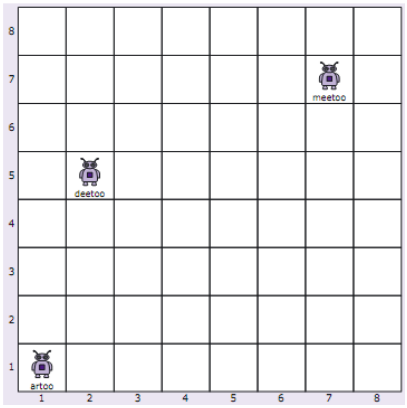
CSCI 1101
Computer Science II
Assignment No. 1
Date Given: Monday, January 23, 2017
Due: Monday, February 6, 2017, 11.55 p.m.
Submission: On Brightspace (dal.ca/brightspace)

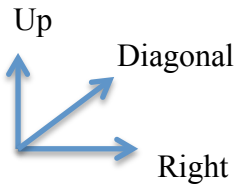
Welcome to your first programming assignment in object-oriented concepts. The assignment consists of two questions. Please follow all the instructions carefully. You must submit one zip file containing the source codes (.java files) and a text document with sample outputs. All your programs must be nicely formatted and commented.

Question No. 1 (60 points)

The objective is to build a simple robot game where the robots are modeled as objects and move around in a 2-dimensional grid as shown in the diagram below. The position of the robot is expressed as x- and y-coordinates, which can take any value within the limits of the grid. The goal of the game is to move the robot so that reaches the grid number (8,8) from grid number (1,1). The robot also faces one of three directions (Up, Right or Diagonal). It can move one, two or three steps in the direction it faces. It can also collect points as it moves across the grid.

The program specification for the first question is long, but read on. The actual programming part is not too hard. You don't need to use 2-d arrays or any other data structures, but just x and y coordinates and random numbers.





Start by defining a robot object (Robot.java) with the following attributes:

- name (a String)
- x position (an integer)
- y position (an integer)
- direction (it faces) (you can decide on this data type)
- points collected (an integer)

Implement the following methods:

- **Constructor** that sets the name and the direction it faces to user-given values. The constructor should also set the x and y positions to (1,1) and the points collected to 0.
- **Get** and **set** methods
- **move** that moves the robot by the given number of steps in the direction it faces.

- **toString** method that returns the attributes of the robot
- **amIAhead** method that compares the x and y coordinates of this robot with the x and y coordinates of another robot and returns true if this robot is closer to (8,8). A simple test is to just add the x and y values and see which is larger. For example, a robot at (3,3) is ahead than the robot at (1,4). As another example, a robot at (7,1) and another at (4,4) are at the same distance from (8,8).

You may add other methods if you find them necessary to complete your program.

Now build a RobotGame.java class that contains the main method to test the above class and run a simple simulation game using two robots as follows:

Prompt the user to enter the name and direction for two robots and create the two robots.

Next, move each robot 1, 2 or 3 steps by generating random numbers. You can use Math.random() to do this.

```
(int)(Math.random() * (max-min)+1) + min
```

generates a random number between min and max, both inclusive.
For example,
(int)(Math.random()*101
will generate a random number between 0 and 100, both inclusive.
(int)(Math.random()*11 + 10)
will generate a random number between 10 and 20, both inclusive.

The Bug.java program in Lab No. 2 shows you how Math.random() works. You can also check out web resources on Math.random(). Change the x and y positions of the robot accordingly. For example, if the robot is facing up and it moves two steps from (1,1) which is the starting point for all robots, then its position changes to (1,3). If the robot is facing Diagonal and it moves three steps, its position changes from (1,1) to (4,4). Note that the robot cannot go beyond the grid. Therefore, if the robot is in box (1,7) and the random number that is generated is 2, it cannot make the move – it stays in its box for this round.

Collect points for each move according to the following rules:

1. At the start, each robot earns 1+1=2 points for being in the box (1,1).
2. If a robot moves into a box (x,y) collect x+y points for that robot and add it to the current number of points.
3. If a robot A's move falls into a box (x,y) which is already occupied by another robot, deduct x+y points from A and do not move A from its current position.

After each move, change the robot's direction by generating another random number from 1 to 3. (1 means direction is Up, 2 means Right and 3 means Diagonal).

After each move, print the attributes of each robot using the toString method. Also print which robot is ahead by comparing using the amIAhead method. If neither one is ahead, print that they are at the same distance.

If one robot reaches (8,8), remove it from further moves (by using a boolean variable or otherwise).

Play the game by putting the moves in a loop and see which robot gets to (8,8) first and which robot collects the most points!

A sample screen dialog would look something like this:

```
Welcome to the Robot Game Simulator!

Enter the name of the first robot: Artoo
Enter the direction of the first robot: Up

Enter the name of the second robot: Detoo
```

```
Enter the direction of the second robot: Right

Simulation started!
Artoo (1,1) Up    2 points
Detoo (1,1) Right 2 points

Move!
Artoo (1,3) Right 6 points
Detoo (2,1) Diag  5 points
Artoo ahead!
```

(this means that in the first move, Artoo's random number was 2 for the steps and 2 for the direction, so it moved up to (1,3) and turned right. The total points for Artoo now is 2 + 4 = 6. Detoo's random number was 1 for the steps and 3 for direction, so it moved to (2,1) and turned diagonal. The total points for Detoo is 2 + 3=5).

```
Move!
Artoo (2,3) Up    11 points
Detoo (4,3) Right 12 points
Detoo ahead!
Move!
Artoo (2,6) Diag  19 points
Detoo (4,4) Right 20 points
Both are at the same distance from the goal!
```

Etc.

```
Move!
Artoo (8,8) Up    56 points
Detoo (7,8) Up    51 points

Artoo has reached the goal (56 points)

Move!
Detoo (7,8) Right 51 points


Move!
Detoo (8,8) Up    66 points

Detoo has reached the goal (66 points)

Game over!
```

Question No. 2 (40 points)

In this question, you are required to construct a class that could be used to play a game of hangman. For those of you who haven't played this game, check out web resources. There is an animation of the game at http://www.proprofs.com/games/word-games/hangman/animation/

We will simplify the game a bit. Rather than randomly generating a secret word from a large dictionary, we will use the words from a much smaller dictionary of 25 words. These will be the names of the first 25 Pokemon characters. These 25 words will be stored in a String array, and each time the game is played, a word is randomly selected from this array. The words are the following:

String[] secretWordDictionary = {“bulbasaur”, “ivysaur”, “venusaur”, “charmander”, “charmeleon”, “charizard”, “squirtle”, “wartortle”, “blastoise”, “caterpie”, “metapod”, “butterfree”, “weedle”, “kakuna”, “beedrill”, “pidgey”, “pidgeotto”, “pidgeot”,“rattata”, “raticate”, “spearow”, “fearow”, “ekans”, “arbok”, “pikachu”};

The class has the following attributes:

a. The secretWordDictionary: the array that has been given above.
b. The secret word: private char[] word (that is, a char array)
c. The disguised word (another array of char), in which each unknown letter in the secret word is replaced with a question mark (?). For example, if the secret word is abracadabra and the letters a and b have been guessed, the disguised word would be ab?a?a?ab?a.
d. The number of guesses made.
e. The number of incorrect guesses.

It will have the following methods:

- No-args constructor: creates an empty Hangman object
- Get methods for the disguised word, number of guesses made and number of incorrect guesses. (DO NOT add public get methods for the secretWordDictionary or the secretword since they must not be accessible from outside. If you need these get methods in your program, make them private).
- Set methods for secret word, disguised word, number of guesses made and number of incorrect guesses. You don’t need a set method for the secretWordDictionary since it is hard-coded into the program.
- generateSecretWord(): Generate a random number from 0 to 24 and select the appropriate word from the secretWordDictionary. Convert it into a char array and store in char[] word.
- makeGuess(char c) guesses that character c is in the word..
- isFound returns true if the hidden word has been discovered.

Here’s the skeleton outline for the program Hangman.java.

```
public class Hangman
{
	private String[] secretWordDictionary = {"bulbasaur","ivysaur”,
	"venusaur","charmander","charmeleon", "charizard", "squirtle",
	"wartortle", "blastoise", "caterpie", "metapod", "butterfree",
	"weedle", "kakuna", "beedrill", "pidgey", "pidgeotto",
	"pidgeot","rattata", "raticate", "spearow", "fearow", "ekans", "arbok",
	"pikachu"};
	private char[] secretWord;
	private char[] disguisedWord;
	private int numGuesses;
	private int numIncorrect;

	//continue the rest of the code

}
```

Test the above class to ensure that all the methods are functional.

Next implement a demo class by writing a demo program Hangmandemo.java and simulate the game of hangman.

Notes:

1. Keep a limit of six on the number of incorrect guesses, and quit the game if the user doesn't get the word right within the limit.
2. You can use two char arrays to keep track of the letters that were guessed, one for correct guesses and another for incorrect guesses. You can use this to prompt the user that they cannot pick a letter that was already guessed.
3. Note the conversions between a char array and a String.

    String s = new String(word);
        will convert a char array word into a String s.
    Similarly,
    char[] word = s.toCharArray();
        will convert a String s into a char array word.

Here are two sample screen dialogs:

```
Welcome to the Hangman game.
I have a secret Pokemon character name. You have to guess it. You are allowed
only six incorrect guesses.

Secret word: ???????
Enter the guess: a
Correct.

Secret word: ?a??a?a
Enter the guess: s
Incorrect. You have 5 incorrect guesses left.

Secret word: ?a??a?a
Enter the guess: t
Correct.

Secret word: ?attata
Enter the guess: g
Incorrect. You have 4 incorrect guesses left.

Secret word: ?attata
Enter the guess: a
You already guessed a. Choose another letter.

Secret word: ?attata
Enter the guess: r
Correct.

Secret word: rattata
You win!
```

Another sample screen dialog:

```
 Welcome to the Hangman game.
I have a secret Pokemon character name. You have to guess it. You are allowed
only six incorrect guesses.

Secret word: ?????
Enter the guess: t
Incorrect. You have 5 incorrect guesses left.
```

```
Secret word: ?????
Enter the guess: s
Correct.

Secret word: ????s
Enter the guess: m
Incorrect. You have 4 incorrect guesses left.

Secret word: ????s
Enter the guess: g
Incorrect. You have 3 incorrect guesses left.

Secret word: ????s
Enter the guess: z
Incorrect. You have 2 incorrect guesses left.

Secret word: ????s
Enter the guess: p
Incorrect. You have 1 incorrect guesses left.

Secret word: ????s
Enter the guess: b
Incorrect. You have 0 incorrect guesses left.

The secret word was ekans.
You lose!
```