CS 211 PROJECT 5: CLASSES AND OBJECTS

DUE MONDAY, APRIL 26TH, 2019 11:59 PM

- THIS ASSIGNMENT IS WORTH APPROXIMATELY 5% OF YOUR TOTAL GRADE
- PLEASE SUBMIT TO BLACKBOARD

The objective of this project is to demonstrate an understanding of the use of recursion and generic classes in Java's Collections Framework.

OVERVIEW:

1. Create the files indicated below and implement the methods described.
2. Download and use a tester module to ensure that your programs are correct. The programs can also be tested using Dr Java's interactive window.
3. Prepare the assignment for submission and submit it.

Files for this Project:
The following files are relevant to the project. Some are provided in the project distribution while others you must create. **You should not submit the files that are provided. DO NOT change any of the data in the provided files.**

File	State	Description
junit-cs211.jar	provided	JUnit library for command line testing
P5Tester.java	provided	Runs all tests
ID.txt	create	Create in setup to identify yourself
Location.java	create	A class to hold game location information
GameLocations.java	create	A class to hold a collection of game locations
GameWorld.java	create	A class to find paths among the game locations

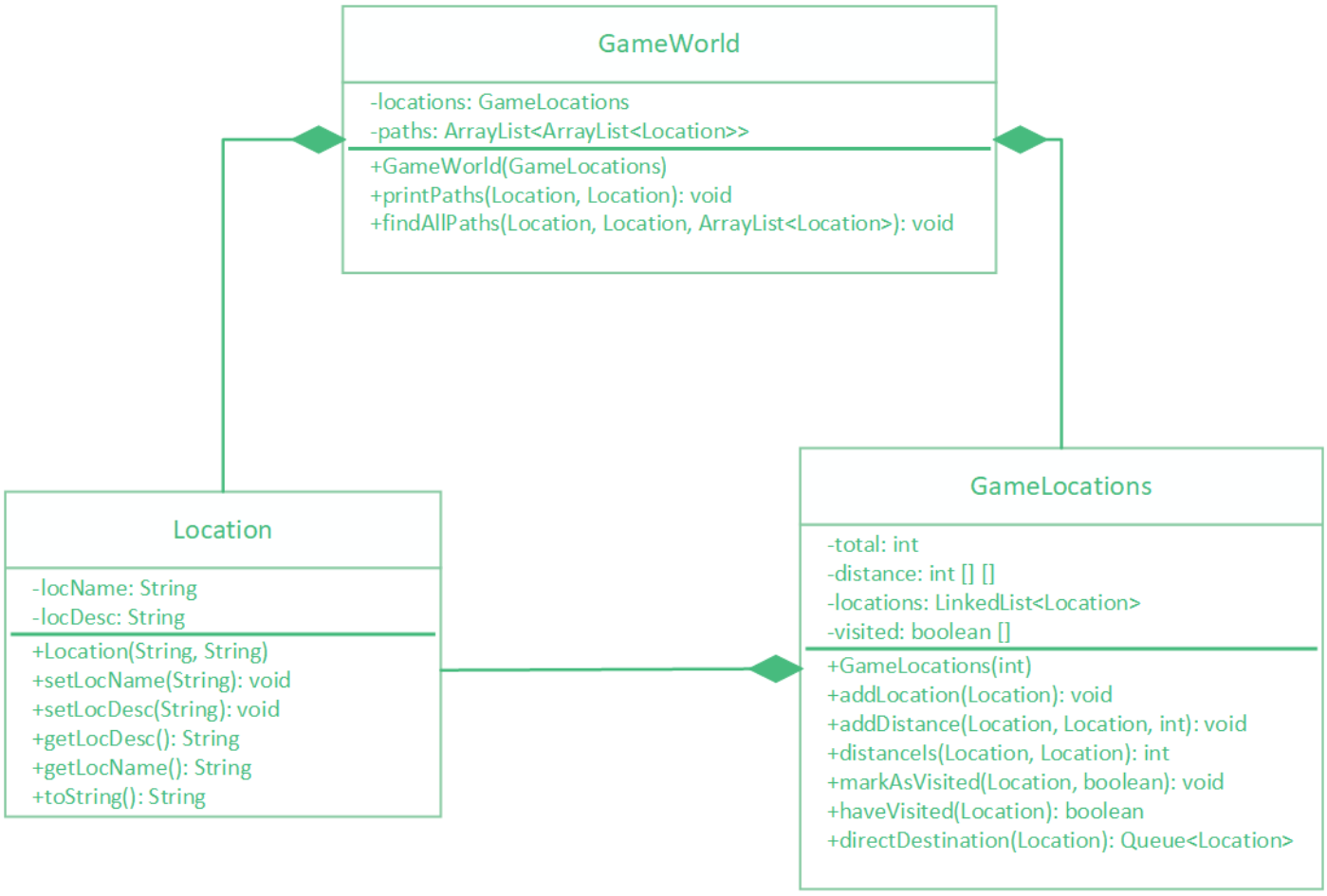
Guidelines and Rules:

- **This project is an individual effort; the Honor Code applies.**
- The main method will not be tested; you may use it any way you want.
- All data fields should be declared private.
- You may add your own helper methods or data fields, but they should be private or protected.
- Format your code clearly and consistently.
- Each class definition will list the fields and methods that must be present. Be sure to match the type and name exactly.
- Use of @Override tags on overridden methods is not required but is strongly recommended.
- When commenting code, use JavaDoc-style comments. Include @param and/or @return tags to explain what is passed in or returned wherever it isn't obvious. Any other JavaDoc tags are optional.

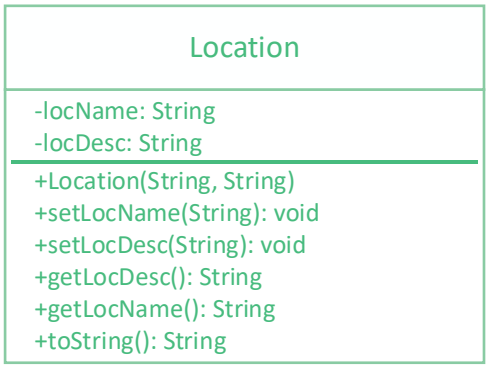
ASSIGNMENT:

You are on a game development team working on a mass multi-player online game. Currently you are responsible for making the classes to help store different locations available in the game world; this will later be used for path finding for NPCs (non-playable characters). The classes are listed in the UML below.

UML:



LOCATION



- The Location class has two private attributes: locName and locDesc, both String types.
- Location(String, String) is a parametrized constructor used to initialize the class attributes.
- The set/get methods are used as mutators/accessors for the class attributes.
- The toString() method is used to return a String of locName and locDesc. The String returned is in this format:
  locName(locDesc)
  ex:
  Abandoned Village(no soul dwell here)

GAMELOCATIONS

Brief Explanation: In the GameLocations class, int [][] distance, LinkedList<Location> locations, boolean [] visited are attributes used to contain data on game locations. The attributes hold data that is parallel.

For Example:
Let us say that a particular game has the following four locations: House, Office, Shop, Gym.

Here is how the data is parallel:

locations might look something like this:

House	Office	Shop	Gym
[0]	[1]	[2]	[3]

The 2D int array distance holds the distances between locations using the index values from the locations list. The distance 2D array might look something like this:

-1	10	12	-1
-1	-1	8	-1
-1	-1	-1	-1
-1	-1	-1	-1

Explanation of the integer values:

- distance[0][1] contains the distance between House and Office = 10 miles
- distance[0][2] contains the distance between House and Shop = 12 miles
- distance[1][2] contains the distance between Office and Shop = 8 miles
- distance[0][3] contains the distance between House and Gym = -1 (meaning there isn't a direct path from House to Gym)

The visited boolean array might look something like this:

true	false	true	true

Again, the data is parallel. visited[0] has true stored in it, which means House was visited.

- The GameLocations class has four private attributes. One 2D integer array called distance, one LinkedList of Location types named locations, one boolean array called visited and one integer variable total.
- GameLocations(int) is the constructor of this class. This constructor is used to initialize the class attributes. The attribute total is initialized to the passed-in integer value, which is the total number of locations in a given game level. distance is a total x total array and visited is a length total array. The LinkedList locations is instantiated in this constructor as well.
- addLocation(Location), this is a void method used to add the passed-in Location object to the LinkedList locations. However, in addition to adding the location to the LinkedList; this method also initializes certain elements in the 2D array distance; initializing distances from the passed-in location to all other locations in the game (and vice-versa) to a default value -1.

  Example:
  Assume we had a total of 5 locations in a game level.
  If the location is at index 0 of the locations LinkedList, then the distance array is initialized to this:

-1	-1	-1	-1	-1
-1				
-1				
-1				
-1				

  If the location is at index 3 of the locations LinkedList, then the distance array is initialized to this:

			-1	
			-1	
			-1	
-1	-1	-1	-1	-1
			-1	

- addDistance(Location from, Location to, int dist), this void method is used to set the specific row and col of the distance array to the passed-in integer value. The row is determined using the from Location object that is passed-in. The column is determined using the to Location object that is passed-in.
- distanceIs(Location from, Location to), this method returns a specific int value from the distance array. The value to return is determined by using the passed-in Location objects.
- markAsVisited(Location, boolean), this void method is used to set a specific location in the visited array to the passed-in boolean value. The specific location is determined by the passed-in Location object.
- haveVisited(Location), this method returns a boolean. This method returns a specific value stored in the visited array. The specific value is determined using the Location object passed-in.

- directDestinations(Location), this method returns a Queue holding Location objects. Based on the Location object passed in, this method returns a Queue of all Location objects that can be visited directly from the passed-in Location object.

Ex:

```
GameLocations g1 = new GameLocations(4);

Location l1 = new Location("House", "live");
Location l2 = new Location("Office", "work");
Location l3 = new Location("Shop", "buy");
Location l4 = new Location("Gym", "exercise");

g1.addLocation(l1);
g1.addLocation(l2);
g1.addLocation(l3);
g1.addLocation(l4);

g1.addDistance(l1, l2, 10);
g1.addDistance(l1, l3, 12);
g1.addDistance(l3, l4, 6);
g1.addDistance(l2, l3, 8);
g1.addDistance(l4, l1, 20);

Queue<Location> returnedValue = g1.directDestinations(l1);
System.out.println(returnedValue);
```

Output:
[Office(work), Shop(buy)]

Explanation:
Based on the locations and distances added above, the only **direct** destinations that can be reached from l1 are l2 and l3. Meaning the directDestinations from l1, which is House, are Office and Shop; as you can see in the output.

GAMEWORLD

- The GameWorld class has two private attributes. One GameLocations object locations, and one 2D ArrayList of Location objects named paths.
- GameWorld(GameLocations), this constructor initializes the class attribute locations using the passed in GameLocations object. This constructor also instantiates paths.

- printPaths(Location, Location), this void method takes in two location objects; one representing a current location and one representing the destination; the method uses the two passed in values to call on the recursive method findAllPaths(Location, Location, ArrayList<Location>); the recursive method populates the paths 2D ArrayList. After calling the recursive method, this method prints out all the paths available between the from Location to the to Location. Use the below output examples for the print formatting.

  Ex:

```
GameLocations g1 = new GameLocations(4);

Location l1 = new Location("House", "live");
Location l2 = new Location("Office", "work");
Location l3 = new Location("Shop", "buy");
Location l4 = new Location("Gym", "exercise");

g1.addLocation(l1);
g1.addLocation(l2);
g1.addLocation(l3);
g1.addLocation(l4);

g1.addDistance(l1, l2, 10);
g1.addDistance(l2, l1, 10);
g1.addDistance(l1, l3, 12);
g1.addDistance(l3, l4, 6);
g1.addDistance(l4, l3, 6);
g1.addDistance(l3, l2, 5);
g1.addDistance(l2, l4, 9);
g1.addDistance(l4, l1, 20);

GameWorld world = new GameWorld(g1);
world.printPaths(l1, l4);
world.printPaths(l3, l1);
world.printPaths(l4, l2);
```

Output:

```
Path 1: House(live)--10->Office(work)--9->Gym(exercise)
Path 2: House(live)--12->Shop(buy)--5->Office(work)--9->Gym(exercise)
Path 3: House(live)--12->Shop(buy)--6->Gym(exercise)

Path 1: Shop(buy)--5->Office(work)--10->House(live)
Path 2: Shop(buy)--5->Office(work)--9->Gym(exercise)--20->House(live)
Path 3: Shop(buy)--6->Gym(exercise)--20->House(live)

Path 1: Gym(exercise)--20->House(live)--10->Office(work)
Path 2: Gym(exercise)--20->House(live)--12->Shop(buy)--5->Office(work)
Path 3: Gym(exercise)--6->Shop(buy)--5->Office(work)
```

- findAllPaths(Location, Location, ArrayList<Location>), this void method is a **recursive** method.
  To earn full credit on this assignment, this method MUST be a recursive method. If you pass the tester/grader using a different methodology than a recursion (i.e using only loops and no recursion), points are deducted manually. [Side Note: If you use a loop in this method **in addition to** recursion, you will **not** lose points].

- Let us look at an example to determine how this algorithm works:

```
g1.addDistance(l1, l2, 10);
g1.addDistance(l2, l1, 10);
g1.addDistance(l1, l3, 12);
g1.addDistance(l3, l4, 6);
g1.addDistance(l4, l3, 6);
g1.addDistance(l3, l2, 5);
g1.addDistance(l2, l4, 9);
g1.addDistance(l4, l1, 20);

GameWorld world = new GameWorld(g1);
world.printPaths(l1, l4);
```

When the printPaths() method above is called passing in l1 and l4, the recursive method findAllPaths, (called from within printPaths), will first look at all possible direct destinations coming from l1. If you look at the above code, you see that all possible direct destinations from l1 are l2 and l3. The recursive method will then look at all the possible direct destinations from l2 and l3. If you look at the above, the direct destinations from l2 are l1 and l4. The direct destinations from l3 are l2 and l4. Utilizing this information, the recursive method determines the following paths:

l1 -> l2
l1 -> l3

l1->l2->l4 [we have a path]
l1->l2->l1 [we won't repeat same thing over again]

l1->l3->l4 [we have a path]

l1->l3->l2
l1->l3->l2->l1 [we won't repeat same thing over again]
l1->l3->l2->l4 [we have a path]

As you can see, based on the algorithm, we end up with 3 different paths. As seen in the output above.

Tip:

Base case:
    the passed-in two locations are the same
Recursive case:
    check each direct destination

GRADING:

Rubric:

- Location (20 pts)
- GameLocations (35 pts)
- GameWorld (35 pts)
- Documentation and Readability (5 pts) [JavaDoc style Commenting, Proper Indentation/Overall Structure]
- Correct Project Setup (5 pts) [zip format, file name, directory structure, files included, ID.txt]

To receive credit for this project you must submit it to Blackboard. Credit will be assigned based on the fraction of tests you pass. You must test your class with the provided tester to ensure that your code can compile and execute. Keep in mind, though, that the grader will have more tests than the provided tester and most of them will be different from the tester. Moreover, **it is your responsibility to verify that the tests pass on the command line, not on IDEs.** Test failures during grading will receive no credit and will not be examined for regrading.

Failure to submit your work in an appropriately named directory and with the ID.txt file in it with correct information will result in a 5% grade deduction.

After verifying that your code runs, zip your directory, and submit it to Blackboard. You may submit as many times as you like; only the final valid submission will be graded. If you turn in late work within 48 hours, there is a penalty (see syllabus). If you turn in late work beyond 48 hours, the submission will be ignored, and you will get zero points.

You should **always verify that your submission is successful**. Go back to the assignment and download your last submission; is there actually a file there? Can you unzip that file and see .java files in there as you would expect? Can you run those files? It turns out you can do quite a bit to ensure you have properly submitted your work. Do not get a zero just because of a failure to turn in the right files! It is your responsibility to correctly turn in your work.

TESTING:

Download the following files provided in the Project 5 assignment link in Blackboard:

- junit-cs211.jar
- P5Tester.java

This is a unit tester which is what we will use to test to see if your classes are working correctly.

The tester file may not be provided on the same day as the assignment instruction is posted. An announcement is made when the tester file is available.

When you think your classes are ready, do the following from the command prompt to run the tests.

On Windows:

```
javac -cp .;junit-cs211.jar *.java
java -cp .;junit-cs211.jar P5Tester
```

On Mac or Linux:

```
javac -cp .:junit-cs211.jar *.java
java -cp .:junit-cs211.jar P5Tester
```

In Dr Java:

- open the files, including `P5Tester.java`, and click the *Test* button.

In Eclipse:

- create a new JUnitTest Case (File --> New --> JUnit TestCase)
- Name it P5Tester
- Copy/paste the content of P5Tester.java in the new file
- Save the file
- Right click on the P5Tester.java on the Package explorer
- Chose Run As -->JUnit Test

SUBMISSION:

Submission instructions are as follows.

1. Let *xxx* be your lab section number and let *yyyyyyyy* be your GMU userid (netid). Create the directory `xxx_yyyyyyyy_P5/`
2. Place all of the `.java` files that you've created into the directory.
3. Create the file `ID.txt` in the format shown below, containing your name, netid, G#, lecture section and lab section, and add it to the directory.
   Full Name: Donald Knuth
   userID: dknuth
   G#: 00123456
   Lecture section: 001
   Lab section: 213

4. Compress the folder and its contents into a .zip file and upload the file to Blackboard.