Prepared by Linan Qiu <lq2137@columbia.edu>

Truth Statements

Shorten If Else Statements

```
public boolean flip() {
  boolean coinflip;

  if (Math.random() < 0.5) {
    coinflip = true;
  } else {
    coinflip = false;
  }

  return coinflip
}
```

can be written as

```
public boolean flip() {
  return Math.random() < 0.5;
}
```

making the code more readable and succinct.

Not comparing booleans with `true` or `false`

Let's say you're trying to implement a `findElement()` method on an array of **unsorted** elements. We can implement it as such:

```
public static <T> int findElement(T[] array, value) {
  boolean found = false;
  int index = 0;

  for (int i = 0; i < array.length; i++) {
    if(array[i].equals(value)) {
      found = true;
      index = i;
    }
  }
```

```
  if (found == true) {
    return i;
  } else {
    return -1;
  }
}
```

Now if you ever see yourself typing `something == true` or `something == false`, you are doing something unnecessary. After all, `something` is itself a boolean and can be evaluated directly. In this case, we can shorten the last part of the block to:

```
public static <T> int findElement(T[] array, value) {
  boolean found = false;
  int index = 0;

  for (int i = 0; i < array.length; i++) {
    if(array[i].equals(value)) {
      found = true;
      index = i;
    }
  }

  if (found) {
    return i;
  } else {
    return -1;
  }
}
```

Ternary Statements

The last four lines in the `if` block still seems redundant. You can shorten it further using **ternary statements**, which are an alternative way of writing `if` or `else` loops *that return something*.

All `if` and `else` statements are essentially questions. A block like this:

```
if (vader.isFatherTo(luke)) {
  return "I am your father";
} else {
  return "Oops wrong Jedi";
}
```

is asking the question *"Is vader luke's father? If yes, return the father string. Otherwise, return the Jedi string."* Note the placement of the question mark in this statement.

In ternary, this statement would be written

```
return (vader.isFatherTo(luke) ? "I am your father" : "Oops wrong Jedi");
```

If you read this statement out (paying attention to the position of the question mark), it is literally saying: is `vader.isFatherTo(luke)` true? If so, whatever in the bracket is equal to `"I am your father"`. Otherwise, it is equal to `"Oops wrong Jedi"`.

This portion `(vader.isFatherTo(luke) ? "I am your father" : "Oops wrong Jedi")` actually returns you a value: that value depends on whether `vader.isFatherTo(luke)` returns `true` or `false`. That value, in turn, gets returned by the `return` keyword at the start. Hence, this shortens the entire block to just one line.

Similarly, we can modify our `findElement` method to use ternary:

```
  boolean found = false;
  int index = 0;

  for (int i = 0; i < array.length; i++) {
    if(array[i].equals(value)) {
      found = true;
      index = i;
    }
  }

  return found ? index : -1;
}
```

This makes the code much more compact.

Exiting a Loop Early using Return

This code is still badly written. Let's say the array contains 10 elements, and we managed to find the element we want at the second place. Do we really still need to keep looking through the rest of the array? We don't actually need to do that, but our code currently does. This can make quite a bit of difference in large arrays / linked lists. Let's improve on our code.

```
  for (int i = 0; i < array.length; i++) {
    if(array[i].equals(value)) {
      return index;
    }
  }

  // only reach here if value is not found
  return -1;
}
```

Now, the code returns immediately when we find the element that is equals to value. Hence, the only way we can reach the `return -1` line is if no element matches value.

This can be applied to any kind of iteration that can be killed off early. Cool right?