coe318 Lab 6: Resistive circuits

Objectives

- Implement a `Node` class.
- Implement a `Circuit` class.
- Implement a `Resistor` class.
- Do a tutorial on debugging.
- **Duration: two weeks.**

In this lab, you will model an electric circuit composed of an arbitrary number of resistors. Each of the two ends of a resistor will be connected to a `Node`. Each `Resistor` will be added to a `Circuit` at the time the resistor is created (i.e. within the constructor.)

Introduction to *IllegalArgumentException()*

What can a programmer do when parameters passed to a constructor make no sense?

For example, in the Resistor class of lab 1, it would make no sense if the resistance specified where negative or zero. Or, in the Lab 3 Counter class, a modulus anything less than 2 would be senseless.

Construction of such "senseless" objects can be aborted in the constructor by *throwing an Exception*.

We will discuss Exceptions in much greater detail later in the course. For now all you need to know is that constructors should use if-statements to detect illegal parameters and, if detected, a `new IllegalArgumentException()` should be *thrown*.

This general technique is illustrated below where (for reasons that don't matter) a constructor of an E object must be passed an integer that cannot be negative and `s` String that cannot be `null`.

```
public class E {
  public E(int i, String s) {
    if (i < 0) {
      throw new IllegalArgumentException("i can't be negative");
    }
    if(s == null) {
      throw new IllegalArgumentException("s can't be null");
    }
  }
```

In the classes you will write in this lab it is *up to you* to determine if it is possible for senseless parameters to be given to the constructor. If so, you have to detect them and throw an IllegalArgumentException.

The Node class

The `Node` class will consist of instance variables, a constructor and a toString() method.

Each node has a unique identifying number (a non-negative integer). The first `Node` created will have an id number of 0 (zero). The next one will have an id number of 1, the next an id number of 2, and so on. The `toString()` method should return the identifying number as a string.

Hint: in addition to an ordinary instance variable containing the identifying number of the Node, you will also need a static variable that indicates what the identifying number of the next Node that is constructed.

The constructor takes no arguments.

Step 1: Create a Netbeans project and Node class

1. Create a Netbeans project called `Circuit`.
2. Create a Java file (class library type) called `Node` in package `coe318.lab6`
3. Determine your instance variables and implement the constructor.
4. Implement the `toString()` method.

Step 2: Implement the Resistor class

The Resistor class has a constructor with the following signature:

```
public Resistor(double resistance, Node node1, Node node2)
```

where `resistance` is the resistance in Ohms and the nodes `node1` and `node2` are the two Nodes the resistor is connected to. (Note that this means the `Node` objects have to be created before the `Resistor`.) Each Resistor should also have a unique identifying number. The first resistor should have the number 1, the second, number 2, etc.

Two methods are required:

- `public Node [] getNodes()` and
- `public String toString()`.

The `getNodes()` method should return an array of Nodes where the first element is the first Node specified in the constructor and the second element is the second specified Node.

The `toString()` method should return a string with 4 components separated by spaces. The first component is the letter 'R' followed by the resistor's id number; the second and third components specify the nodes it is connected to; the fourth component gives the resistance. For example, a 30Ω resistor whose id number is 5 connected between nodes 6 and 9 should have the string representation `R5 6 9 30`.

1. Create a Java file (class library type) called `Resistor`.
2. Implement the constructor and `getNodes` and `toString` methods.
3. Consider the possibility of illegal arguments to the constructor, detect them and throw an exception if one is found.

Step 3: Implement the Circuit class

The `Circuit` class has a special feature: it is a *singleton*, which means that there is only one Circuit object. This behaviour is achieved with the following code:

```
private static Circuit instance = null;
public static Circuit getInstance() {
  if (instance == null) {
     instance = new  Circuit();
  }
  return instance;
}
private Circuit() {} //Yes, the constructor is PRIVATE!
```

In any other class, you can obtain the Circuit object with the code:

```
Circuit cir = Circuit.getInstance();
```

The methods required for Circuit are `add(Resistor r)` and `toString()`. The add method should add the resistor to the collection of resistors in the circuit (hint: use an `ArrayList` instance variable). For example, the Resistor constructor should add the newly constructed Resistor (`this`) to the Circuit instance.

The `toString()` method should return a string composed on the String representations of each resistor separated by newline characters.

Step 4: Submit your lab

1. Submit your lab by zipping it to a file called `lab6.zip`
2. Then use the command `submit coe318 lab6 lab6.zip` to complete the submission.

Step 5: Recommended—Learn how to use the debugger

It is recommended that you learn how to use the Netbeans debugger. There are no marks for doing this, but knowing how to use the debugger can help you in future labs. You should try to learn (at least) how to single step and set breakpoints.