Java Objects and Text Files

1. Overview

In this short tutorial, we'll look at how to use Java objects and text files. In other words, we'll see how to write in and read objects from text files using Java.
To learn how to read and write in binary files, we have the Binary Files in Java article. Furthermore, to learn how to read and write in text files, we have the Text Files in Java article.

2. Write Objects in a File

When dealing with textual files and objects, we should be aware that we cannot easily store the entire object in the text file. To store entire objects, we can use bytes of stream and binary files.

Furthermore, the text files contain only string values. In other words, whatever we store will be stored as a string. Therefore, we'd get the String values back from a file.

Now, we'll use the PrintWriter class and its println() method to write data in the text file. Using PrintWriter , we can store the String representation of an object. If we pass the entire object as an argument of the println() method, Java will call the toString() method on our object.

Firstly, let's create a class Order that will represent the data we'll going to write in the file:

```
class Order {
    private UUID orderNumber;
    private double price;
    public Order(UUID orderNumber, double price) {
        this.orderNumber = orderNumber;
        this.price = price;
    }

    // getters and setters
}
```

The println() method will write a String representation of an object. We need to override the toString() method inside our Order class.
We can use “;” as a delimiter for our values:

```
@Override
public String toString() {
    return orderNumber + ";" + price;
}
```

Secondly, we’ll create a new object of the Order class:

```
Order order = new Order(UUID.randomUUID(), 50.85);
```

Finally, let’s create our PrintWriter instance and call the println() method:

```
try (FileWriter fileWriter = new FileWriter(filePath, true);
     PrintWriter printWriter = new PrintWriter(fileWriter)) {
    printWriter.println(order);
} catch (Exception e) {
    System.err.println("Error - " + e);
}
```

When we execute the code, the following data will appear in the file:

```
cab5d1ae-0730-4e19-8c4d-c63fbfec0849;50.85
```

Each value is separated by the delimiter. Using a specific delimiter can come in handy if we need to read values from the text file.

3. Read Objects from File Using BufferedReader Class

We can read objects from a file using BufferedReader class. Lines in text files are represented as string values. Inside our text file, each line contains data of one object. Moreover, an object’s attributes are separated using “;” as a delimiter.

Now, let’s create an instance of BufferReader class:

```
BufferedReader br = new BufferedReader(new FileReader(filePath));
```

Using the while statement we can read each line from the file:

```
List<Order> orders = new ArrayList<>();
while ((row = br.readLine()) != null) {
    System.out.println(row);
    String[] data = row.split(";");
    UUID orderNumber = UUID.fromString(data[0]);
    String price = data[1];
    orders.add(new Order(orderNumber, Double.parseDouble(price)));
}
```

The while statement will be executed until lines in the file exist. We can split each line using a delimiter. We can create a new instance of an Order class and add it to the list.

4. Conclusion

In this tutorial, we've explained how objects can be stored and read from textual files.
As always, the entire code of this example can be found over on GitHub.
Read More