Debugging Classes

by Sophia

WHAT'S COVERED

In this lesson, you will learn about debugging classes when using Java. Specifically, this lesson covers:

1. Implementing toString() to Check an Object's State

In previous lessons about debugging arrays and collections, you have seen how the `toString()` methods provided in Java's standard libraries can facilitate tracking down and fixing problems. Using the `toString()` method with objects can be helpful in debugging, by allowing us to see the values currently stored in the object (its state).

All Java objects have a default `toString()` method, but this default version may not be very helpful with classes that you create. Let's take a look at the `UserAccount` class that you have worked with previously.

Here is the code (which needs to be in a file called `UserAccount.java`):

```
import java.time.LocalDate;
public class UserAccount {
  private String userName;
  private String password;
  private LocalDate dateJoined;
  private boolean activeUser;

  public UserAccount(String userName, String password) {
    this.userName = userName;
    this.password = password;
    this.dateJoined = LocalDate.now();
```

```
    this.activeUser = true;
  }

  // Allow read-only access to user name
  public String getUserName() {
    return userName;
  }

  // Allow read-only access to date joined
  public LocalDate getDateJoined() {
    return dateJoined;
  }

  // Allow activeUser to be read & set (can change)
  public boolean isActiveUser() {
    return activeUser;
  }

  public void setActiveUser(boolean activeUser) {
    this.activeUser = activeUser;
  }
}
```

Here is a simple driver class that makes use of the `UserAccount` class. Type this code into a file named EmployeeExample.java:

```
public class UserAccountExample {
  public static void main(String[] args) {
    UserAccount account = new UserAccount("Sophia2", "TestTest123");
    System.out.println("Result from calling account.toString():");
    System.out.println(account.toString());
  }
}
```

Run the `UserAccount` example program. The results should look like this:

```
Result from calling account.toString():
UserAccount@b3d7190
```

HINT

The exact **hexadecimal** (base 16) number in the last line of the output will vary from computer to computer, but that doesn't matter because it doesn't tell a human reader anything useful.

Here is the code for a `toString()` method to be added to the `UserAccount` class. This code produces a `String` with the names of the attributes and their current values:

```
public String toString() {
  String state = "UserName: " + userName + "\n";
  state += "password: " + password + "\n";
  state += "dateJoined: " + dateJoined + "\n";
  state += "activeUser: " + activeUser + "\n";
  return state;
}
```

The results of running the UserAccountExample program with this version of the `UserAccount` class produces this output:

```
Result from calling account.toString():
UserName: Sophia2
password: TestTest123
dateJoined: 2024-04-29
activeUser: true
```

This version of `toString()` provides a much more useful representation of the data in the object. The format of the date (year - month - day) may be a bit unexpected, but it is the default format. You can use another Java format class to put the date into a more familiar format. First, the code (in UserAccount.java) needs to add an import at the top of the file (with the other import for `java.time.LocalDate`):

EXAMPLE

```
import java.time.format.DateTimeFormatter;
```

Then, in the body of the toString() method, add this statement to create the desired format:

EXAMPLE

```
DateTimeFormatter dateFormat = DateTimeFormatter.ofPattern("MM/dd/YYYY");
```

The method used to create a `DateTimeFormatter` is a bit different from how the constructor for `DecimalFormat` is used, but as you can hopefully make out, the code above formats the date as month/day/year. Calling `dateFormat`'s `format()` method outputs the date in the specified format. The `toString()` method should now look like this:

```
public String toString() {
  DateTimeFormatter dateFormat = DateTimeFormatter.ofPattern("MM/dd/YYYY");
```

```
  String state = "UserName: " + userName + "\n";
  state += "password: " + password + "\n";
  state += "dateJoined: " + dateFormat.format(dateJoined) + "\n";
  state += "activeUser: " + activeUser + "\n";
  return state;
}
```

The output from the program should now look like this:

```
Result from calling account.toString():
UserName: Sophia2
password: TestTest123
dateJoined: 04/29/2024
activeUser: true
```

Let's now look at an example of how this method might be useful. The constructor for the UserAccount class includes a line that sets the `activeUser` attribute to true, but let's imagine the programmer has made an error and the constructor looks like this instead:

EXAMPLE

```
public UserAccount(String userName, String password) {
 this.userName = userName;
 this.password = password;
 this.dateJoined = LocalDate.now();
}
```

Since activeUser is a boolean, the default value is false, if it is not initialized to true. If the code in the application's `main()` were like this, the result would not be what you expected for a newly created account:

```
public class UserAccountExample {
  public static void main(String[] args) {
    UserAccount account = new UserAccount("Sophia2", "TestTest123");
    if(account.isActiveUser()) {
      System.out.println(account.getUserName() + " is active.");
    }
    else {
      System.out.println(account.getUserName() + " is not active.");
    }
  }
}
```

The results from running this code look like this:

```
Sophia2 is active.
```

This is not what is expected, so adding a call to the UserAccount object's toString() method can help us figure out the cause of the trouble:

```
public class UserAccountExample {
  public static void main(String[] args) {
    UserAccount account = new UserAccount("Sophia2", "TestTest123");
    if(account.isActiveUser()) {
      System.out.println(account.getUserName() + " is active.");
    }
    else {
      System.out.println(account.getUserName() + " is not active.");
    }
    // Added for debugging
    System.out.println("\nDebugging info:\n" + account.toString());
  }
}
```

Running the program now produces the following output:

```
Sophia2 is active.

Debugging info:
UserName: Sophia2
password: TestTest123
dateJoined: 04/29/2024
activeUser: true
```

The data displayed by the toString() method shows that the activeUser is set to false (the default value for a boolean variable in Java) since it is not assigned the value true by the constructor.

Putting back the line:

EXAMPLE

```
this.activeUser = true;
```

This will get the program working as intended.

TERM TO KNOW

Hexadecimal
A value expressed in a base 16 number system (rather than base 10).

SUMMARY

In this lesson, you have learned how to **implement a toString()** method in a class that you have created, **to check an object's state**. As you saw when working with arrays and collections, the toString() method can be very helpful in tracking down and fixing problems that arise through programming mistakes and other errors.

Source: This content and supplemental material has been adapted from Java, Java, Java: Object-Oriented Problem Solving. Source cs.trincoll.edu/~ram/jjj/jjj-os-20170625.pdf

It has also been adapted from “Python for Everybody” By Dr. Charles R. Severance. Source py4e.com/html3/

TERMS TO KNOW

Hexadecimal
A value expressed in a base 16 number system (rather than base 10).