Quiz 2

SOLUTIONS

Study Guide / Practice Problems

Topics:

- Object class and Overriding (equals/toString)
    - Resources:
        - Module 07/08 (nim example from that day - equals / toString overridden in Player class)
        - Lab 3
    - You should know:
        - How to describe the Object class
        - How to override equals/toString

- OOP vs Procedural (refactoring)
    - Resources
        - Project 1
        - Lab 2
        - Module 04 / 05 / 07
    - You should know:
        - How to achieve encapsulation
        - How to make a class OOP with data hiding
        - How to make methods static -> instance
        - The difference between static and instance

Study Material:

- p1, lecture material, this practice quiz, and labs

Practice problems

1. Simple Classes and Object Overriding:

Write a Theatre class. The data should be seating capacity, the number of tickets sold, and the name of the production playing. All instance variables should be specified upon construction of the object. None of the instance variables should be accessible or modified outside of the class. You should override the toString and equals methods.

Do not worry about commenting on your code. Do not worry about importing.

You should:

- Add the data and constructor
- Override the toString method to print the following on two lines as shown (filling in the words in carrots with the actual data from the instance variables)

  ```
  Production: <production>
  Tickets Sold: <ticketsSold>
  ```

- Override the equals method
- You **should not** add any getters

```
public class Theatre
{

  private int capacity;
  private int ticketsSold;
  private String production;

  public Theatre(int capacity, int ticketsSold, String
production)
  {
        this.capacity = capacity;
        this.ticketsSold = ticketsSold;
        this.production = production;
  }
```

```
  public String toString() {
        return "Production: " + production +
"\nTickets Sold: " + ticketsSold;
    }

  public boolean equals(Object o)
  {
        if (o == null) {return false;}
        if (o.getClass() != this.getClass()){ return
    false;}
        Theatre t = (Theatre) o;
        return t.capacity == capacity && t.ticketsSold
    == ticketsSold && t.production.equals(production);
      }
}
```

2. Refactoring

a.

In OOP, why have we asked you to design classes with all instance fields being declared as private?

Making instance variables private protects the integrity of the data from unintended (or incorrectly intended) changes. The data may only be accessed externally in a controlled fashion as allowed by the class. In simpler language, the class knows how its data should be used - by making the instance variables private, it can control the use and make sure everything is being accessed correctly.

b.

Why would you want to use an instance method instead of a static method?

You would use an instance method when you want/need to access data for a particular instance of a class. You cannot access instance variables from a static method.

c.

The code below represents a **TVWatcher** that has a location, minutes spent watching TV, and a list of shows they watch. How would you refactor the code to conform to the design specifications you have been learning this quarter? Be able to show your answer with UML, words, or by rewriting the code. **Only add getters and setters as needed! Not for everything.**

```
public class TVWatcher {
      public Point location;
      public int minutes;
      public List<String> shows;

      public TVWatcher(Point location, int minutes) {
            this.location = location;
            this.minutes = minutes;
            shows = new ArrayList<String>();
      }
      // space to maybe add code?


   }
   public class Util {
      public static void printAShow(TVWatcher watcher){
         System.out.println("Start the day with: " +
   watcher.shows.get(0));
      }
      public static void combineMins(TVWatcher watcher1, TVWatcher
   watcher2) {
         watcher1.minutes = watcher1.minutes + watcher2.minutes;
      }
   }
   public class UseWatchers {
      public static void main(String[] args) {
      TVWatcher julie = new TVWatcher(new Point(1,2), 30);
      TVWatcher kirsten = new TVWatcher(new Point(-7,15), 100);

      Util.combineMins(julie, kirsten);
       System.out.println("More minutes: " + julie.minutes);
```

```
}

public class TVWatcher {
   private Point location;
   private int minutes;
   private List<String> shows;

   public TVWatcher(Point location, int minutes) {
         this.location = location;
         this.minutes = minutes;
         shows = new ArrayList<String>();
   }
   // space to maybe add code?
     public void printAShow(){
               System.out.println("Start the day with: " +
shows.get(0));
   }
   public void combineMins(TVWatcher watcher2) {
            minutes = this.minutes + watcher2.minutes; //can
index into directly
    }
      public int getMinutes() {
              return minutes;
    }



}
public class UseWatchers {
   public static void main(String[] args) {
   TVWatcher julie = new TVWatcher(new Point(1,2), 30);
   TVWatcher kirsten = new TVWatcher(new Point(-7,15), 100);

   julie.combineMins(kirsten);
   System.out.println("More minutes: " + julie.getMinutes());
}
```