Problem statement: In HW1 you will deliver the first version of a game-application named **WordNerd** that offers word games to the players. This version will have a **Hangman** game played in a character user interface.

Hangman rules: The Hangman game picks up a puzzle-word randomly from **wordsFile.txt**. It then replaces at least half of its alphabets with dashes and asks the player to guess the missing alphabets in 10 trials. If the player makes a correct guess, it is counted as a 'hit', else a 'miss'. If the player makes a guess that is already part of the clue, or was guessed in previous trials, then it is ignored, i.e. not counted in 10 trials. The final score = number of hits / number of misses. If player made no misses, then the score equals the number of hits. The score is calculated after either the puzzle word is guessed completely, or after 10 trials, whichever comes first. You can see some scenarios to understand the game flow in Figure 1 to 6. To keep it simple, assume that all words in words-file and all user inputs are in lower-case.

Solution design: As shown in the class diagram in Figure 7, your application has to be designed in five classes. The WordNerd class starts the application, offers choices to the player to choose a game from a menu, and then plays the chosen game. This version will have only Hangman game. The Twister game listed in the menu will be added in HW2.

The Game class is an abstract class that has members common to Hangman and Twister. It has some abstract and some implemented methods. The Hangman class is a concrete class that implements the Game class and also has its own methods to play the game. A game is played in rounds, represented by the class GameRound. It has a 'puzzleWord' which is randomly selected from the list of words read from wordsFile.txt. The puzzleWord is then converted into a 'clueWord' by the Game's makeAClue() method by replacing some alphabets with dashes as mentioned above. The HangmanRound class extends GameRound and stores hit and miss counts for a round. These counts are used by Hangman's getScore() method to print the score at the end of a round. See comments provided in code files for more details.

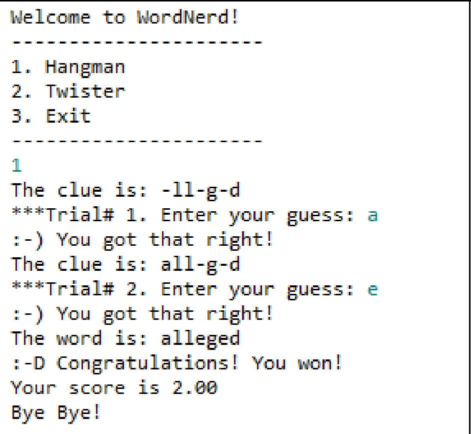
Design constraint: The design is created with a constraint that you cannot use **Collection classes for HW1**

Instructions: Download all java files, wordlist.txt, miniWordList.txt[1]. Copy or import java files in a package named hw1, and the txt files in your project folder. Complete the missing code as per the class diagram (Fig. 7) and instructions provided as comments in code files. You need to fill in method-bodies for the signatures provided. You can add more methods, but cannot change any of the method signatures provided in .java files. You will test your program in two ways: Console output (refer Fig.1 to Fig. 6) and TestHangman.java. These tests will get you 80% of the points. Other criteria applied to evaluate your program are:

1. Documentation (5%): Your code should be well-commented, i.e. neither too many comments, nor too few. Name your variables in a self-explanatory way. Write your name and Andrew id at the top in the comments in each class. Indent your code properly. (In Eclipse, press Ctrl-A to select all your code and then Ctrl-I to indent)
2. Code quality (5%): coding conventions, no unused variables/libraries, etc. Use your judgment to assess these criteria.
3. Code robustness (5%): Your program should not throw any errors while processing. You can safely assume that the user will not enter any garbage input.
4. Submission instructions (5%): Zip all your java files except test-file into AndrewId-hw1.zip. Do not submit any other folders, class files, test file, text files, and rest of your kitchen sink! Only last submission will be graded. Wrong files, incorrect package name, etc. may take away some points.

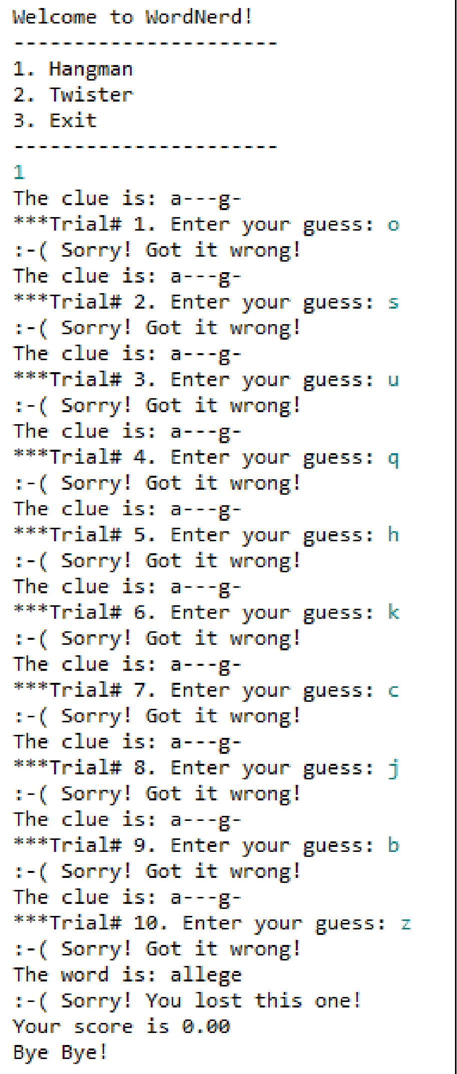
NO LATE SUBMISSIONS PLEASE! If you are unable to submit on time, you lose all the points. Please avoid last minute submission as Canvas may decide to quit on you! Learn to trust technology only to the extent you should! Do not take that risk! **I will not accept late submission. Good luck!!**

[1] *Provided for testing with small set of words. Your program must run with both .txt files. To change the file, change the value of WORD_FILE_NAME in Game.java. DO NOT hard code file name anywhere else in the program.*



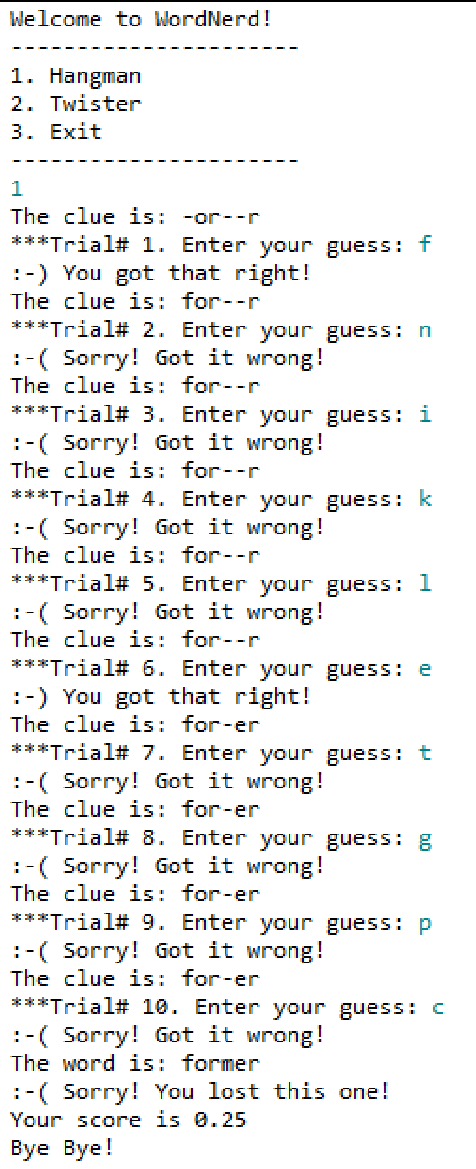
```
Welcome to WordNerd!
----------------------
1. Hangman
2. Twister
3. Exit
----------------------
1
The clue is: -ll-g-d
***Trial# 1. Enter your guess: a
:-) You got that right!
The clue is: all-g-d
***Trial# 2. Enter your guess: e
:-) You got that right!
The word is: alleged
:-D Congratulations! You won!
Your score is 2.00
Bye Bye!
```

Figure 1: All correct guesses.



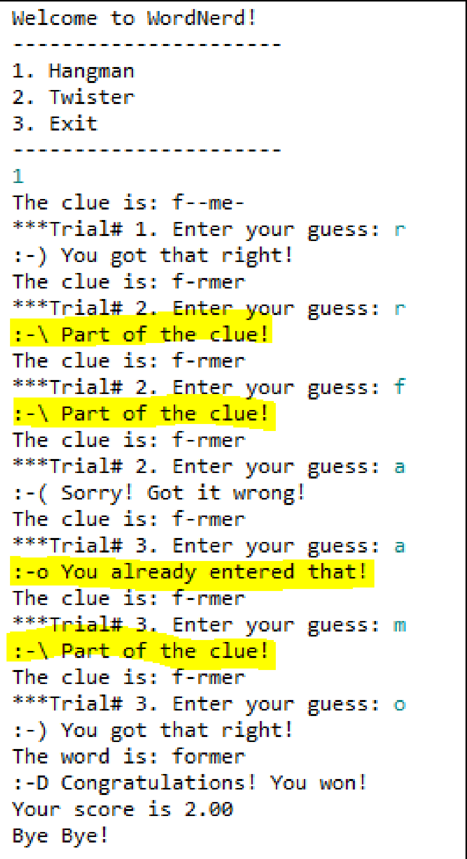
```
Welcome to WordNerd!
----------------------
1. Hangman
2. Twister
3. Exit
----------------------
1
The clue is: a---g-
***Trial# 1. Enter your guess: o
:-( Sorry! Got it wrong!
The clue is: a---g-
***Trial# 2. Enter your guess: s
:-( Sorry! Got it wrong!
The clue is: a---g-
***Trial# 3. Enter your guess: u
:-( Sorry! Got it wrong!
The clue is: a---g-
***Trial# 4. Enter your guess: q
:-( Sorry! Got it wrong!
The clue is: a---g-
***Trial# 5. Enter your guess: h
:-( Sorry! Got it wrong!
The clue is: a---g-
***Trial# 6. Enter your guess: k
:-( Sorry! Got it wrong!
The clue is: a---g-
***Trial# 7. Enter your guess: c
:-( Sorry! Got it wrong!
The clue is: a---g-
***Trial# 8. Enter your guess: j
:-( Sorry! Got it wrong!
The clue is: a---g-
***Trial# 9. Enter your guess: b
:-( Sorry! Got it wrong!
The clue is: a---g-
***Trial# 10. Enter your guess: z
:-( Sorry! Got it wrong!
The word is: allege
:-( Sorry! You lost this one!
Your score is 0.00
Bye Bye!
```

Figure 4: All incorrect guesses



```
Welcome to WordNerd!
----------------------
1. Hangman
2. Twister
3. Exit
----------------------
1
The clue is: -or--r
***Trial# 1. Enter your guess: f
:-) You got that right!
The clue is: for--r
***Trial# 2. Enter your guess: n
:-( Sorry! Got it wrong!
The clue is: for--r
***Trial# 3. Enter your guess: i
:-( Sorry! Got it wrong!
The clue is: for--r
***Trial# 4. Enter your guess: k
:-( Sorry! Got it wrong!
The clue is: for--r
***Trial# 5. Enter your guess: l
:-( Sorry! Got it wrong!
The clue is: for--r
***Trial# 6. Enter your guess: e
:-) You got that right!
The clue is: for-er
***Trial# 7. Enter your guess: t
:-( Sorry! Got it wrong!
The clue is: for-er
***Trial# 8. Enter your guess: g
:-( Sorry! Got it wrong!
The clue is: for-er
***Trial# 9. Enter your guess: p
:-( Sorry! Got it wrong!
The clue is: for-er
***Trial# 10. Enter your guess: c
:-( Sorry! Got it wrong!
The word is: former
:-( Sorry! You lost this one!
Your score is 0.25
Bye Bye!
```

Figure 2: Some correct guesses. 10 trials exhausted



```
Welcome to WordNerd!
----------------------
1. Hangman
2. Twister
3. Exit
----------------------
1
The clue is: f--me-
***Trial# 1. Enter your guess: r
:-) You got that right!
The clue is: f-rmer
***Trial# 2. Enter your guess: r
:-\ Part of the clue!
The clue is: f-rmer
***Trial# 2. Enter your guess: f
:-\ Part of the clue!
The clue is: f-rmer
***Trial# 2. Enter your guess: a
:-( Sorry! Got it wrong!
The clue is: f-rmer
***Trial# 3. Enter your guess: a
:-o You already entered that!
The clue is: f-rmer
***Trial# 3. Enter your guess: m
:-\ Part of the clue!
The clue is: f-rmer
***Trial# 3. Enter your guess: o
:-) You got that right!
The word is: former
:-D Congratulations! You won!
Your score is 2.00
Bye Bye!
```

Figure 3: Some correct guesses. Correct word guessed in just 3 trials. The highlighted trials ignored.

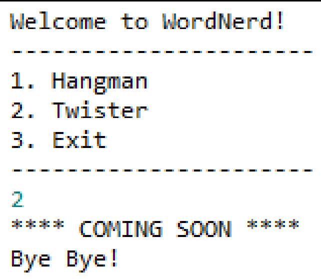
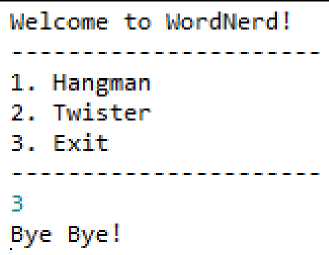
	

Figure 5: Twister menu option chosen	Figure 6: Exit option chosen

```
Welcome to WordNerd!
----------------------
1. Hangman
2. Twister
3. Exit
----------------------
2
**** COMING SOON ****
Bye Bye!
```

```
Welcome to WordNerd!
----------------------
1. Hangman
2. Twister
3. Exit
----------------------
3
Bye Bye!
```

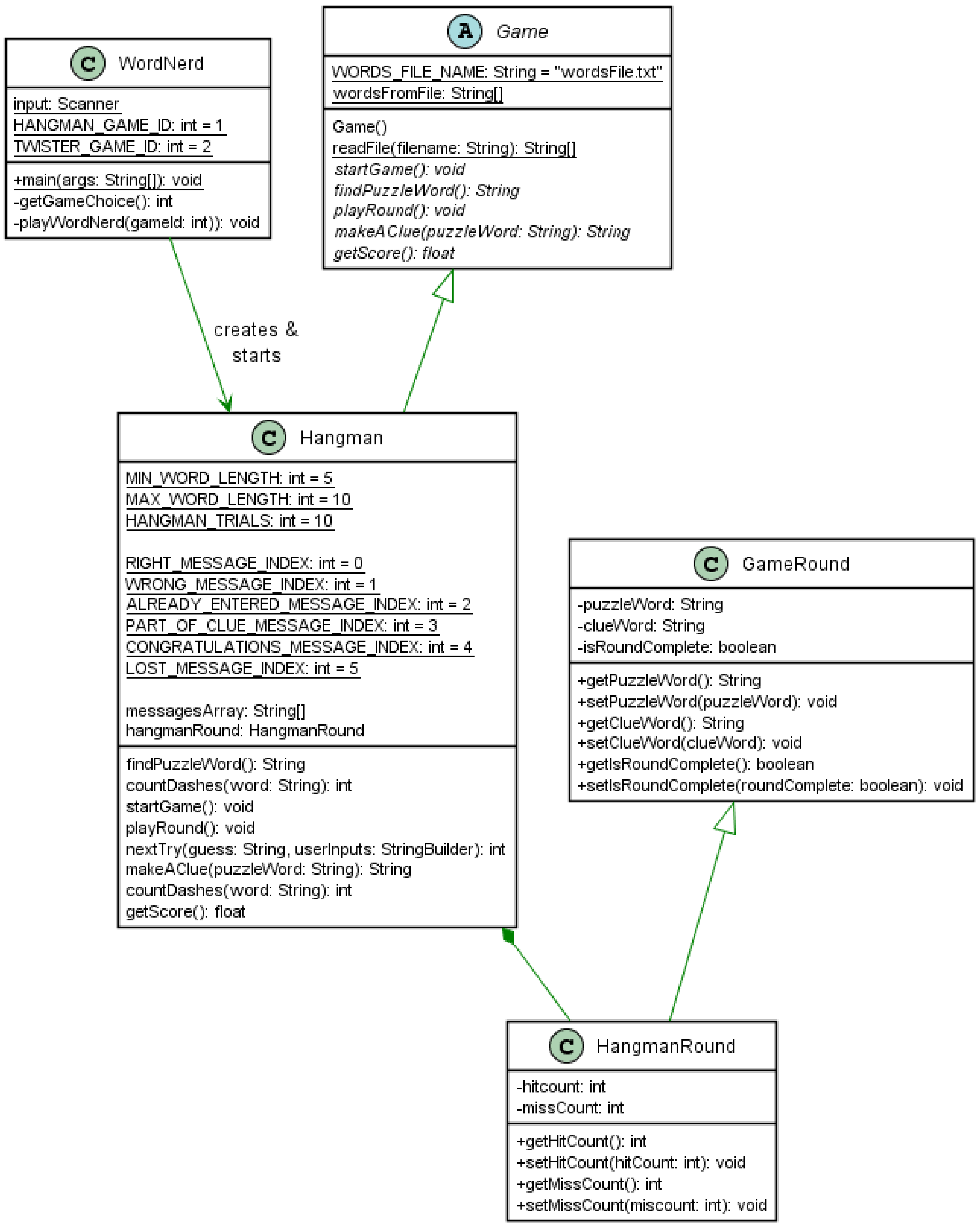

C WordNerd
input: Scanner
HANGMAN_GAME_ID: int = 1
TWISTER_GAME_ID: int = 2
+main(args: String[]): void
-getGameChoice(): int
-playWordNerd(gameId: int)): void
A Game
WORDS_FILE_NAME: String = "wordsFile.txt"
wordsFromFile: String[]
Game()
readFile(filename: String): String[]
startGame(): void
findPuzzleWord(): String
playRound(): void
makeAClue(puzzleWord: String): String
getScore(): float
creates & starts
C Hangman
MIN_WORD_LENGTH: int = 5
MAX_WORD_LENGTH: int = 10
HANGMAN_TRIALS: int = 10
RIGHT_MESSAGE_INDEX: int = 0
WRONG_MESSAGE_INDEX: int = 1
ALREADY_ENTERED_MESSAGE_INDEX: int = 2
PART_OF_CLUE_MESSAGE_INDEX: int = 3
CONGRATULATIONS_MESSAGE_INDEX: int = 4
LOST_MESSAGE_INDEX: int = 5
messagesArray: String[]
hangmanRound: HangmanRound
findPuzzleWord(): String
countDashes(word: String): int
startGame(): void
playRound(): void
nextTry(guess: String, userInputs: StringBuilder): int
makeAClue(puzzleWord: String): String
countDashes(word: String): int
getScore(): float
C GameRound
-puzzleWord: String
-clueWord: String
-isRoundComplete: boolean
+getPuzzleWord(): String
+setPuzzleWord(puzzleWord): void
+getClueWord(): String
+setClueWord(clueWord): void
+getIsRoundComplete(): boolean
+setIsRoundComplete(roundComplete: boolean): void
C HangmanRound
-hitcount: int
-missCount: int
+getHitCount(): int
+setHitCount(hitCount: int): void
+getMissCount(): int
+setMissCount(miscount: int): void

Figure 7: Class Diagram