Quiz 1

SOLUTIONS

Study Guide / Practice Problems

Topics:

- Simple Classes
    - Resources
        - Module 03 / 04
        - Lab 1 part 2
    - You should know
        - How to define OOP terms in your own words/use properly in sentences
            - OOP, class, object, instance variables, access modifiers, encapsulation, constructor, public, private, static
        - Some benefits of OOP
        - Be able to make a class or parts of a simple class following OO rules
- Util Classes (Maps, Lists, etc)
    - Resources:
        - Modules 01-04
            - (Map example problem 04)
        - Lab 1 part 1:
        - Lab 1 part 2:
            - getGradeFor method
    - You should know:
        - How to do a for each loop vs for loop
        - How to use the java api methods for an ArrayList / HashMap
        - How to index into/loop through an Array vs ArrayList
        - How to loop over objects and grab data from objects while looping
        - How to loop through an use elements in a HashMap

Study Material:

- The modules listed above, this practice quiz, and lab1

The quiz will be during lab on canvas - open notes, open internet, closed friends

Practice problems

1. Designing Simple Classes:

Write a simple class called `Book`. This class should hold information about the number of pages and title of the book only. You should be able to set both pieces of data when constructing the object. You should be able to access the page number, but not be able to change it **from outside the class**. You should be able to access and change the book title from outside the class (but still follow OOP guidelines).

```
public class Book {
    //this should be int, not float
    private int pages;
    private String title;

    public Book(int pages, String title)
    {
       this.pages = pages;
       this.title = title;
    }

    public int getPages()
    {
       return pages;
    }

    public String getTitle()
    {
       return title;
    }

    public void setTitle(String newTitle)
    {
       title = newTitle;
    }
}
```

2. Util Classes (ArrayLists, Arrays, HashMaps):

a.

For this problem, reference the class you made in part 1.
Assume an **array** of type `Book` has been declared with the name `books`.
Assume it has been filled with Book objects.

Write a for loop (not for each loop) to count how many books are novellas (have less than 100 pages).

```
int numNovellas = 0;

for (int i = 0; i < books.length; i++)
{
    if (books[i].getPages() < 100)
       numNovellas++;
}
```

b.

Which parts of the for loop would change if it were a for each loop? If it were an ArrayList instead of Array?
For each loop:

```
//for each loop
for (Book b: books)
{
if (b.getPages() < 100)
     numNovellas++;
}
```

ArrayList:

```
for (int i = 0; i < books.size(); i++)
{
if (books.get(i).getPages() < 100)
     numNovellas++;
```

```
        }
```

c. What are some differences betweenArrays and ArrayLists?
Syntax, but more importantly available methods. An ArrayList is an array internally but has helpful methods we can call. It also only can hold objects (but we have wrapper classes to help with that). Arrays can hold primitive data types and objects.

d. (Maps) Given the following classes Pokemon and Trainer, complete the method below with the proper implementation. **Note: this example is more intricate than the one you will get on the quiz. Go over the examples from class and lab 1 part 1 if you would rather skip this.**

```
public class Pokemon
{
	private String id;
	private String name;
	private String type;

	public Pokemon(String id, String name, String type)
	{
		this.id = id;
		this.name = name;
		this.type = type;
	}

	public String getID(){ return id; }

	public String getName(){ return name; }

	public String getType() { return type; }

}

public class Trainer
{
	private String id;
	private String name;
```

```
    public Trainer(String id, String name)
    {
        this.id = id;
        this.name = name;
    }

    public String getID() { return id; }

    public String getName() { return name; }
}


public class PokemonTracker
{


   /*
    * TODO:  fill in the following method to return the team
    *        with the max number of fire type pokemon, given
    *        the following parameters.
    *
    *        trainersFromTeam: a map where the key is the team (ex//
    *                          "yellow", "red", "blue") and the value
    *                          is a list of trainer objects on that
    *                          team.
    *
    *        pokemonFromTrainer: a map where the key is the trainer's
    *                            ID and the value is a list of pokemon
    *                            that belong to that trainer
    *
    *        Return: this function should return the team that has the
    *                maximum number of fire type pokemon out of the
    *                teams in trainersFromTeam
    */
   public static String getTeamWithMaxFire(
      Map<String, List<Trainer>> trainersFromTeam,
      Map<String, List<Pokemon>> pokemonFromTrainer)
   {
          //Variables to keep track of the current best team
          String bestTeam = "";
          int maxFire = 0;
```

```
            //loop through teams and get all the trainers
            for (String team: trainersFromTeam.keySet())
            {
               //count of fire type on this team
               int fire = 0;

               List<Trainer> trainers = trainersFromTeam.get(team);
               if (trainers != null)//can't loop if null!
               {
                  //loop through trainers on team to get pokemon
                  for (Trainer trainer: trainers)
                  {
                    List<Pokemon> poks= pokemonFromTrainer.get(trainer.getID());
                    if (poks != null) //can't loop through null list
                    {
                        for (Pokemon pok: poks)
                        {
                           if (pok.getType().equals("fire")) // not ==
                           {
                              fire++;
                           }
                        }
                     }
                  }
                   //check if this team beats the best so far
                   if (fire > maxFire)
                   {
                       bestTeam = team;
                       maxFire = fire;
                   }

            }

      return bestTeam;


   }

}
```

3. Definitions:

Analyzing a simple class:

Use the following class to answer the next set of questions:

```
1 public class Point {
2    public static final int COUNT = 0;
3
4    private double x;
5    private double y;
6
7    public Point(double inX, double inY) {
8         this.x = inX;
9         this.y = inY;
10   }
11
12   public double getX() {
13        return x;
14   }
15
16   public double getY() {
17        return y;
18   }
19
20   public void translateX(int dx) {
21        x += dx;
22   }
23 }
```

a. What are the instance variables in Point? x and y

b. Are there any setters/mutator methods in the Point class above? yes

c. What is the return type of getX() method (lines 12-14)? double

d. Are the words 'this.' required on line 8 and 9 in this particular Constructor? If not, when would it be needed? If it is needed, why?

No because the parameters have different names than the instance variables - it's only needed if the parameters names match the instance variables (i.e. if there are conflicting names within the same scope).

Using a simple class:

a. Write the code to declare and create a `Point` named p1, with an x value of 0.5 and y value of 1.0.

```
Point p1 = new Point(0.5, 1.0);
```

b. Write the code to access p1's x value.

```
p1.getX();
```

c. Write the code to shift p1's x value by 5.

```
p1.translateX(5);
```

d. Would the following code compile? Why or why not?

```
System.out.println(Point.COUNT);
Point.COUNT++;
System.out.println(Point.COUNT);
```

No, because COUNT is final. If it was not final, that would be correct since you can access public static variables directly through the class name since they belong to the class.

OOP:

1. Encapsulation is one of the tenants of OOP. What is encapsulation and why can it be useful?

Encapsulation is bundling data and methods. It allows us to have data hiding (private methods). It can also make debugging easier and helps with organization.