Side Topic
File Input/Output in Java

An introduction to persistent data.

Research Assignment: Java Files API

Because we don’t want to manually enter long lists of data.

ResearchMe.txt

- Find the answers to the following questions:
  - How do you **READ** a text file using Java?
  - How do you **WRITE** to a text file using Java?
  - How would you read a list of items from a text file and load them into an array?
  - How would you write the updates to the list back to the text file?

File New I/O Java API (JDK 7 and above)

- From the **`java.io`** package
  - `BufferedReader`
  - `BufferedWriter`
  - `IOException`

- From the **`java.nio`** package
  - `file` subpackage
    - `Files (static)`
    - `Path`
    - `Paths (static)`
  - `charset` subpackage
    - `Charset`
    - `StandardCharsets (static)`

Reading a Text File

- Create a method **`openFile`** that accepts a String `filename`
- You may use the following template code to read a text file:

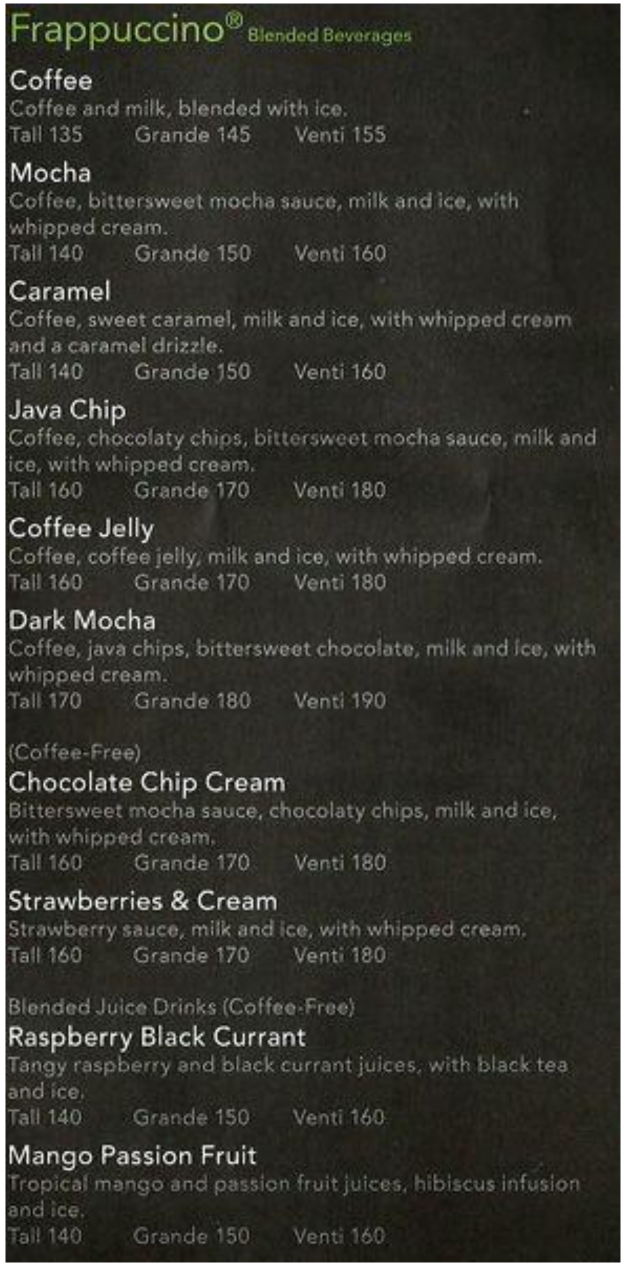
```
Path path = Paths.get(filename);
Charset cs = StandardCharsets.US_ASCII;
// you can also use UTF_8

try (BufferedReader reader =
Files.newBufferedReader(path, cs)){
      String line = null;
      while((line = reader.readLine()) != null){
            // do something with line
      }
} catch(IOException x){
      System.err.println(x);
}
```

Design the format of the text data

- Encode some of these menu items into a text file (at least five).
- You can create your own format.

  Hint: Make the position of the different data (name, description, etc.) isolated and consistent so reading the file is easier
- `Double.parseDouble()` converts a String into a double value.



Frappuccino® Blended Beverages

Coffee
Coffee and milk, blended with ice.
Tall 135 Grande 145 Venti 155

Mocha
Coffee, bittersweet mocha sauce, milk and ice, with whipped cream.
Tall 140 Grande 150 Venti 160

Caramel
Coffee, sweet caramel, milk and ice, with whipped cream and a caramel drizzle.
Tall 140 Grande 150 Venti 160

Java Chip
Coffee, chocolaty chips, bittersweet mocha sauce, milk and ice, with whipped cream.
Tall 160 Grande 170 Venti 180

Coffee Jelly
Coffee, coffee jelly, milk and ice, with whipped cream.
Tall 160 Grande 170 Venti 180

Dark Mocha
Coffee, java chips, bittersweet chocolate, milk and ice, with whipped cream.
Tall 170 Grande 180 Venti 190

(Coffee-Free)

Chocolate Chip Cream
Bittersweet mocha sauce, chocolaty chips, milk and ice, with whipped cream.
Tall 160 Grande 170 Venti 180

Strawberries & Cream
Strawberry sauce, milk and ice, with whipped cream.
Tall 160 Grande 170 Venti 180

Blended Juice Drinks (Coffee-Free)

Raspberry Black Currant
Tangy raspberry and black currant juices, with black tea and ice.
Tall 140 Grande 150 Venti 160

Mango Passion Fruit
Tropical mango and passion fruit juices, hibiscus infusion and ice.
Tall 140 Grande 150 Venti 160

Sample Format

```
<number of items>
<item 1 name>
<item 1 description>
<number of item 1 sizes>
<item 1 size 1>
...
<item 1 size n>
<item 1 price 1>
...
<item 1 price n>
...
<item n name>
...
```

```
5
Coffee
Coffee and milk blended with ice
3
Tall
Grande
Venti
135
145
155
Mocha
Coffee, bittersweet mocha sauce...
...
```

You are free to modify this or make your own.

Why that format?

- It's similar to how we would normally prompt the user for data when we used the console. The process of getting the data should be more familiar.

- Let's try an easier example:

Ask the user for a number **n**. Then, ask the user for **n** numbers and store all these values in int array. Print out the values in ascending order.

A familiar sample:

- Normally, it would go something like this:

```
System.out.print(“Enter a number:”);
int n = sc.nextInt();
System.out.println(“Enter ”+n+” integers:”);
int[] intArr = new int[n];
for(int i = 0; i < n; i++)
        intArr[i] = sc.nextInt();
Arrays.sort(intArr);
for(int i = 0; i < n; i++)
        System.out.print(intArr[i]+”,”);
```

A familiar sample:

- A sample of what a user would see is shown below, note that the orange values were those typed by the user:

```
Enter a number: 5
Enter 5 integers:
9
5
8
2
3
2,3,5,8,9,
```

Comparing the input data

In the Console

```
Enter a number: 5
Enter 5 integers:
9
5
8
2
3
```

In a Text File

```
5
9
5
8
2
3
```

Unlike the console version, we can format the file to provide all the data we need from the user.

Comparing the data input code

In the Console

```
System.out.print("Enter a number:");
int n = sc.nextInt();
System.out.println("Enter "+n+" integers:");
int[] intArr = new int[n];
for(int i = 0; i < n; i++)
  intArr[i] = sc.nextInt();
```

In a Text File

```
Path path = Paths.get(filename);
Charset cs = StandardCharsets.US_ASCII;
int[] intArr = null;

try (BufferedReader reader =
Files.newBufferedReader(path, cs)){
  String line = null;
  int n = Integer.parseInt(reader.readLine());
  intArr = new int[n];
  int i = 0;
  while((line = reader.readLine()) != null && i < n){
    intArr[i] = Integer.parseInt(line);
    i++;
  }
} catch(IOException x){
  System.err.println(x);
}
```

Notice the similarities?

What have we learned?

- We can replace repetitive user input with a persistent text file so that we don't have to re-enter the values every time we run the program.
- `bufferedReader.readLine()` is like `scanner.nextLine()`
- It's easier to read the data from the text file if we use the same process and input sequence that we would have used for the console.

Loading the data

- Remember the arrays we used last meeting (String[] name, String[] description, etc.)? For now, make them static fields of your java class.
    - E.g., `public static String[] name;`
- Modify the **`openFile`** method so that it stores the menu data into the arrays. You should be able to access the (now static field) array variables directly from this method.

How a **field/attribute** works

```
public class Sample{
  public static int[] aFieldArray;

  public static void aMethod(){
    System.out.println(“aMethod: I can access aFieldArray directly!");
    aFieldArray = new int[2];
    aFieldArray[0] = 20;
  }

  public static void main(String[] args){
    System.out.println(“Main: I can call aMethod()");
    aMethod();
    System.out.println(“Main: I can access aFieldArray directly, too!");
    aFieldArray[1] = 13;
    System.out.println(“Main: Year "+aFieldArray[0]+aFieldArray[1]);
  }
}
```

Just like static methods can call other static methods, they can now also access this array since it is a static field.

For now, only important data containers are allowed to be fields.
Do **not** make scanners, temporary variables, or loop counters fields.

Checkpoint

- ✓Create a standard formatted text file
- ✓Read the file
- ✓Save the data in the correct arrays
- ✓Perform operations on the menu items

Writing to a Text File

- Writing to a text file is actually very similar to reading

```
try (BufferedWriter writer =
Files.newBufferedWriter(path, cs)){
      for(String data : yourSourceOfData){
            writer.write(data);
            writer.newline();
      }
} catch(IOException x){
      System.err.println(x);
}
```