Project 1
Report
INF4121/3121

Sebasting Søberg(INF4121) & Thomas Oddsund(INF3121)

March 8, 2016

1 *Requirement 1* - Description, analysis and test cases

1.1 Description

We are going to review the program Hangman, which derives its name from the game, written in the language Java. The program consists of 5 source files, namely;

- Command.java
- FileReadWriter.java
- Game.java
- HangmanTets.java
- Players.java

Command.java only contains an enumerated type, which contains the possible commands for the program. **FileReadWriter.java** contains the logic for reading from and writing player objects to file for the scoreboard functionality, as well as a method for sorting and printing the scoreboard. **Game.java** contains the logic for the game itself, which means handling out, user input, checking input and printing the game dialogue. **HangmanTets.java** only initializes and starts the Game.java logic. **Players.java** contains the data structure for a player, as well as methods for fetching name and score.

1.2 Analysis of the testable parts of the program

1.3 Test cases

1. **Start Hangman**
   Initial state: No program running.
   Steps:

   1. Run "java -cp . hangman.HangmanTets" from the folder containing the hangman folder with binaries

   Expected results: Game displays welcome-statement:

```
Welcome to the Hangman game. Please, try to guess my secret word.
Use 'TOP' to view the top scoreboard, 'RESTART' to start a new game,
'HELP' to cheat and 'EXIT' to quit the game.
The secret word is: *One-or-more underlines*
Enter your guess(1 letter allowed):
```

Post condition: Game awaits input

2. **Single non-alphabetic character**
   Initial State: Game running, awaiting input.
   Steps:

   1. User enters character which is not an english letter

   Expected results: Game ignores input, asks for new character.
   Post condition: Game awaits input.

3. **Multiple characters entered**
   Initial State: Game running, awaiting input.
   Steps:

   1. User enters multiple characters.

   Expected results: Game ignores input, asks for a new character.
   Post condition: Game awaits input.

4. **Correct letter entered**
   Initial State: Multiple underlines are displayed in the secret word, more then one unique letter missing.
   Steps:

   1. User enters a valid letter X.

   Expected results: Game displayes the following message;

   ```
   Good job! You revealed 1 letter(s).
   The secret word is: *Current state of secret word,  letter X revealed*
   Enter your guess(1 letter allowed):
   ```

   Post condition: One or more underlines are replaced with the letter in the correct position, game awaits input.

5. **Wrong letter entered**
   Initial State: One or more underlines are displayed in the secret word.
   Steps:

   1. User enters an invalid letter X.

   Expected results: Game displays the following error message:

   ```
   Sorry! There are no unrevealed letters 'X'.
   The secret word is:  *Current state of secret word*
   Enter your guess(1 letter allowed):
   ```

   Post condition: The state of the secret word remains unchanged, mistake counter increased by one.

6. **Correct letter reveals last letter(no help used)**
   Initial State: One unique letter missing and no help used.
   Steps:

1. User enters a valid letter X

Expected results: Game displays the following success-message;

```
Good job! You revealed 1 letter(s).
You won with Y mistake(s).
The secret word is:  *SECRET WORD*
Please enter your name for the top scoreboard:
```

Where Y is the number of unique incorrect letters typed in by the user.
Post condition: Game ready for input for scoreboard

7. **Correct letter reveals last letter(help used)**
Initial State: One unique letter missing and help used.
Steps:

1. User enters a valid letter X.

Expected results:

```
Good job! You revealed 1 letter(s).
You won with Y mistake(s). but you have cheated. You are not allowed to
enter into the scoreboard.
The secret word is:  *SECRET WORD*
```

Where Y is the number of unique incorrect letter typed in by the user.
Post condition: Game restarts, then displays welcome statement and awaits input.

8. **User enters name for scoreboard**
Initial State: User have guessed the correct word without the help command.
Steps:

1. User guesses the correct word
2. User inputs name for scoreboard

Expected results: Name+score stored in records, new game started
Post condition: Scoreboard has one new entry containing the playername + score, and a new game has started.

9. **Help command**
Initial State: Game running and awaiting user input.
Steps:

1. User enters "help"

Expected results: One letter is revealed in the secret word.
Post condition: Game awaits user input.

10. **Restart command**
Initial State: Game running and awaiting user input.
Steps:

1. User enters "restart"

Expected results: The game starts a new game.
Post condition: New game with a new word running, game awaits user input.

11. **Top command**
Initial State: Game running and awaiting user input.
Steps:

    1. User enters "top"

    Expected results: Game displays the scoreboard and starts a new game.
    Post condition: Game awaits user input.

12. **Exit command**
Initial State: Game running and awaiting user input.
Steps:

    1. User enters "exit"

    Expected results: Game exits
    Post condition: No game running

1.4 Design of manual system tests

1.5 Test cases

2 Metrics at project and file level

2.1 Metrics at project level

2.2 Metrics at file level

3 Improvements based on metrics

3.1 Metrics at project level that needs improvment

3.2 List of improved/refactored code

3.3 New metrics at project level vs old

3.4 New metrics at file level vs old

4 Conclusion

make a couple of remarks about how easy or not it was for you to maintain the code (to modify it in order to improve it). Is there anything that you would have done differently on the ini'al code to make its maintenance easier?