Lab 1 - Input and Output

due August 23, 2020 at 23:55 EDT

Objective

This week's lab will help you get familiar with

- Using BlueJ
- `public static void main`
- Reading input from command line
- Reading from files using the `Scanner` class
- Writing to files
- Using the command line

Using BlueJ

This section is for students who used Processing in their intro course. For those who have already used BlueJ, feel free to skip this section.

Download and install BlueJ from here:
https://www.bluej.org/

Since some of you have not seen BlueJ before, here are a few tips on how to use it, and its relation to processing:

1. Whereas Processing had sketches which could have multiple Java files, BlueJ uses projects. To create a new project simply go to Project → New Project. Choose the location where you would want to save all your projects and give your project a name. This will create an empty folder with the project's name where all your Java files will be created (similar to processing, although there is no initial .pde file).
2. To create a new class, right click anywhere in the project and select new class. Give the class a name and an example class with that name will be created for you. (similar to opening a new tab in

Processing). Notice that a dummy instance variable, constructor and method are created. You will need to erase those when starting to work on your own program.

3. Finally to run the program you will need to run the main method (kind of like what setup is in Processing). In order to do this right click the class that has the main method and select the main method. Read the next section for explanation.

`public static void main(String[] args)`

When running a program written in Java, the first line to be executed is the first line in the **`main method`**. Usually only one class has a **`main method`**, which is called by the operating system when the program is run (for example, when an icon is double clicked). If you were using Processing in CS104/105 you did not have to explicitly write a main method, but Processing would create a main method for you behind the scenes.

For CS150 you will always need to use the **`main method`**, so there are a few things to understand about it. First, this is a static method that runs in a static context. A static method is a method that is not called from an object but is called directly from the class using its name. It makes sense that the **`main method`** needs to be static since no objects have been created yet. This should be the **ONLY** static method in your program. (note that this is not what is necessarily done in the book, but we will insist on doing it this way here)

Since static methods are not called from an object, this means you do not have access to non-static variables and methods (which are in an object). The non-static context will only come into existence once you have instantiated the object. The most common approach is to minimize the code in main to just instantiating the class and calling a method. So often you will see something like the following code, where the class will instantiate itself. Pay attention to the comment in the main method and convince yourself why it is true.

```
public class StaticTestClass {

          Integer x;

          Integer y;

          public static void main(String[] args) {
            // The following if executed would be a compile error:
            // System.out.println(y);

            StaticTestClass st1 = new StaticTestClass();

            st1.run();

          }

          //constructor
          public StaticTestClass() {

            this.x = 3;
            this.y = 4;
          }

          public void run() {

            System.out.println("st1: " + this.x + "   " + this.y);

            x = 100;
            y = 300;

            System.out.println("st1: " + this.x + "   " + this.y);

          }

}
```

To run your program, you can right click the class which has the `public static void main` method and select `void main`. Then simply click OK to run the program.

When run in BlueJ, the above code should produce the following output.

```
st1: 3  4
st1: 100  300
```

Start a new project in BlueJ and make sure you can run it.

String[] args

When you start a program you can send in different parameters. This can be helpful for example if your program needs to read a file, and you would like the file to be sent as a parameter. These are sent as the **String [] args**. To do this in BlueJ, simply put in the strings in the curly braces after right clicking the class and selecting void main. For example the following program will simply print out the contents of the args array.

```java
public class ArgsTest
{
    public static void main(String[] args) {
        // The following if executed would be a compile error.
        // System.out.println(y);

        ArgsTest st1 = new ArgsTest();

        st1.run(args);

    }

    public void run(String[] a)
    {
        for(int i=0;i<a.length;i++)
        {
            System.out.println(a[i]);
        }
    }
}
```

Run this program and make sure you understand how it works.

Part 1: Reading from command line

Scanner Class

Let's start with the Scanner class. First off you will use the Scanner class to break your input stream into pieces, e.g., lines into words. To quote the Java API, the Scanner class is:

- A simple text scanner which can parse primitive types and strings using regular expressions.
- A Scanner breaks its input into tokens using a delimiter pattern, which by default matches whitespace. The resulting tokens may then be converted into values of different types using the various next methods.

An example program that reads and prints two lines from the console can be found below. It shows you how to create a program that reads and prints to/from the terminal using the Scanner class.

```
import java.util.Scanner;

public class TerminalDemo {
   public static void main(String[] args) {

      TerminalDemo demo = new TerminalDemo();
      demo.run();

   }

   public void run() {
      Scanner reader = new Scanner(System.in);
      String name = null;
      int age;
      try {
         // read line from the user input
         System.out.print("Name: ");
         name = reader.next();
         // prints
         System.out.println("Name entered : " + name);

         System.out.print("Age: ");
         age = reader.nextInt();
         System.out.println("Age is: " + age);
      } catch (Exception e) {
         System.out.println( "Exception occured " + e);
      }

   }
}
```

When dealing with input and output, you will generally need to learn how to handle exceptions. A tutorial on exception handling can be found here while the motivation for the use/need of the mechanism can be found here.

Assignment:

Your assignment is to write a simple program which evaluates inequalities of heights (2 numbers - feet and inches) that works the following way:

1. First the program will ask how many inequalities you wish to evaluate.
2. Then for each inequality your program should do the following:
    a. Ask for the first number for the inequality (int). This represents the feet.
    b. Ask for the second number for the inequality (float). This represents the inches
    c. Ask for the inequality operation (string):
        i. "gt" - means greater than
        ii. "lt" - means less than
        iii. "gte" - means greater than or equals

    iv. “lte” means less than or equals
    v. If any other string is given, an error message should be printed.

d. Ask for the third number for the inequality (int).
e. Ask for the fourth number for the inequality (float).
f. Print the result of the inequality (should be true or false)

1. Example input/output data from the user should look like:

```
number of calculations:3
First number: 5
Second number: 3.5
Operation: gt
Third number: 6
Fourth number: 0.0
false
First number: 6
Second number: 1.0
Operation: gte
Third number: 6
Fourth number: 0.5
true
First number: 5
Second number: 8.1
Operation: equals
Third number: 5
Fourth number: 9.0
undefined operator
```

Part 2: Reading from files

The scanner class can be used to read from files as well.

```
Scanner sc=null;
try{
  sc = new Scanner(new FileReader("myNumbers.txt"));
  while (sc.hasNextLong()) {
     long aLong = sc.nextLong();
     System.out.println(aLong);
   }
}
catch(Exception e)
   {System.out.println(e);}
```

You can check what is the next token in the input stream by using the "has" methods, e.g., the program above is using the `hasNextLong()` method to determine if the next token is a Long. Then using the "next" methods you can retrieve the next token (which will be removed from the input stream) and another token will become the "next token". Be very careful about the difference between the "has" and the "next" methods. The "has" methods return a boolean but do not actually read anything from the input stream. The "next" methods will return a token of the appropriate type and will read past where that token occurs in the input stream. For example:

```
sc.hasNextLong();
sc.hasNextLong();
aLong = sc.nextLong();
```

will only read ONE token from the input stream - this is done in the 3rd line. But:

```
sc.nextLong();
sc.nextLong();
aLong = sc.nextLong();
```

will read THREE tokens of the type Long and the last one will be stored in the variable `aLong`.

Writing to files

There are different ways to write to text files in java. One common way is to use the PrintWriter class. If using that class make sure to close the PrintWriter when done.

File Location

Typically if you have input files that are needed to run or test your program, you should provide those with your code at submission time. When you execute your java code, where the code is located is known as the **working directory** and everything is accessed relative to this. Thus if you created an instance of the File class, like **`new File("myNumbers")`**, then myNumbers would be in the working directory. But if you were to store your files in a sub-directory (or folder) named **`fldr`**, then the invocation might look something like this: **`new File("fldr/myNumbers")`**. One annoying feature about java, is how a directory is referenced depends upon the Operating System you are using.

Assignment:

For this part of the assignment create a program that will read a file from the current working directory and write information to a file in the working directory.

The input file contains lines with text

```
The fox in the house is sad
I know you are loving this assignment
Maybe I'm wrong
```

The output file presents the number of words and the number of letters in each line not including whitespaces). In addition, the last word should be printed out as well. For example, the output for the previous input would be:

```
Words: 7 Letters: 21 Last: sad
Words: 7 Letters: 31 Last: assignment
Words: 3 Letters: 13 Last: wrong
```

Your input data file (i.e., you should create one (or more) of your own to test your program) should be submitted to the moodle in addition to the code.
You can assume the input file is correctly formatted. The output file should be created in your current working directory.

Notice: Scanner can be created with FileReader or String. So in this assignment you might want to use two scanners: one to read in from the file line by line, and another to read each line as separate strings.

Submission:

For this lab you need to submit to programs (part1 + part2). Please submit two Zip files to the Moodle Page

Part 3: Using the command line

BlueJ is an IDE (a development environment). It is also possible, and sometimes preferable, to create and run your java programs using the Linux/Ubuntu/Windows command line interface (CLI). For this lab we will demonstrate how to run java programs which were created in BlueJ, using the CLI.

If you have never used the command line before you might use the following websites for help:
Windows: https://introcs.cs.princeton.edu/java/15inout/windows-cmd.html
Mac: https://introcs.cs.princeton.edu/java/15inout/mac-cmd.html

You will need to navigate to the directory in which your source files are stored. The following commands will help you to do that:

ls/**dir** --- to list all the directories and files in the current working directory

cd <directory name> to navigate to a subdirectory of the current working directory

cd .. to navigate up one level from the current working directory

To compile a program type **javac <filename>.java.** The result will be a .class file. To run your java program, use the command **java <filename>**

It is important to check the version of java to assure that it is the same version used in BlueJ. If it is not, you will always need to compile your program in the CLI (as described above) before running it.

Assignment:

1. Using the CLI, compile your program from part 1.
2. Run the program in the CLI. Have at least 3 inequalities with different inputs. The final one should demonstrate what happens with a user error.
3. Take a screenshot of the CLI showing the two tasks above. The screenshot will be submitted as part of this lab. You should put it in the report for part 1.

Grading:

commenting+style(0.5 pt)
report (0.5 pt) - (notice that this report does not require unit testing)

Part 1:

1. looping and reading input correctly (2 pts)
2. calculating and printing correctly (2 pt)

Part 2:

1. Reading from a local file. (1 pt)
2. Writing correctly to the output file. (1 pt)
3. Counting letters and words (1pt)
4. Finding the last word in a line (1 pt)

Part 3 (in report of part 1):

1. Compiling and running program from command line interface (1pt)