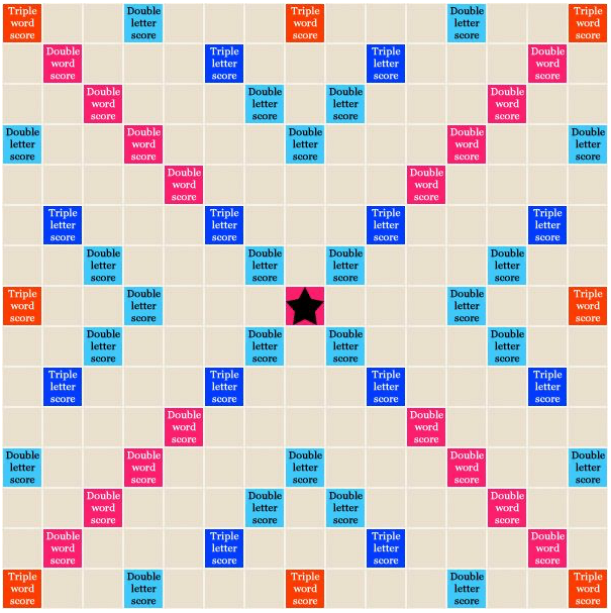
Computer Science 12 **Name: _____**

Scrabble **Date: _____**

Plan, design, implement and test a program to model the classic boardgame Scrabble.



Program Specifications

- Player versus player (2 – 4)
- Command line game (Of course you can add in a GUI!)
- Correct number of letters in letter bag available
- Scrabble tile letter distribution is as follows: A-9, B-2, C-2, D-4, E-12, F-2, G-3, H-2, I-9, J-1, K-1, L-4, M-2, N-6, O-8, P-2, Q-1, R-6, S-4, T-6, U-4, V-2, W-2, X-1, Y-2, Z-1 and Blanks-2
- Accurate scoring system – considering face value of words and position on board
- Point value is as follows:
  - (1 point)-A, E, I, O, U, L, N, S, T, R
  - (2 points)-D, G
  - (3 points)-B, C, M, P
  - (4 points)-F, H, V, W, Y
  - (5 points)-K
  - (8 points)- J, X
  - (10 points)-Q, Z

Program Specifications
Level 4
True to life – accurate board values (double letter, double word etc in right location and applied), only valid words can be player, word played must connect to board in such a way that any new word played must build off existing word on board. At each turn player has 3 options – play a word, exchange tiles, pass. Any number of tiles can be exchanged. First player must start at star in centre of the board. When the game begins, the first player will place their word on the star spin in the centre of the board. The star is a double square and will offer a double word score. Draw for first play - player with the letter closest to "A" plays first. A blank tile beats any letter. Return the letters to the bag and mix Game ends when all of the letters from the letter bag have been drawn. Game can also end if any player passes or exchanges twice in a row. The visual, though text based, should be organized and visually appealing. Program is robust. Clear messages as to what the user did wrong should be displayed if word unable to be played. Design should be user friendly. Fifty point bonus for using all 7 letters. High score feature. Create custom exceptions.

Level 3	Level 2
For the most part, the player is able to only put down valid words that build off an existing word on the board. First player must start at star in centre of the board At each turn player has 2 options – play a word or pass. Game ends when all of the letters from the letter bag have been drawn. The visual, though text based, should be organized. Program is mostly robust. Design is mostly user friendly.	The player is able to put down valid words but it is not true to life (the word might not build off an existing word/collision when word is put down/incorrectly scored). Program is somewhat robust.

In general, in this project you will demonstrate the following expectations:

i. Object oriented programming principles
ii. Effective use of data structures (Arrays, ArrayLists, HashMaps)
iii. Reading and writing files
iv. Effective exception handling

Rubric

Expectations & Achievement Categories		Level 4	Level 3	Level 2	Level 1
Communication	**Class docstrings include description and description of each instance variable.** **Method docstrings include description, description of each parameter and return.**	Docstrings are well written and clearly explain how the method is used.	Docstrings are mostly well written and mostly explain how the method is used.	Docstrings are provided but missing criteria and/or unclear.	Minimal/missing.
Knowledge	**Approaches software design using object oriented programming (as shown in UML)**	UML class diagram is clear and complete.	UML class diagram is mostly clear and complete.	UML class diagram is somewhat complete.	Missing/missing.
	Exceptions are thrown and handled effectively.	Program is robust and handles errors. Try/catch used effectively.	Program is mostly robust and mostly handles errors, try/catch mostly used appropriately.	Program is somewhat robust and somewhat handles errors, try/catch somewhat used appropriately.	Minimal/missing.

Application	**Programming structures are used effectively.**	Data structures (arrays, arraylists etc) are used effectively. Built in methods are used effectively. No unnecessary duplication of algorithms. Conditionals and loops (for and while) are used well and appropriately.	Data structures are mostly used effectively. Built in methods are mostly used effectively. Minimal duplication of algorithms. Conditionals and loops (for and while) are mostly used appropriately.	Data structures are somewhat used effectively. Built in methods are somewhat used effectively. Some duplication of algorithms and/or data structures. Conditionals and loops (for and while) are somewhat used appropriately.	Minimal/missing.
Thinking/Application	**Classes are designed and implemented effectively to solve complex problems.**	Design is clear and easy to understand. Program is divided into classes – each class has a singular purpose. Each class has strong cohesion – all methods in class support singular purpose. Instance variables (properties of object) are well thought out. Date is hidden. Getters and setters are implemented. Special methods are implemented effectively. Methods are written and implemented effectively with purpose. Methods are concise – no method, including the main method should exceed 30 lines.	Design is mostly clear and easy to understand. Program is mostly divided into classes where each class has a singular purpose. Each class mostly has strong cohesion. Instance variables (properties of object) are mostly well thought out. Date is hidden. Getters and setters are implemented. Special methods are implemented mostly effectively. Methods are written and implemented mostly effectively with purpose. Methods are mostly concise.	Design is somewhat clear and easy to understand. Program is somewhat divided into classes where each class has a singular purpose. Each class somewhat has strong cohesion. Instance variables (properties of object) are somewhat thought out. Date is hidden. Getters and setters are implemented. Special methods are implemented somewhat effectively. Methods are written and implemented somewhat effectively with purpose. Methods are somewhat concise.	Minimal/missing.
Thinking	**Software solution meets defined requirements.**	The program exceeds the requirements in the specification.	The program meets the requirements in the specification.	The program meets most of the requirements in the specification.	Minimal/missing.