Now that you have some familiarity with the Linux environment, it's time to return to some actual Java programming. This week's lab exercises are intended to serve a couple of different purposes:

- to get you programming again with a couple of (relatively) easy tasks,
- to suggest a "test as you code" methodology which can be useful in uncovering bugs before they get too deeply embedded into your code

The data we'll be working with in these exercises represent the results of a series of coin tosses. There are various ways we could represent them, but one of the simplest is as an array of boolean values (i.e., **`boolean[]`**) using **`true`** to represent "heads" and **`false`** to represent "tails".

In the directory `/home/cshome/coursework/241/pickup/`, you will find some code which you should use as your starting point when completing this lab.

Part one (1%)

The skeleton code provided includes some data fields and a basic constructor. To this, add two functions:

`countHeads()` A method that returns an **`int`** which is the number of occurrences of "heads" in the coin tosses.

`toString()` A method that returns a **`String`** representation of the coin tosses, using **`H`** to represent heads and **`T`** to represent tails.

You can progressively test that each method you write is working as it should. One way of doing this is by adding code to a main method which exercises the other methods as you write them. For example, once you have written your **`countHeads()`** method you could write a simple **`main()`** to use it like so:

```
public static void main(String[] args) {
   boolean[] b = {HEADS, TAILS, HEADS, HEADS, TAILS};
   Coins c = new Coins(b);
   System.out.println(c.countHeads());
}
```

This creates a **`boolean`** array to pass to the provided constructor. It then creates a new **`Coins`** instance using that constructor. And finally, prints out the result of calling **`countHeads()`** on that instance. If our **`countHeads()`** works correctly then we should see the number 3 printed out when we run our program.

Part two (1%)

Now add two more constructors:

`Coins(String c)` Creates a **`Coins`** object from a **`String`** consisting entirely of the characters **`H`** and **`T`** (i.e., the result of applying **`toString()`** to the constructed object should be the original string **`c`**).

`Coins(int length)` Constructs a **`Coins`** object consisting of a series of **`length`** coins – the value of each coin should be determined by a random coin toss.

Also add one more method:

`countRuns()` Returns an **`int`** which is the number of runs in this sequence of coins (a run is a block of coins all showing the same face, so for example in **`HHTHHHTTT`** there are four runs namely **`HH`**, **`T`**, **`HHH`**, and **`TTT`**.

Marking

Check that your program works correctly, and then use the command *241-check* to make sure it passes all of our tests. If it does then you can submit it using *241-submit* as usual. In the unlikely event that you don't complete part two, *241-submit* will allow you to just submit part one.

Reflection and extension

- Why is the **`Coins`** representation using an internal boolean array more convenient than just using a **`String`** consisting of **`'H'`** and **`'T'`** characters? Or is it?
- How might you use the classes and methods that have been written here to investigate the questions:
  - What is the average number of runs when 1000 coins are tossed?
  - What is the average length of the longest run when 1000 coins are tossed, and how are the lengths of the longest runs distributed?

  Of course '1000' here is just a placeholder for "an arbitrary integer n".
- Your pointy-haired boss suggests that "a coin might land on its edge". How could you accommodate his astute observation?