Refactoring Documentation

Project “Hangman-7”

1. Redesigned the original project structure
    - Renamed the project to **Hangman**
    - Renamed the main class from **MainClass** to **Hangman**
    - Made class **Hangman public**
    - Renamed the **TopPlayer** class to **Player**
2. Reformatted the source code
    - Removed all unnecessary empty lines
    - Inserted empty lines where they were needed, e.g. between the methods
    - Removed all unnecessary usings
    - Formatted the curly braces according to the C# quality code recommendations
    - Split into several lines all complex statements
    - Added curly braces for all conditional statements and loops where they were missing
    - Unified the variable name casing to comply with the **camelCase** formatting
    - Unified the method name casing to comply with the **PascalCase** formatting
    - Formatted all elements in the code according to the best quality code practices
3. Added implementation to the class **Player**
    - Made class **public**
    - Added private fields **name** and **score**
    - Added getters and setters to the **Name** and **Score** properties
    - Added **validation logic** to the setters
    - Added new **ToString()** implementation to facilitate the display of the results
4. Updated the code related to **Player** in **Hangman**
5. Introduced class **Scoreboard**

    The **Scoreboard** class is intended to keep in a centralized fashion the top results achieved by the players.

For that purpose it is realized as a combination of the **Singleton and the Lazy-Initialization creational design patterns**, known as Initialization-on-Demand Holder Idiom (http://en.wikipedia.org/wiki/Initialization-on-demand_holder_idiom), which results in a thread-safe lazy singleton initialization.

- Declared class as **public**
- Added private fields **scoreboardInstance** and **topPlayers** (a list that holds the top players results)
- Introduced private constant **NumberOfTopPlayers = 5** – used to limit the number of player positions from the scoreboard which would be displayed
- Added **Scoreboard** constructor – **private**, instantiates an empty **topPlayers** list
- Added public **ScoreboardInstance** property – holds the Initialization-on-Demand Holder Idiom implementation
- Added public method **AddPlayer(Player newPlayer)**
- Added private method **SortByHighestScore()** – uses a Lambda expression
- Added public method **IsNewTopScore(int topScoreCandidate)**
- Added new **ToString()** implementation – used to show the resulting scoreboard without exposing as public the **scoreboardInstance**

6. Updated the code related to **Scoreboard** in **Hangman**
7. Introduced class **GameEngine**

   The **GameEngine** class is intended to hold all methods relevant to the general game flow and as such it is designed after the **Module structural design pattern** (http://en.wikipedia.org/wiki/Module_pattern).

   - Declared class as **public static**

8. Renamed **Top()** method in **Hangman** to **ShowScoreboard()**
9. Moved the **ShowScoreboard()** method to class **GameEngine** and modified it to public
10. Simplified and modified the **ShowScoreboard()** to implement the **Scoreboard** class functionality
11. Renamed **Help(Word Game)** method in **Hangman** to **GiveHint(Word secretWord)**
12. Moved **GiveHint(Word secretWord)** method to class **GameEngine** and modified it to public
13. Modified class **Word**

    - Made class **public**
    - Renamed field variable **w** to **secretWord**
    - Renamed field variable **printedWord** to **maskedWord**

- Changed the type of **maskedWord** from **StringBuilder** to **string**
- Added public properties **SecretWord** and **MaskedWord**
- Added getters and setters to the **SecretWord** and **MaskedWord** properties
- Removed methods **SetPlayedWord(string theWord)**, **GetPlayedWord()**, **SetPrintedWord(System.Text.StringBuilder theWord)** and **GetPrintedWord()**, which were doubling the getters and setters functionality
- Renamed method **CheckForLetter(char TheLetter)** to **ContainsLetter(char letter)**
- Modified method **ContainsLetter(char letter)** – moved the input parameter conversion to lower-case out of the conditional statement and removed unnecessary else

```
if (secretWord.Contains(char.ToLower(letter))
```

↓

```
letter = char.ToLower(letter);
if (this.SecretWord.Contains(letter))
```

- Renamed method **WriteTheLetter(char TheLetter)** to **RevealLetterPosition(char letter**)
- Modified method **RevealLetterPosition(char letter)**
  - replaced repeated **this.secretWord.IndexOf(char.ToLower(letter), wordLength)** statement with a new internal variable **letterIndex**
  - moved the letter conversion to lower-case out and assigned the result to a new internal variable **letterLowerCase**
  - added a new buffer variable **StringBuilder maskedWordBuilder** to simplify the operations
- Renamed method **NumberOfInput(char TheLetter)** to **NumberOfLetterOccurrences(char letter)**
- Modified method **NumberOfLetterOccurrences (char letter)** – moved the input parameter conversion to lower-case out of the conditional statement

```
if (this.secretWord[wordLength].Equals(char.ToLower(letter))
```

↓

```
letter = char.ToLower(letter);
if (this.SecretWord[wordLength].Equals(letter))
```

14. Introduced class **RandomWordGenerator**
    - Declared class as **public static**
    - Added private static readonly field **randomWordGenerator**
    - Added **RandomWordGenerator** constructor – **static**, initializes the private field **randomWordGenerator** with **new Random()**
    - Addedpublic static method **GenerateWord()**
15. Moved **words** array from **Hangman** to **RandomWordGenerator**
16. Removed **Random RandomWord = new Random();** and all related functionality from Hangman
17. Customized class **RandomWordGenerator**
    - Renamed **words** to **Words** and changed its type to **private static readonly**
    - Added functionality to **GenerateWord()** method – returns the value of an index of the **Word** array, defined by the **randomWordGenerator**
18. Modified class **Word**
    - Added **Word** constructor – **public**
    - Assigned **SecretWord** to receive its value from **RandomWordGenerator.RandomWord();**
    - Addedprivate method **Mask(string word)** – responsible for the obfuscation of the **SecretWord**
    - Extracted the obfuscation logic from the **Hangman** class, customized it and included it in the **Mask(string word)** method
    - Assigned **MaskedWord** to receive its value from **Mask(this.SecretWord)**
    - Added **validation logic** to the setters of **SecretWord** and **MaskedWord**, reflecting the new values they receive
19. Updated the code related to **Word** in **Hangman** and **GameEngine**
20. Renamed variable **playerMistakes** in **Hangman** to **numberOfMistakes**
21. Extracted from **Hangman** a new method **EstimateScore(Word secretWord, int numberOfMistakes)** and added it to **GameEngine**
22. Modified **Console.Write("\nYou won with " + numberOfMistakes + " mistakes");** to **Console.Write("\nYou won with " + numberOfMistakes + comment);**, where **string comment = numberOfMistakes == 1 ? " mistake" : " mistakes";**
23. Renamed variable **notUseHelp** to **usedHelp**, changed it to **private** and moved it from **Hangman** to **GameEngine**

24. Introduced class **Game**, class **GameState**, class **InitialState** and class **PlayState**

    These classes participate in the implementation of a **State behavioral design pattern** (http://en.wikipedia.org/wiki/State_pattern), where **Game** acts like the **Context**, **GameState** is **State** and **InitialState** and **PlayState** are the **Concrete States**.

    The State pattern is used to describe the internal states of the game flow and to allow for the transition between them, or out of the game, following the user commands.

25. Defined class **GameState** – a model for the states of the game

    - Declared class as **public abstract**
    - Added public abstract method **PerformAction(Game game)**

26. Defined class **Game**

    The **Game** class is designed in such a way that it would also perform as a Façade to Word and GameState, implementing the **Façade structural design pattern** (http://en.wikipedia.org/wiki/Facade_pattern).

    - Declared class as **public**
    - Added private fields **state**, **word** and **numberOfMistakes**
    - Added public properties **State**, **Word** and **NumberOfMistakes**
    - Added getters and setters to **State**, **Word** and **NumberOfMistakes** properties
    - Added **validation logic** to the setters
    - Added **Game** constructor – **public**, initializes the game state
    - Added public method **Run()** – intended to perform different action depending on the current state of the game

27. Defined class **InitialState** – inherits from **GameState**

    - Declared class as **public**
    - Created **welcomeMessage** variable and assigned it value imported from **Hangman**
    - Added new implementation of method **PerformAction(Game game)**
        - added code to print out the **welcomeMessage**
        - initialized **game.Word** with a **new Word()**
        - initialized **game.NumberOfMistakes = 0**
        - initialized **game.State** and assigned it the value of the next state – **PlayState**

28. Defined class **PlayState** – inherits from **GameState**

- Declared class as **public**
- Added new implementation of method **PerformAction(Game game)**
  - moved all code from **Hangman**, internal to **while(true)**, to the method
  - substituted **while(game.Word.MaskedWord.Contains('_'))** with **if (game.Word.MaskedWord.Contains('_'))**
  - substituted **if (!game.Word.MaskedWord.Contains('_'))** with **else**
  - removed variable **restart** and the following if statement

```
if (restart)
{
    Console.WriteLine("Game is Restarted");
    break;
}
```

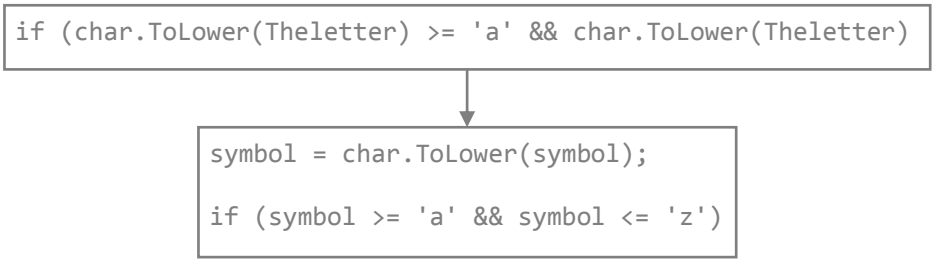
  - added **game.State = new InitialState();** to the **"restart"** case of the **switch**
  - added **game.State = new InitialState();** to the end of the **else** after the **GameEngine.EstimateScore(game.Word, game.NumberOfMistakes);** call
  - added **Console.WriteLine("Good bye!");** to the **"exit"** case of the **switch**

29. Updated **Hangman**
    - Removed parameter **string[] args** from **Main()**
    - Removed all unused variables and constants, e.g. **ONE_LETTER = 1**
    - Added instance **Game game = new Game(new InitialState());** - the game begins in InitialState
    - Added **game.Run()** call inside **while(true)**
30. Extracted from **PlayState** a new method **GuessLetter (Game currentGame, char letter)** and added it to **GameEngine**
31. Modified **Console.WriteLine("Good job! You revealed " + numberOfOccurrences + " letter");** to **Console.WriteLine("Good job! You revealed " + numberOfOccurrences + comment);**, where **string comment = numberOfOccurrences == 1 ? " letter" : " letters";**
32. Introduced class **Decoder** and class **CommandInterpreter**

    The classes **Decoder** and **CommandInterpreter**, along with the **PlayState** class, participate in the implementation of a **Mediator behavioral design pattern** (http://en.wikipedia.org/wiki/Mediator_pattern), where **Decoder** acts like the **Mediator**, **CommandInterpreter** is the **Concrete Mediator** and **PlayState** is a **Coleague**.
33. Defined class **Decoder** – serves as a model for the implementation of a command decoder

- Declared class as **public abstract**
- Added public abstract method **Decode(string command)**

34. Defined class **CommandInterpreter** – inherits from **Decoder**
    - Declared class as **public**
    - Added new implementation of method **Decode(string command)**
        - moved the **switch(command)** statement from **PlayState** to the method
    - Defined new public **delegate Game GetGameDelegate();**
    - Added private field **getGame**
    - Added public static method **AssignGameDelegate(GetGameDelegate getGameDElegate)** – assigns the value extracted by the delegate to the private field variable getGame
35. Added delegate call **CommandInterpreter.AssignGameDelegate(() => game);** in the **PerformAction(Game game)** method of **InitialState**
36. Updated the method calls in the **switch** statement in **Decode(string command)** method of **CommandInterpreter** class
37. Modified method **PerformAction(Game game)** in **PlayState** class
    - Changed the format of **command** to **string**
    - Added instance **CommandInterpreter decoder = new CommandInterpreter();**
    - Added call **decoder.Decode(command);**
38. Moved the all the logic checking whether the command is a single symbol and a valid Latin alphabet letter to the **default** case of the **switch** in the **Decode(string command)** of **CommandInterpreter** class
39. Moved method **Isletter(char Theletter)** from class **Word** to **CommandInterpreter**
40. Renamed method **Isletter(char Theletter)** to **IsValidLetter(char symbol)**
41. Modified method **IsValidLetter(char symbol)** - moved the input parameter conversion to lower-case out of the conditional statement



42. Modified the code in the **default** case of the **switch** in the **Decode(string command)** of **CommandInterpreter** class
    - Updated the **IsValidLetter(char symbol)** calls
    - Consolidated the **two if** statements into a single one **if (command.Length == 1 && IsValidLetter(command[0]))**
    - Added **else** to the if statement
    - Moved the original **default** case logic to the **else** statement
43. Created new **Unit Test project HangmanTest**
44. Added tests for all testable classes of the Hangman project