Week 13: Computer Science 1

File Reading and Writing

File Reading and Writing

We have been creating and saving data within our programs each time we run them. This is great for small amounts of data, but what if we need to save data for later use or share data with another program? Or we just want our data to persist between runs of our program?

We need to write our data to a file.

Let's start with passing data to our program from the command line.

```
public class InClass {

    public static void main(String[] args) {
        System.out.println("Hello, " + args[0]);
    }
}
```

We know now that `String[] args` is an array of strings and it's a parameter to the `main` method. We also know that our `main` method is the entry point to our program and runs when we execute our program.

This allows us to pass data to our program when we run it.

Let's pass Michael to our simple program.

```
$ java InClass Michael
Hello, Michael
```

When you include `Michael` after the name of the program, it is passed to the `main` method as an argument. We are passing the string `Michael` to the `main` method and it is stored in the `args` array. We can access each string we pass by its index.

If multiple `Strings` are passed to our program, we can access them by their index.

```
public class InClass {

    public static void main(String[] args) {
        System.out.println("Hello, " + args[0] + " " + args[1]);
    }
}
```

```
$ java InClass Michael Curry
Hello, Michael Curry
```

What happens if we want to pass numbers to our program?

```
public class InClass {

    public static void main(String[] args) {
        System.out.println(args[0] + " " + args[1]);
        int num1 = Integer.parseInt(args[0]);
        int num2 = Integer.parseInt(args[1]);
        System.out.println("The sum of " + num1 + " and " + num2 + " is " + (num1 + num2));
    }
}
```

```
$ java InClass 5 10
5 10
The sum of 5 and 10 is 15
```

We can use the wrapper class `Integer` and the method `parseInt` to convert a `String` to an `int`. We can do the same for `double` with `Double` and `parseDouble`.

We now have a way to pass external data into our program. What if we want to save data from our program to a file?

We can also do this through the command line by using the `>` operator. The `>` operator is the pipe operator and it redirects the output of a program to a file.

```
public class InClass {

    public static void main(String[] args) {
        System.out.println("Hello, " + args[0]);
    }
}
```

```
$ java InClass Michael > output.txt
```

You should now see a file named `output.txt` in your directory. If you open it, you should see `Hello, Michael`.

Java File Class

Java also has a built-in class called `File` that allows us to interact with files on our system. We can create a `File` object and use it to read and write data to a file.

```
import java.io.File;

public class InClass {

    public static void main(String[] args) {
        File file = new File("output.txt");
    }
}
```

We first need to import the `File` class from the `java.io` package. We then create a `File` object named `file` and pass the name of the file we want to interact with. This creates a reference to the file on our system.

Along with `.txt` files we an create `.csv`. Both of these formats are universal and can be opened in any text editor.

```
import java.io.File;

public class InClass {

    public static void main(String[] args) {
        File file = new File("output.csv");
    }
}
```

Let's write out program so we can write to a file. We need to add some code to your `main` method to write to the file. There could be an error while writing to the file, so we need to add a `throws IOException` clause to the `main` method. This will allow us to handle the error if it occurs. You will learn more about error exceptions in CS2.

```
import java.io.File;

public class InClass {

    public static void main(String[] args) throws IOException {
        File file = new File("output.csv");

        //Code to write to the file
    }
}
```

Now that we have a reference to a file, we can use the `File` class to interact and save data to the file. We can use the `PrintWriter` class to write data to a file.

```
import java.io.File;

public class InClass {

    public static void main(String[] args) throws IOException {
        File file = new File("output.txt");
        PrintWriter writer = new PrintWriter(file);
        writer.println("Hello, Michael");
        writer.close();
    }
}
```

Notice that we close the `PrintWriter` object after we are done writing to the file. This is important because it flushes the data to the file and closes the file.

Let's crate a program that prompts the user for their name and writes it to a file.

```
import java.io.File;

public class InClass {

    public static void main(String[] args) throws IOException {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter your name: ");
        String name = scanner.nextLine();
        File file = new File("output.txt");
        PrintWriter writer = new PrintWriter(file);
        writer.println("Hello, " + name);
        writer.close();
    }
}
```

We now can write files to our system. What if we want to read data from a file?

To read data from a file, we can use the `Scanner` class. We can create a `Scanner` object and pass the file we want to read from.

```
import java.io.File;

public class InClass {

    public static void main(String[] args throws IOException) {
        File file = new File("input.txt");
        Scanner scanner = new Scanner(file);
        String name = scanner.nextLine();
        System.out.println(name);
        scanner.close();
    }
}
```

Now that we have a Scanner object, we can read data from the file. We can use the `nextLine` method to read the next line of the file. We can also use the `hasNextLine` method to check if there is another line in the file.

```
import java.io.File;

public class InClass {

    public static void main(String[] args throws IOException) {
        File file = new File("input.txt");
        Scanner scanner = new Scanner(file);
        while(scanner.hasNextLine()) {
            String name = scanner.nextLine();
            System.out.println(name);
        }
        scanner.close();
    }
}
```

We can also save the data that we are reading so we can use it in our program. Now, we don't know the size of the data we are reading, so we can use an `ArrayList` to store the data.

```
import java.io.File;

public class InClass {

    public static void main(String[] args throws IOException) {
        File file = new File("input.txt");
        Scanner scanner = new Scanner(file);
        ArrayList<String> names = new ArrayList<>();
        while(scanner.hasNextLine()) {
            String name = scanner.nextLine();
            names.add(name);
        }
        scanner.close();

        printNames(names);
    }

    public static void printNames(ArrayList<String> names) {
        for(int i = 0; i < names.size(); i++) {
            System.out.println(names.get(i));
        }
    }
}
```

Other than `.txt` files you can also read and write to `.csv` (comma separated values) files. These files are used to store data in a table format. Each line in the file represents a row in the table and each value is separated by a comma.

We need to loop through each line in the file and split the line by the comma. We can then store the values in an array. When we learned about strings we learned about the `split` method. We can use this method to split a string by a delimiter.

Let's create a program that reads data from a `.csv` file and stores it in an `ArrayList`.

```
import java.io.File;

public class InClass {

    public static void main(String[] args throws IOException {
        File file = new File("input.csv");
        Scanner scanner = new Scanner(file);
        ArrayList<String[]> data = new ArrayList<>();
        while(scanner.hasNextLine()) {
            String line = scanner.nextLine();
            String[] values = line.split(",");
            data.add(values);
        }
        scanner.close();

        printData(data);
    }

    public static void printData(ArrayList<String[]> data) {
        for(int i = 0; i < data.size(); i++) {
            String[] values = data.get(i);
            for(int j = 0; j < values.length; j++) {
                System.out.print(values[j] + " ");
            }
            System.out.println();
        }
    }
}
```

Let's read a file that contains numbers and calculate the sum of the numbers.

```
import java.io.File;

public class InClass {

    public static void main(String[] args throws IOException {
        File file = new File("input.csv");
        Scanner scanner = new Scanner(file);
        int sum = 0;
        while(scanner.hasNextLine()) {
            String line = scanner.nextLine();
            String[] values = line.split(",");
            for(int i = 0; i < values.length; i++) {
                sum += Integer.parseInt(values[i]);
            }
        }
        scanner.close();

        System.out.println("The sum of the numbers is " + sum);
    }
}
```

We now can read and write data to files. We can also pass data to our program from the command line. We can use these techniques to save data for later use or share data with another program.

Our data is now persistent!