}

...

1. Reformatted the source code:
   * Removed all unneeded empty lines.
   * Inserted empty lines between the methods.
   * Split the lines containing several statements into several simple lines;
   * Formatted the curly braces { and } according to the best practices for the C# language.
   * Put { and } after all conditionals and loops (when missing).
   * Character casing: variables and fields made camelCase; types and methods made PascalCase.
   * Formatted all other elements of the source code according to the best practices introduced in the course “High-Quality Programming Code”.
   * Renamed variables with names, according to their purpose and the best practices.
   * Introduced constants and enumerations where needed.
2. Introduced various methods and classes, for example UserInputHandler, which is used by the Command Pattern.