Practices for Lesson 13: Java I/O Fundamentals

Chapter 13

Practices for Lesson 13: Overview

Practices Overview

In these practices, you will use some of the java.io classes to read from the console, open and read files, and serialize and deserialize objects to and from the file system.

Practice 13-1: Summary Level: Writing a Simple Console I/O Application

Overview

In this practice, you will write a simple console-based application that reads from and writes to the system console. In NetBeans, the console is opened as a window in the IDE.

Tasks

1. Open the project `FileScanner13-01Prac` in the following directory:
   `/home/oracle/labs/13-IO_Fundamentals/practices/practice1`
2. Open the file `FileScanInteractive.java`.

   Notice that the class has a method called `countTokens` already written for you. This method takes a String `file` and String `search` as parameters. The method will open the file name passed in and use an instance of a `Scanner` to look for the search token. For each token encountered, the method increments the integer field `instanceCount`. When the file is exhausted, it returns the value of `instanceCount`. Note that the class rethrows any `IOException` encountered, so you will need to be sure to use this method inside a try-catch block.
3. Code the main method to check the number of arguments passed. The application expects at least one argument (a string representing the file to open). If the number of arguments is less than one, exit the application with an error code (-1).
   a. The main method is passed an array of Strings. Use the `length` attribute to determine whether the array contains less than one argument.
   b. Print a message if there is less than one argument, and use `System.exit` to return an error code. (-1 typically is used to indicate an error.)
4. Save the first argument passed into the application as a `String`.
5. Create an instance of the `FileScanInteractive` class. You will need this instance to call the `countTokens` method.
6. Open the system console for input using a buffered reader.
   a. Use a `try`-with-resources to open a `BufferedReader` chained to the system console input. (Recall that `System.in` is an input stream connected to the system console.)
   b. Be sure to add a catch statement to the try block. Any exception returned will be an `IOException` type.
   c. In a while loop, read from the system console into a string until the string “q” is entered on the console by itself.
      Note: You can use `equalsIgnoreCase` to allow your users to enter an upper- or lowercase “Q.” Also the `trim()` method is a good choice to remove any whitespace characters from the input.
   d. If the string read from the console is not the terminate character, call the `countTokens` method, passing in the file name and the search string.
   e. Print a string indicating how many times the search token appeared in the file.
   f. Add any missing import statements.
7. Save the `FileScanInteractive` class.

8. If you have no compilation errors, you can test your application by using a file from the resources directory.
   a. Right-click the project and select Properties.
   b. Click Run.
   c. Enter the name of a file to open in the Arguments text box, for example: `/home/oracle/labs/resources/DeclarationOfIndependence.txt`
   d. Click OK.
   e. Run the application and try searching for some words like `when`, `rights`, and `free`. Your output should look something like this:

```
Searching through the file:
/home/oracle/labs/resources/DeclarationOfIndependence.txt
Enter the search string or q to exit: when
The word "when" appears 3 times in the file.
Enter the search string or q to exit: rights
The word "rights" appears 3 times in the file.
Enter the search string or q to exit: free
The word "free" appears 4 times in the file.
Enter the search string or q to exit: q
BUILD SUCCESSFUL (total time: 16 seconds)
```

Practice 13-1: Detailed Level: Writing a Simple Console I/O Application

Overview

In this practice, you will write a simple console-based application that reads from and writes to the system console. In NetBeans, the console is opened as a window in the IDE.

Tasks

1. Open the project `FileScanner13-01Prac`.
   a. Select File > Open Project.
   b. Browse to `/home/oracle/labs/13-IO_Fundamentals/practices/practice1`.
   c. Select `FileScanner13-01Prac` and select the "Open as Main Project" check box.
   d. Click the Open Project button.
2. Open the file `FileScanInteractive.java`.

   Notice that the class has a method called `countTokens` already written for you. This method takes a String `file` and String `search` as parameters. The method will open the file name passed in and use an instance of a `Scanner` to look for the search token. For each token encountered, the method increments the integer field `instanceCount`. When the file is exhausted, it returns the value of `instanceCount`. Note that the class rethrows any `IOException` encountered, so you will need to be sure to use this method inside a try-catch block.
3. Code the `main` method to check the number of arguments passed.

   The application expects at least one argument (a string representing the file to open). If the number of arguments is less than one, exit the application with an error code (-1).
   a. The `main` method is passed an array of Strings. Use the `length` attribute to determine whether the array contains less than one argument.
   b. Print a message if there is less than one argument, and use `System.exit` to return an error code. (-1 typically is used to indicate an error.) For example:

```
if (args.length < 1) {
    System.out.println("Usage: java FileScanInteractive <file to
search>");
    System.exit(-1);
}
```

4. Save the first argument passed into the application as a `String`.

```
String file = args[0];
```

5. Create an instance of the `FileScanInteractive` class. You will need this instance to call the `countTokens` method.

```
FileScanInteractive scan = new FileScanInteractive ();
```

6. Open the system console for input using a buffered reader.
   a. Use a `try`-with-resources to open a `BufferedReader` chained to the system console input. (Recall that `System.in` is an input stream connected to the system console.)
   b. Be sure to add a catch statement to the try block. Any exception returned will be an `IOException` type. For example:

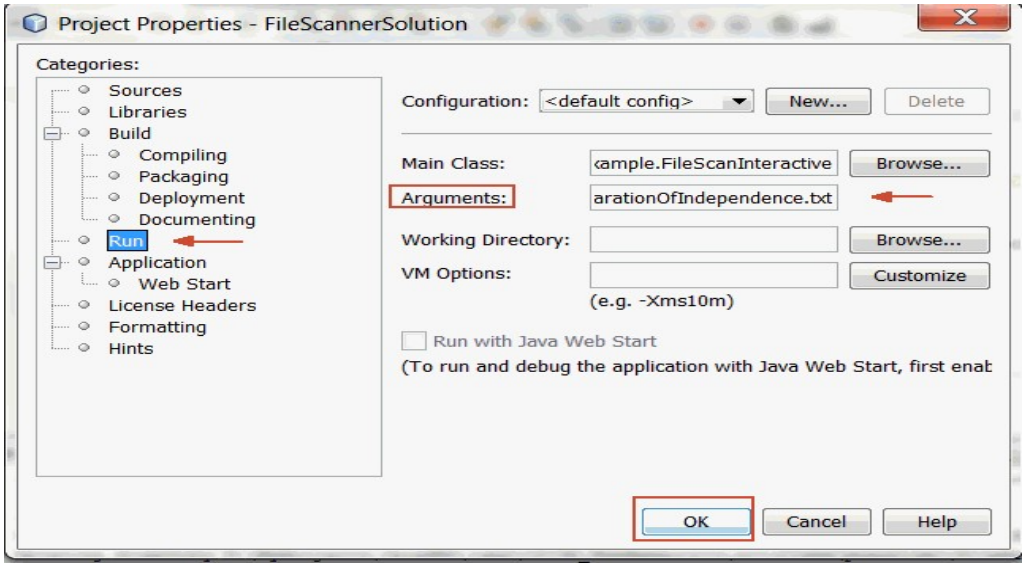
```
try (BufferedReader in =
    new BufferedReader(new InputStreamReader(System.in))) {

} catch (IOException e) { // Catch any IO exceptions.
    System.out.println("Exception: " + e);
    System.exit(-1);
}
```

   c. In the try block that you created, add a while loop. The while loop should run until a break statement. Inside the while loop, read from the system console into a string until the string "q" is entered on the console by itself.
   Note: You can use `equalsIgnoreCase` to allow your users to enter an upper- or lowercase "Q." Also the `trim()` method is a good choice to remove any whitespace characters from the input.
   d. If the string read from the console is not the terminate character, call the `countTokens` method, passing in the file name and the search string.
   e. Print a string indicating how many times the search token appeared in the file.
   f. Your code inside the try block should look something like this:

```
String search = "";
System.out.println ("Searching through the file: " + file);
while (true) {
    System.out.print("Enter the search string or q to exit: ");
    search = in.readLine().trim();
    if (search.equalsIgnoreCase("q")){
     break;
    }
    int count = scan.countTokens(file, search);
    System.out.println("The word \"" + search + "\" appears "
                        + count + " times in the file.");
}
```

   g. Add any missing import statements.
7. Save the `FileScanInteractive` class.

8. If you have no compilation errors, you can test your application by using a file from the resources directory.
   a. Right-click the project and select Properties.
   b. Select Run from the Categories column.
   c. Enter the name of a file to open in the Arguments text box, for example: `/home/oracle/labs/resources/DeclarationOfIndependence.txt`
   d. Click OK.



   e. Run the application and try searching for some words like `when`, `rights`, and `free`. Your output should look something like this:

```
Searching through the file:
/home/oracle/labs/resources/DeclarationOfIndependence.txt
Enter the search string or q to exit: when
The word "when" appears 3 times in the file.
Enter the search string or q to exit: rights
The word "rights" appears 3 times in the file.
Enter the search string or q to exit: free
The word "free" appears 4 times in the file.
Enter the search string or q to exit: q
BUILD SUCCESSFUL (total time: 16 seconds)
```

Practice 13-2: Summary Level: Serializing and Deserializing a `ShoppingCart`

Overview

In this practice, you use the `java.io.ObjectOutputStream` class to write a Java object to the file system (serialize), and then use the same stream to read the file back into an object reference. You will also customize the serialization and deserialization of the `ShoppingCart` object.

Tasks

1. Open the `SerializeShoppingCart13-02Prac` project in the directory:
   `/home/oracle/labs/13-IO_Fundamentals/practices/practice2`
2. Expand the `com.example.test` package. Notice there are two Java main classes in this package, `SerializeTest` and `DeserializeTest`. You will be writing the code in these main classes to serialize and deserialize `ShoppingCart` objects.
3. Open the `SerializeTest.java`. You will write the methods in this class to write several `ShoppingCart` objects to the file system.
   a. Read through the code. You will note that the class prompts for the cart ID and constructs an instance of `ShoppingCart` with the cart ID in the constructor.
   b. The code then adds three `Item` objects to the `ShoppingCart`.
   c. The code then prints out the number of items in the cart, and the total cost of the items in the cart. Look through the `ShoppingCart` and `Item` classes in the `com.example.domain` package for details on how these classes work.
   d. You will be writing the code to open an `ObjectOutputStream` and write the `ShoppingCart` as a serialized object on the file system.
4. Create the try block to open a `FileOutputStream` chained to an `ObjectOutputStream`. The file name is already constructed for you.
   a. Your code will go where the comment line is at the bottom of the file.
   b. Open a `FileOutputStream` with the `cartFile` string in a `try`-with-resources block.
   c. Pass the file output stream instance to an `ObjectOutputStream` to write the serialized object instance to the file.
   d. Write the `cart` object to the object output stream instance by using the `writeObject` method.
   e. Be sure to catch any `IOException` and exit with an error as necessary.
   f. Add a success message before the method ends:
      ```
      System.out.println ("Successfully serialized shopping cart with
      ID: " + cart.getCartID());
      ```
   g. Save the file.
5. Open the `DeserializeTest.java`. The main method in this class reads from the console for the ID of the customer shopping cart to deserialize.

6. Your code will go where the comment line is at the bottom of the file.
   a. Open a `FileInputStream` with the `cartFile` string in a `try`-with-resources block.

b. Pass the file input stream instance to an `ObjectOutputStream` to read the serialized object instance from the file.
c. Read the `cart` object from the object input stream using the `readObject` method. Be sure to cast the result to the appropriate object type.
d. You will need to catch both `ClassNotFoundException` and `IOException`, so use a multi-catch expression.
e. Finally, print out the results of the `cart` (all of its contents) and the `cart` total cost using the following code:

```
System.out.println ("Shopping Cart contains: ");
List<Item> cartContents = cart.getItems();
for (Item item : cartContents) {
    System.out.println (item);
}
System.out.println ("Shopping cart total: " +
NumberFormat.getCurrencyInstance().format(cart.getCartTotal()));
```

f. Save the file.

7. Open the `ShoppingCart.java`. You will customize the serialization and deserialization of this class by adding the two methods called during serialization/deserialization.
   a. Add a `writeObject` method invoked during serialization. This method should serialize the current object fields and then add a timestamp (`Date` object instance) to end of the object stream.
8. Add a method to the `ShoppingCart` class that is invoked during deserialization.
   a. Add a `readObject` method with the appropriate signature. This method will recalculate the total cost of the shopping cart and print the timestamp that was added to the stream.
   b. Save the file.
9. Test the application. This application has two main methods, so you will need to run each main in turn.
   a. To run the `SerializeTest.java`, right-click the class name and select Run File.
   b. The output will look like this:

```
Enter the ID of the cart file to create and serialize or q exit.
101
Shopping cart 101 contains 3 items
Shopping cart total: $58.39
Successfully serialized shopping cart with ID: 101
```

   c. To run the `DeserializeTest.java`, right-click the class name and select Run File.

d. Enter the ID `101` and the output will look like something this:

```
run:
Enter the ID of the cart file to deserialize or q exit.
101
Restored Shopping Cart from: Apr 16, 2014
Successfully deserialized shopping cart with ID: 101
Shopping cart contains:
Item ID: 101 Description: Duke Plastic Circular Flying Disc Cost: 10.95
Item ID: 123 Description: Duke Soccer Pro Soccer ball Cost: 29.95
Item ID: 45 Description: Duke "The Edge" Tennis Balls - 12-Ball Bag Cost: 17.49
Shopping cart total: $58.39
BUILD SUCCESSFUL (total time: 4 seconds)
```

Practice 13-2: Detailed Level: Serializing and Deserializing a `ShoppingCart`

Overview

In this practice, you use the `java.io.ObjectOutputStream` class to write a Java object to the file system (serialize), and then use the same stream to read the file back into an object reference. You will also customize the serialization and deserialization of the `ShoppingCart` object.

Tasks

1. Open the `SerializeShoppingCart13-02Prac` project in the `/home/oracle/labs/13-IO_Fundamentals/practices/practice2` directory.
   a. Select File > Open Project.
   b. Browse to the `/home/oracle/labs/13-IO_Fundamentals/practices/practice2` directory.
   c. Select the project `SerializeShoppingCart13-02Prac`.
   d. Click Open Project.
2. Expand the `com.example.test` package. Notice there are two Java main classes in this package, `SerializeTest` and `DeserializeTest`. You will be writing the code in these main classes to serialize and deserialize `ShoppingCart` objects.
3. Open the `SerializeTest.java`. You will write the methods in this class to write several `ShoppingCart` objects to the file system.
   a. Read through the code. You will note that the class prompts for the cart ID and constructs an instance of `ShoppingCart` with the cart ID in the constructor.
   b. The code then adds three `Item` objects to the `ShoppingCart`.
   c. The code then prints out the number of items in the cart, and the total cost of the items in the cart. Look through the `ShoppingCart` and `Item` classes in the `com.example.domain` package for details on how these classes work.
   d. You will be writing the code to open an `ObjectOutputStream` and write the `ShoppingCart` as a serialized object on the file system.
4. Create the try block to open a `FileOutputStream` chained to an `ObjectOutputStream`. The file name is already constructed for you.
   a. Your code will go where the comment line is at the bottom of the file.
   b. Open a `FileOutputStream` with the `cartFile` string in a `try`-with-resources block.
   c. Pass the file output stream instance to an `ObjectOutputStream` to write the serialized object instance to the file.
   d. Write the `cart` object to the object output stream instance by using the `writeObject` method.
   e. Be sure to catch any `IOException` and exit with an error as necessary.

f. Your code might look like this:

```
try (FileOutputStream fos = new FileOutputStream (cartFile);
     ObjectOutputStream o = new ObjectOutputStream (fos)) {
    o.writeObject(cart);
} catch (IOException e) {
    System.out.println ("Exception serializing " + cartFile + ":
" + e);
    System.exit (-1);
}
```

g. Add a success message before the method ends:

```
System.out.println ("Successfully serialized shopping cart with
ID: " + cart.getCartID());
```

h. Add any missing import statements.

i. Save the file.

5. Open `DeserializeTest.java`. The main method in this class reads from the console for the ID of the customer shopping cart to deserialize.

6. Your code will go where the comment line is at the bottom of the file.

   a. Open a `FileInputStream` with the `cartFile` string in a `try`-with-resources block.

   b. Pass the file input stream instance to an `ObjectInputStream` to read the serialized object instance from the file.

   c. Read the `cart` object from the object input stream using the `readObject` method. Be sure to cast the result to the appropriate object type.

   d. You will need to catch both `ClassNotFoundException` and `IOException`, so use a multi-catch expression.

   e. Your code should look like this:

```
try (FileInputStream fis = new FileInputStream (cartFile);
     ObjectInputStream in = new ObjectInputStream (fis)) {
    cart = (ShoppingCart)in.readObject();
} catch (ClassNotFoundException | IOException e) {
    System.out.println ("Exception deserializing " + cartFile +
": " + e);
    System.exit (-1);
}
System.out.println ("Successfully deserialized shopping cart
with ID: " + cart.getCartID());
```

f. Finally, print out the results of the `cart` (all of its contents) and the `cart` total cost using the following code:

```
System.out.println ("Shopping cart contains: ");
List<Item> cartContents = cart.getItems();
for (Item item : cartContents) {
    System.out.println (item);
}
System.out.println ("Shopping cart total: " +
NumberFormat.getCurrencyInstance().format(cart.getCartTotal()));
```

g. Save the file.

7. Open the `ShoppingCart.java`. You will customize the serialization and deserialization of this class by adding the two methods called during serialization/deserialization.
   a. Add a method invoked during serialization that will add a timestamp (Date object instance) to the end of the object stream.
   b. Add a method with the signature:

```
private void writeObject(ObjectOutputStream oos) throws
IOException {
```

   c. Make sure that the method serializes the current object fields first, and then write the Date object instance:

```
    oos.defaultWriteObject();
    oos.writeObject(new Date());
}
```

8. Add a method to the `ShoppingCart` class that is invoked during deserialization. This method will recalculate the total cost of the shopping cart and print the timestamp that was added to the stream.
   a. Add a method with the signature:

```
private void readObject(ObjectInputStream ois) throws
IOException, ClassNotFoundException {
```

   b. This method will deserialize the fields from the object stream, and recalculate the total dollar value of the current cart contents:

```
    ois.defaultReadObject();
    if (cartTotal == 0 && (items.size() > 0)) {
        for (Item item : items)
                cartTotal += item.getCost();
    }
```

   c. Get the Date object from the serialized stream and print the timestamp to the console.

```
    Date date = (Date)ois.readObject();
    System.out.println ("Restored Shopping Cart from: " +
DateFormat.getDateInstance().format(date));
}
```

   d. Save the ShoppingCart.

9. Test the application. This application has two main methods, so you will need to run each main in turn.
   a. To run the `SerializeTest.java`, right-click the class name and select Run File. Enter a cart ID, such as 101.
   b. The output will look like this:

```
Enter the ID of the cart file to create and serialize or q exit.
101
Shopping cart 101 contains 3 items
Shopping cart total: $58.39
Successfully serialized shopping cart with ID: 101
```

   c. To run the `DeserializeTest.java`, right-click the class name and select Run File.
   d. Enter the ID `101` and the output will look like this:

```
Enter the ID of the cart file to deserialize or q exit.
101
Restored Shopping Cart from: Oct 26, 2011
Successfully deserialized shopping cart with ID: 101
Shopping cart contains:
Item ID: 101 Description: Duke Plastic Circular Flying Disc
Cost: 10.95
Item ID: 123 Description: Duke Soccer Pro Soccer ball Cost:
29.95
Item ID: 45 Description: Duke "The Edge" Tennis Balls - 12-Ball
Bag Cost: 17.49
Shopping cart total: $58.39
```