Java toString and equals

Lab setup

1. Create a new Java project in Eclipse.
2. Add the Bicycle class and BicycleFactory class from "JavaStatic" lab (provided)
3. Add the BicycleListing class from the "JavaEnum" lab (provided)
4. Add a main class that we can use as a driver.

`Object` class

The `Object` class is the base class for all Java classes. If a class does not extend another class, then it is a direct child class of `Object`. If a class extends another class then it is indirectly derived from `Object` through its base class. The derivation of a class from `Object` is implicit in a Java program.

The `Object` class has several methods that all classes inherit. The two methods which we will focus on in this lab are `toString()` and `equals()`.

`toString()`

The `toString()` method returns a String representation of the object.

The default behavior of `Object.toString()` returns a string with the value of `getClass().getName() + '@' + Integer.toHexString(hashCode())`

For example, for the `MountainBike` class, `toString()` might return "MountainBike@214c265e".

We can change this behavior by overriding `toString()` for our own classes.

This method is implicitly called whenever we print an object. Assuming `c` is an object, the following two lines are equivalent.

```
System.out.println(c.toString());
System.out.println(c);
```

Graded Submission Task 1

Override the `toString()` method for the BicycleListing class so that it returns a String in the following format:

Bicycle <ID>: Price: $<price> Condition: <condition>

For <condition> print a String representation of the condition. Enums define toString to return a String representation of their predefined values. Demonstrate your overloaded method works by creating some BicycleListings and printing them to the console.

** STOP **

Ensure you have saved the source files in your project. Make a copy of your Eclipse project at this point (you can do this from your system's file explorer). Name the file labelling it as Task 1. You will submit this copy at the end of this lab.

`equals()`

This method implements an equivalence relation on non-null object references. For any non-null reference values x, y, and z the following should be true.

- Reflexive: x.equals(x) is true.
- Symmetric: x.equals(y) is true if and only if y.equals(x) is true.
- Transitive: If x.equals(y) is true, and y.equals(z) is true, then x.equals(z) is true.
- Consistent: Multiple invocations if x.equals(y) should consistently return true or consistently return false provided none of the information used in equals comparison is changed.
- x.equals(null) is false.

The `equals()` method returns a boolean value indicating whether two objects are equal. The default behavior of `equals(Object)` compares the memory addresses of the two objects.

We can change this behavior by overriding `equals()` for our own classes.

```java
@Override
public boolean equals(Object obj) {

    // Check for self-comparison
    if (this == obj) {
        return true;
    }
    // Check if obj is an instance of Bicycle
    else if (!(obj instanceof Bicycle)) {
        return false;
    }
    // obj is a Bicycle instance; do comparison
    else {
        Bicycle rhs = (Bicycle) obj;
        return (this.serialNo == rhs.serialNo);
    }
}
```

Figure 1 equals overridden for Bicycle.

The procedure for comparison is as follows:

1) Check for self-comparison.
    a. If the object argument for equals is the same as **`this`**, return true.
2) Check if the given object is an instance of the proper type.
    a. Use the **`instanceof`** operator to check if the object argument is the proper type.
    b. If it is not, return false.
3) If neither of the above two cases are true, we can do a comparison.
    a. Cast the argument from Object to the appropriate class.
    b. Return true or false according to the definition of equality for the class.

The **`instanceof`** operator is a binary operator which returns a Boolean value. The left operand is an object, and the right operand is a type. It returns true if the object on the left is an instance of the type on the right. The equals() method is shown overridden for Bicycle in Figure 1.

Graded Submission Task 2

Override the `equals()` method for BicycleListing so that objects of type BicycleListing are equal iff their Bicycles are equal and their prices are equal and their conditions are equal.

Bicycles are equal iff they have the same serialNo. Demonstrate this works by creating BicycleListings and testing the 3 cases in the method (self-comparison, type, comparison).

** STOP **

Ensure you have saved the source files in your project. Make a copy of your Eclipse project at this point (you can do this from your system's file explorer). Name the file labelling it as Task 2. You will submit this copy at the end of this lab.

hashCode()

When equals() is overridden, it is generally necessary to also override the hashCode() method. This method returns a hash code value for the object. The default implementation converts the internal address of the object to an integer.

It is generally necessary because of the general contract of the hashCode() method. This states that equal objects must have equal hash codes. Generally, this can be achieved by using the same fields that are used to check equality to also calculate the hash code. There are many ways to calculate the hash value, some are better suited for different applications. In this course, it will be sufficient to perform simple arithmetic with the hashes of the fields, or using the static method Objects.hash(Object…) to generate the hash code.

Lab wrap-up

In this lab we discussed the methods toString() and equals(). These methods are part of the Object class, which is the base class of all Java classes (directly or indirectly). toString() is used to return a String representation of the object. The default implementation prints the class type and its hashcode. toString() is implicitly called when an object is printed.

equals() returns a boolean indicating whether two objects are equal. The default behavior of this method compares the memory addresses of the two objects. We should take some things into consideration when implementing equals(). First, it takes an Object object as a parameter. This means it will not have access to the necessary fields to make a proper comparison. We will need to cast the argument into the appropriate type. The procedure for equals() is this: first, check for self-comparison. If an object is being compared to itself, return true immediately. Next, check if the object is as instance of the appropriate class. You can do this using the instanceof operator. If it is not, return false. Otherwise, cast the argument to the appropriate class and perform the comparison.

Submission notes for graded tasks:

- Submit a zipped folder containing the Eclipse projects for each graded task.
- For each project, ensure you have included the src folder containing the .java file, the .classpath file, and the .project file.
- Ensure that each project is named appropriately to correspond with each task.
- Zip the projects together and name the zipped folder “Lab14_firstName_lastName”.
- Submit the zipped file to the respective Canvas link.

Rubrics	
Task 1 (points allotted) - Compilation (2) - toString overridden to return string in proper format (4) - Driver demonstrates toString works properly (4)	Task 2 (points allotted) - Compilation (2) - Check for self-comparison (2) - Check for correct type (2) - Method evaluates equality correctly (2) - Driver demonstrates equals() is implemented correctly (2)