|  |
| --- |
| Refactoring Documentation for Project “Labyrinth” Author: Svetozar Toskov, f.n. HQ245\_SVTO   1. Redesigned the project structure:    * Renamed the solution to **LabyrinthGame**    * Renamed the project to **LabyrinthGame**    * Created a new project folder **LabyrinthGame**    * Renamed the main class **Program** to **Game**    * Renamed the namespace **EscapeFromLabyrinth** to **LabyrinthGame**    * Created folders for each namespace    * Created separate class file – **Utilities.ScoreRecord**    * Created separate class file – **Utilities.Scoreboard**    * Created separate class file – **Utilities.PlayerPosition**    * Created separate class file – **Utilities.GameUtilities**    * Created separate class file – **Utilities.GameResourceCollection**    * Created separate class file – **GameExceptions.InvalidCommandException**    * Created separate class file – **GameExceptions.InvalidResourceNameException**    * Created new resource file – **Resources.GameResources**    * Created separate project for unit tests - **LabyrinthGameTests** 2. Reformatted the source code:    * Removed all unneeded empty lines    * Removed all the unneeded comments    * Removed all the unneeded using-s    * Inserted empty lines between the methods.    * Formatted the curly braces **{** and **}** according to the best practices for the C# language.    * Put **{** and **}** after all conditionals and loops (when missing).    * Break all the lines longer than 100 columns to several lines    * Split the complex conditions and Boolean expressions in several lines    * Introduced additional variable everywhere before return command    * Character casing: variables and fields made **camelCase**; types and methods made **PascalCase**.    * Replaced all the switch-s with decision tables    * Written comments to all the public methods in the project. 3. Renamed variables:    * In class **Labyrinth**: **ll** 🡪 **labyrinthMap**.    * In method **Labyrinth**. **InitializeLabyrinth**: **random**🡪 **labyrinthCellState**.    * In method **Labyrinth**. **InitializeLabyrinth**: **i**🡪 **row**    * In method **Labyrinth**. **InitializeLabyrinth**: **j**🡪 **column**    * Renamed 4. Renamed methods    * In class **Labyrinth**: **InitializeLabyrinth**🡪 **InitializeLabyrinthMap**    * In class **Labyrinth**: **ShowLabyrinth** 🡪 **PrintLabyrinth**    * In class **Labyrinth**: **Move** 🡪 **Play**    * In class **Labyrinth**: **ShowTopScores**🡪 **PrintScoreboard** 5. Introduced constants:    * **LABYRINTH\_ROWS = 7**    * **LABYRINTH\_COLUMNS = 7**    * **PLAYER\_INITIAL\_POSITION\_ROW = 3**    * **PLAYER\_INITIAL\_POSITION\_COLUMN = 3**    * **SCOREBOARD\_STORED\_RECORDS\_SIZE = 5** 6. Refactored class **Labyrinth**    * Moved all the string messages to a resource file.    * Corrected the messages printed on the screen to be like in the requirements.    * Removed the class fields counters to local method variables    * Introduced **playerInitialPosition** private PlayerPosition field to keep the player’s starting position in case of new game start    * Introduced **resourcesList** private GameResourcesCollection field to read the game print messages    * Introduced **isGameActive** private bool field to indicate the game status    * Introduced **currentMovementsCount** private int field to count the current game movements    * Introduced **playerCurrentPosition** private PlayerPosition field to keep the player’s position    * Introduced **scoreboard** private Scoreboard field keep the top scores    * Introduced **UserCommand** private enumeration to describe the user commands    * Created constructor in with four parameters defining the labyrinth size and initial player position.    * Created private method **StartNewGame** to prepare the new labyrinth for new game    * Created private method **GetPrintCharacter** to return the correct character according to the labyrinth cell status    * Created private method **ParseUserCommand** translate the user input into understandable command    * Created private method **ExecuteUserCommand** to execute the translated user command using decision table    * Created private method **IsValidMovement** tocontrol the user movements    * Created private method **IsGameWon** to the game status    * Created private method **MoveRight** to move the player position right    * Created private method **MoveLeft** to move the player position left    * Created private method **MoveUp** to move the player position up    * Created private method **MoveDown** to move the player position down    * Created private method **UpdateLabyrinthAfterMovement** to update the labyrinth status after user command    * Created private method **ExitGame** close the game    * Created private method **CompleteWonGame** to update the game status after escaping the labyrinth 7. Created new class **ScoreRecord** to encapsulate the score records structure 8. Created new class **Scoreboard** to encapsulate the list of scores 9. Created new class **PlayerPosition** to encapsulate the player’s position in the labyrinth 10. Created new class **GameResourceCollection** to encapsulate game resource management 11. Created new class **GameUtilities** to reduce the classes coupling     * Created static method **IsLabyrinthSolvable** to verify that the labyrinth has at least one solution     * Created static method **IsPlayerPositionInLabyrinthRange** to check if the player’s positionis within the labyrinth borders     * Created static method **IsPlayerPositionAtLabyrintBorder** to check if the player’s position is at the labyrinth borders     * Created static method **CheckLabyrinthMapValidity** to control the input parameters in the utility method     * Created static method **CheckPlayerPositionValidityInLabyrinthMap**to control the input parameters in the utility method 12. Created exception class **InvalidResourceNameException** 13. Created exception class **InvalidCommandException** 14. Created test project to unit test the labyrinth game covering more than 95% of the code |