Refactoring Documentation for Project “Hangman”

1. Redesigned the project structure:

* Renamed the project to **HangmanGame**
* Renamed the main class **besenka** to **HangmanMain**
* Extracted/Created the following classes in separate files:
  + **ConsoleRenderer.cs**
  + **LetterHandler.cs**
  + **RandomWordGenerator.cs**
  + **ScoreManager.cs**
  + **CommandParser.cs**
  + **Game.cs**
  + **Player.cs**
* Created enumeration **LetterStatus.cs**

1. Reformatted the source code:

* removed unnecessary comments, e.g. “hahaha, izpih edno kilo rakiya vcera i poznavam veche vsichki dumi4ki ot pyrvi puyt, muahahahahahahaaaaaaaa”
* 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](http://codecourse.telerik.com/)”.

1. Renamed variables:
   * In class **RandomWordGenerator: rand**🡪 **randomizer**
   * In class **RandomWordGenerator: r**🡪 **wordIndex**
   * In class **RandomWordGenerator: arr**🡪 **words**
   * Namespace **Proekt\_KPK** 🡪 **HangmanMain**
   * Class name **besenka** 🡪 **HangmanMain**
   * In class **besenka: bukvata**🡪 **guesedLetter, revealedLetter,** depending on the usage
   * In class **besenka: word**🡪 **wordToGuess**
   * In class **besenka: dashWord**🡪 **wordToDisplay**
   * In class **besenka: tempArray, l:** 🡪 **not used in final code**
2. Introduced constant **MaxNumberOFPlayers**
3. Extracted methods **ParseCommand()** and **ExecuteCommand()** from the **Main()** methodand moved them to the **CommandManager** class.
4. Removed variable **m**, responsible for the health state of the player, and replaced it with variable **mistakes** in **Game** class, since the implementation requires this functionality.
5. In the original code user-friendly messages were missing and also some required in the project documentation. So a class **ConsoleRenderer** was created. It is responsible for all messages printed on the console, i.e. **PrintWelcomeMessage()**, **PrintExitMessage().**
6. Added method **PrintRepeatingLetterMessage(char guessedLetter) -** to print a message if the entered letter has already been used. This functionality is not considered in the project documentation**.**
7. Changed **Game**, **Letter Handler**, and **Score Manager** to instance classes.
8. Moved **ExecuteCommand** from the **CommandManager** to **Game**, because of the dependencies to the methods and instances in Game
9. Renamed the class **CommandManager** to **CommandParserer** to reflect the change in the functionality
10. Added an additional message to be displayed in **Console.Renderer PrintScoreboard()** when no records are found in the scoreboard yet
11. Added constructor for LetterHandler to take the randomly assigned word as an argument and removed the randomly assigned word from the arguments in the functions in the letterHandler
12. Renamed:
    * In classes **Game, LetterHandler: dashWord** 🡪**wordToGuess**
    * In classes **Game, LetterHandler: userWord** 🡪**wordToDisplay**
    * In class **ScoreManager: MAX\_NUMBER\_OF\_PLAYERS**🡪**MaxNumberOFPlayers**
    * In class Game: BlankWord**() 🡪 GenerateBlankWord()**
13. Implemented methods **GetRevealedLetter()**, **RevealLetter()**, **HandleLetterGuess()**, **FillLetter()** in class **LetterHandler**
14. Applied a fix in **GenerateBlankWord()** to generate the blank work without space separators. Added a private method in ConsoleRenderer to display all words separated by spaces.
15. Applied fixed in the unit tests for the ConsoleRenderer to reflect the change in the expected input because the words are now passed without the space separators
16. Added in **Game** - mistakes counter and check if the player has used help
17. Added in **Game**, **ExecuteCommand** handling for the single letter input
18. Moved the mistakes counter to the **LetterHandler** class. Since a new **LetterHandler** instance is generated when restarting the Game, this will clear the stats for the mistakes too without additional operations.
19. Added a counter for the successful guessed and passed the correct letters count to **PrintCorrectLetterMessage()**
20. Changed **wordToGuess**, **wordToDisplay**, **userInput**, **isGameOver** to public properties with private setter, so they can be used in the tests
21. Moved **usedHelp** and **guessedLetter** to the methods they are used in, so we can reduce the scope of these variables
22. Extracted method **HandleLetterGuessCommand** from **ExecuteCommand** to handle the single letters input
23. Added ToLower in the input in **ExecuteCommand**, so this will be handled well in the unit tests
24. Methods **RevealLetter** and **HandleLetterGuess** in the **LetterHandler** were changes to accept wordToDisplay as a string and passing it by reference was removed. Both are changed to accept string by value and return a string as well.
25. Interfaces – Interfaces were added for all major classes to improve the extensibility of the classes. The Game constructor was changed to accept the abstract interfaces, so we are no longer reliant on the specific implementation but on the abstract class. This allows further extensions of the code by adding a new layer in the structure of the classes.
26. To resolve the deep dependencies in Game, we have extracted the **ExecuteCommand**() method to a new class **CommandManager**. It handles the commands and all needed instances are generated by the constructors of Game and **CommandManager** using the Poor man's dependency injection pattern.
27. Unit testing:

* Class **ConsoleRenderer** contains 15 public methods . Created 17 test methods. Achieved 100% code coverage. Used **StringWriter** to record the console output and to compare it with the expected output.
* Class **RandomWordGenerator** contains 3 public methods . Created 1 test method. Achieved 100% code coverage.
* Class **ScoreManager** contains 4 public methods . Created 5 test methods. Achieved 100% code coverage.
* Class **Game** contains 11 methods. Created 8 test methods. Achieved 24% code coverage.
* Class **LetterHandler** has 5 methods. Created 20 test methods. Achieved 100% code coverage.
* Class **CommandParser** has 1 method. Created 6test methods. Achieved 100% code coverage.
* Total coverage: **85%**