CMSC 131 Quiz 3 Worksheet

The next quiz for the course will be on Wed, Mar 6. The following list provides additional information about the quiz.

- The quiz will be a written quiz (no computer).
- Closed book, closed notes quiz.
- Answers must be neat and legible.
- Quiz instructions can be found at http://www.cs.umd.edu/~nelson/classes/utilities/examRules.html.
- **Regarding Piazza -** Feel free to post questions in Piazza regarding the worksheet and possible solutions to problems.

The following exercises cover the material to be included in this quiz. Solutions to these exercises will not be provided, but you are welcome to discuss your solutions with the TAs or instructor during office hours. It is recommended that you try these exercises on paper first (without using the computer).

Exercises

1. What is the difference between a static method and a non-static method?
2. When you should define a method as static?
3. What is the difference between a non-static field and a static field?
4. What does "this" represent in a class method?
5. Does a constructor for a class constructs the object?
6. Can an object live in the stack?
7. When are parameters and local variables created and destroyed?
8. Define a class Telephone according to the following information:

   Instance Variables (all private)

   a. area code → integer value
   b. three digit value → integer value
   c. four digit value → four digit value
   d. user name → String reference

   Instance Methods

   a. **Constructor** - Allows you to initialize all the instance variables of the class. Name the parameters after the instance variables (i.e., you must use the **this** reference)
   b. **Default constructor** – Initializes the object to the number 555-555-5555 and the name to null. This constructor relies on the previous constructor for object initialization (i.e., you must use **this**).
   c. **Copy constructor**
   d. **Get/Set methods** – Define get/set methods for all instance variables of the class.
   e. **equals** – Two numbers are considered the same if they have the same area code, three and four digit values.
   f. **toString –** Returns a string with the user name, followed by the phone number.

   Static Variable (private)

   a. count – keeps track of the number of Telephone objects that have been created.

   Static Methods

   a. **getCount** – Returns the count value.
   b. **getDigits** – In a phone keypad numbers 0 to 9 are associated with characters. The getDigits method returns a string with the digits that correspond to the provided string. For example for "CAR" the method will return the integer "227".

9. Define a class called **House** that has the following specification:

   Instance Variables (all private)

   a. id → integer value representing a house's id
   b. rooms → integer representing the number of rooms

   Static Variable (private)

   count → integer with the default value of 0.

   Methods

   a. **Constructor with id and rooms parameters** - Initializes all the instance variables of the object. Name the parameters after the instance variables. You must increase the count accordingly.
   b. **Default Constructor** – Initializes the object with an id value of -1 and a number of rooms equals to 1. You must call the previous constructor in order to implement this one.
   c. **toString** – Returns a string with the id, followed by the rooms, followed by the count. Use the following format:

      id: **id** rooms: **rooms** count: **count**

      where **id** and **rooms** represent the values associated with the object and **count** is the static variable value.

   d. **addToHouse** – **Static method** that has as parameters a **House** reference and an integer representing a number of rooms. The method will increase the number of rooms of the provided house by the provided integer value. The method will return a reference to the **House** parameter.

   e. **addRooms** – **Non-static method** that has an integer as a parameter. The parameter represents the number of rooms to be added to the number of rooms associated with the current object. You must implement this method using the **addToHouse** method.

The following driver and associated output illustrates the functionality associated with the **House** class.

Driver

```
public static void main(String[] args) {
       House house1 = new House(2, 4);
       System.out.println(house1);

       House house2 = new House();
       System.out.println(house2);

       House.addToHouse(House.addToHouse(house1, 10), 50);
       System.out.println(house1);

       house2.addRooms(100).addRooms(500);
       System.out.println(house2);
}
```

Output

```
id: 2 rooms: 4 count: 1
id: -1 rooms: 1 count: 2
id: 2 rooms: 64 count: 2
id: -1 rooms: 601 count: 2
```