Homework #9 Solution
(Java Programming for Beginners - OnLine)
(Directly helps with your final)

There are three distinct areas you need to research and focus on to get your final done properly:

- Process command line arguments to get the names of input and output files from the user using Elipse
- Read and write information from and into the disk files
- Parse the data read from input file, store in a variable, process it and then print it on the screen and into the output file.

Here are couple of exercises, which will help you in writing code for the above problems.

9.1 *Write a class named DiskIO. Your main method's header looks like this:*

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

where, args is the String array of command line arguments passed by JVM. Write a method processCLArguments() like this:

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

if the command args contains less than two strings, it displays the message like this:

```
Usage: java DiskIO inputFile outputFile
```

If the arguments contains input and output file names it displays the message like this:

```
 Input will be read from: input_final.txt
Output will be written into: output_final.txt
```

Solution:

```
import java.util.Scanner;

public class DiskIO {
     public static void processCLArguments(String[] args) {
          // proceed because you found two arguments
          // check if user gave the command line arguments
          if (args.length < 2) {
             System.out.println("Usage: java Exercise5
                                 inputFile outputFile");
             System.exit(1);
          } else {
             System.out.println("Input will be read from:"+
                                 args[0] +
                        "\nOutput will be written into: " +
                                 args[1]);
             System.out.println();
          }
     }

     public static void main(String[] args) {
          // call method to process command line arguments
          processCLArguments(args);
     }
}
```

9.2 *Write another static method called processInputOutputFiles(), which takes a String array as arguments which has names of input and output files.*

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

It reads the content of the input file whose name is in the first element of args String array with the scores for unknown number of students like the following. The format of the data is, name of a student, followed by comma-separated scores of quiz1, quiz2, quiz3, quiz4, midterm 1, midterm 2 and final for that student:

```
Thui Bhu, 100, 90, 80, 100, 89, 99, 88
Ariana B. Smith, 90, 90, 100, 100, 99, 100, 95
//more students data
```

Read each line (scores of a student) at a time, and

a) display the output on the screen like this:

```
Student #: 1 Thui Bhu, 100, 90, 80, 100, 89, 99, 88
Student #: 2 Ariana B. Smith, 90, 90, 100, 100, 99, 100, 95
```

b) and print the output on the disk file whose name is given in the second element of args String array. The disk output looks like this:

```
Student #1 is: "Thui Bhu" whose raw scores are:  100:  90:  80:  100:  89:  99:  88:
Student #2 is: "Ariana B. Smith" whose raw scores are:  90:  90:  100:  100:  99:  100: 95:
```

Solution:

```java
import java.io.File;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Scanner;

public class DiskIO {
     public static void processInputOutputFiles(
                                             String[] args) {
          // call method to process command line arguments
          File inputFile = new File(args[0]);
          if (!inputFile.exists()) {
               System.out.println("Input file " +
                                   args[0] +
                                   " does not exist");
               System.exit(2);
          }

          // Output file
          File outputFile = new File(args[1]);

          try {
               // Create input and output files
               Scanner input = new Scanner(inputFile);
               PrintWriter output = new
                              PrintWriter(outputFile);
               int i = 1;
               while (input.hasNextLine()) {
                    String line = input.nextLine();
                    // display the whole line on screen
                    System.out.println("Student #: " + i +
                                        " " + line);

                    // this line has one students data,
                    //use split
                    String[] studentInfo = line.split(",");
                    // write the data in output file
                    output.print("Student #" + i++ +
                              " is: \"" +
                              studentInfo[0] +
                              "\" whose raw scores are: ");
                    for (int j = 1; j < studentInfo.length;
                                           j++)
                         output.print(studentInfo[j] +
                                      ": ");
                    output.println(); // one student done
                                      //give a line feed
               }
               input.close(); // done with them, close it
               output.close(); // done with them, close it
          } catch (IOException e) {
               System.out.println(
                       "Error reading from input file: " +
                       args[0]);
          }
     }

     public static void main(String[] args) {
          // Process input and output file
          processInputOutputFiles(args);
     }
}
```

9.3 Research on how to give the command line arguments in your class file using Eclipse.

When starting a Java application, you may want to supply few input parameters to your application even before your program starts. For example, giving database names, supplying username and password etc.

Supplying these arguments before the application starts differs in the way you start your application. You can run your Java application through command window (Windows) or terminal window (MAC). You can even supply these arguments while running from your favorite IDE, e.g. Eclipse.

So, there are two questions:

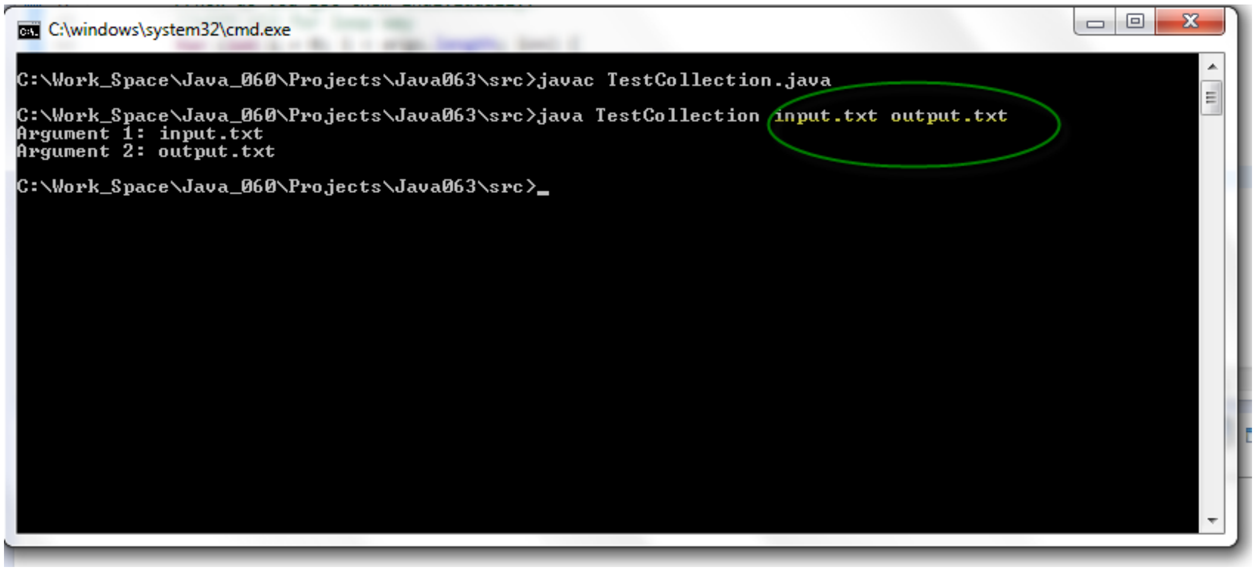
a) How do you supply these arguments to your program, and

b) How do you process these arguments in your program

Solution:

1) **Supply arguments when running your Java program from command window**:

- Write your Java command as usual (meaning Java YourClassName), and then write your arguments one after another separated by a space. Each word after the class name becomes one argument to your program. JVM, picks up these words (arguments) you write into a String array and supplies to your program through **args** parameter in your main method of the class (YourClassName)
- If you move your .class file in some other folders, then you have to make sure your CLASSPATH environment variable is set properly. However, if you are using Eclipse for the development, you don't have to worry about CLASSPATH, Eclipse takes care of it for most cases.
- If you only use Eclipse to run your code, there is no need to test through command or terminal window like this. Go to Step #2 and use Eclipse dirctly



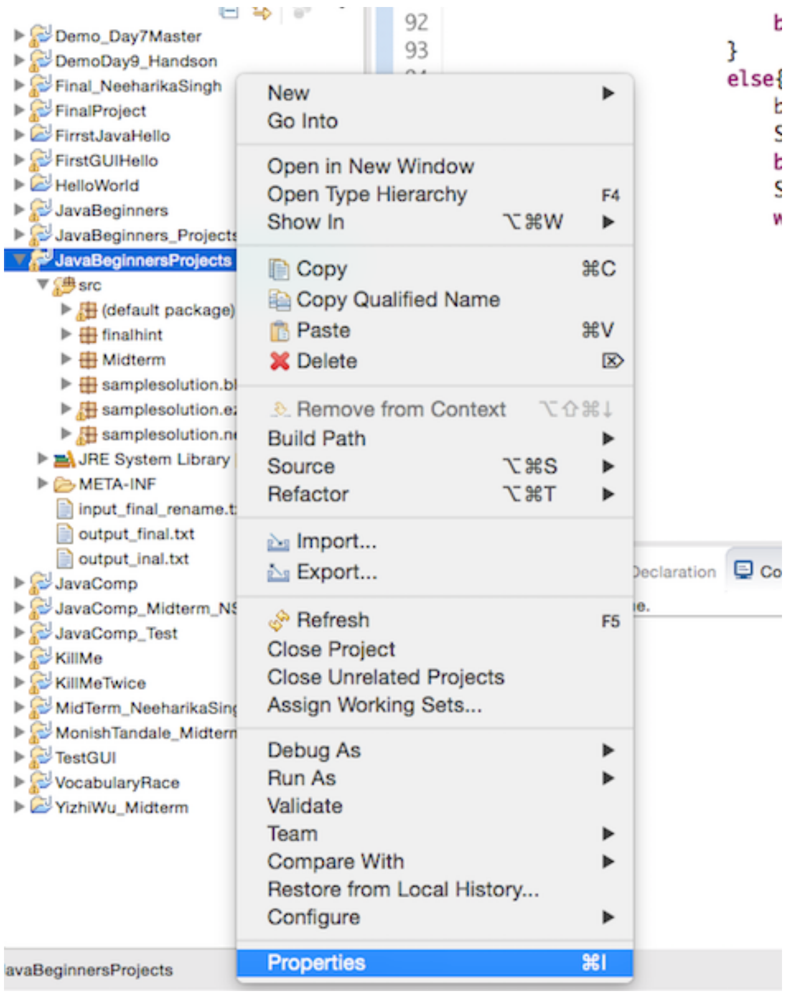
```
C:\Work_Space\Java_060\Projects\Java063\src>javac TestCollection.java

C:\Work_Space\Java_060\Projects\Java063\src>java TestCollection input.txt output.txt
Argument 1: input.txt
Argument 2: output.txt

C:\Work_Space\Java_060\Projects\Java063\src>_
```

2) **Supply arguments when you are running application in Eclipse**

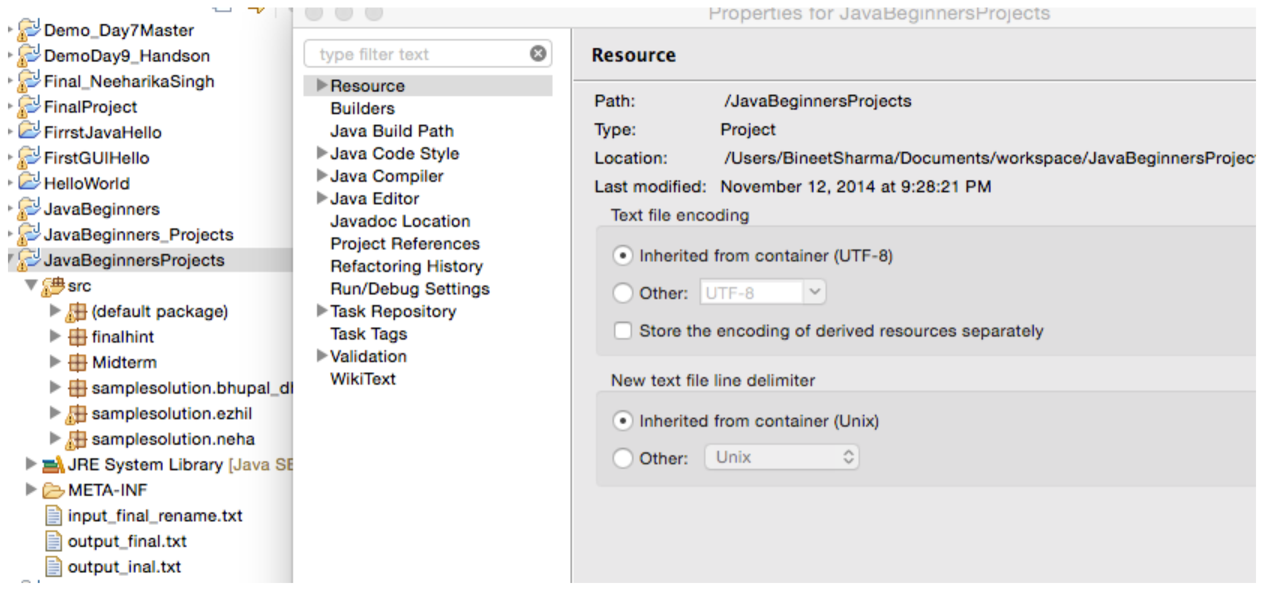
By default, Eclipse looks for the files at the project's directly level (not src or bin folders which one level below project's folder. Also, note that you are allow to give full path or relative path of the files if you like).

- If you want to test your application in **Eclipse**, then you have to set these arguments as an environment variable in the Eclipse itself as you can't write the arguments before your program runs like you would do in a command window.

- First, you should know where your java and class files are located in the computer. In case of Eclipse, you can determine that by checking properties of files and projects.
- First right clicking on the your Java project in left hand pane, and then clicking on the properties.



- Observe the path, which is the relative path of your project from the project directory
  Observe the full path of the project in Resource.



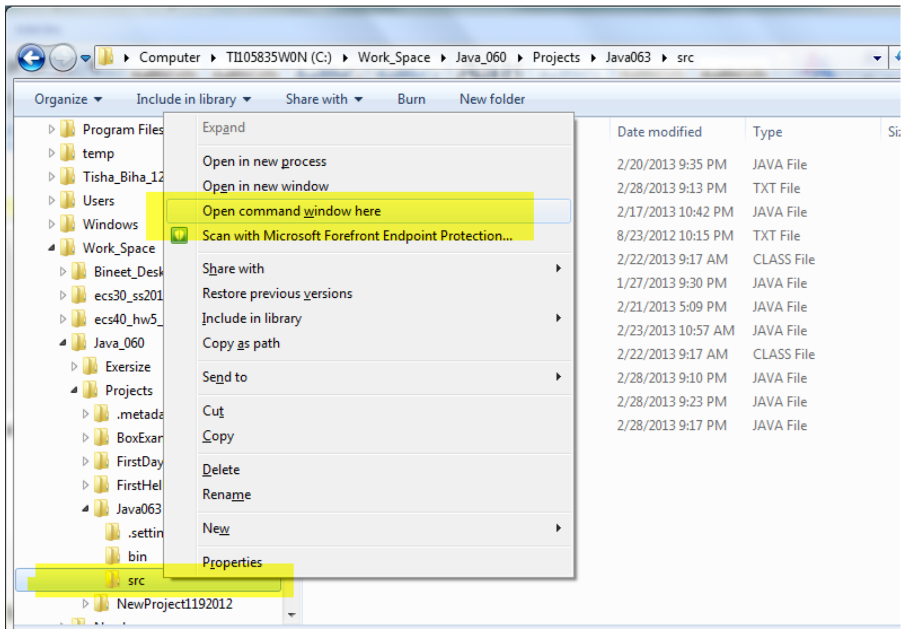
Resource

Path:	/JavaBeginnersProjects
Type:	Project
Location:	/Users/BineetSharma/Documents/workspace/JavaBeginnersProjec
Last modified:	November 12, 2014 at 9:28:21 PM

Text file encoding
- [✓ Inherited from container (UTF-8)]
- Other: UTF-8
- Store the encoding of derived resources separately

New text file line delimiter
- [✓ Inherited from container (Unix)]
- Other: Unix

- Now, open the command window in that location (BTW: you could also correctly set the path and CLASSPATH environment variables and you should be able to run your code from anywhere – however, you don’t need to do that when using Eclipse)
  - You could do this in one of two ways, either directly navigate (change directory using “cd” command) to that directory in a command window
  - Or, for Windows only, open the windows explorer and then shift+right click to open the context menu and then choose **open command window here** (shift+right click works only for Windows).
  -

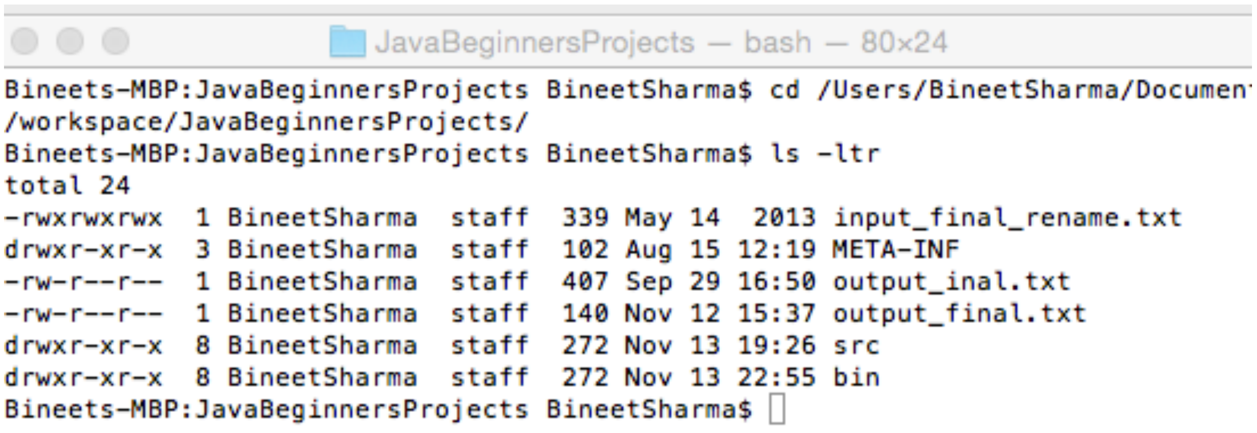


Date modified	Type
2/20/2013 9:35 PM	JAVA File
2/28/2013 9:13 PM	TXT File
2/17/2013 10:42 PM	JAVA File
8/23/2012 10:15 PM	TXT File
2/22/2013 9:17 AM	CLASS File
1/27/2013 9:30 PM	JAVA File
2/21/2013 5:09 PM	JAVA File
2/23/2013 10:57 AM	JAVA File
2/22/2013 9:17 AM	CLASS File
2/28/2013 9:10 PM	JAVA File
2/28/2013 9:23 PM	JAVA File
2/28/2013 9:17 PM	JAVA File

-
-

This opens the command window and even takes you directly to that folder

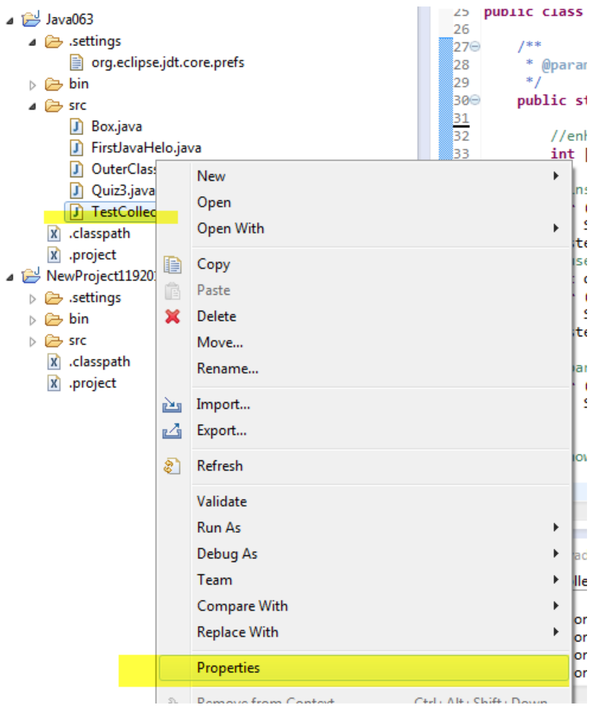
If you are using mac (like me), simply open a terminal and cd to that folder like below



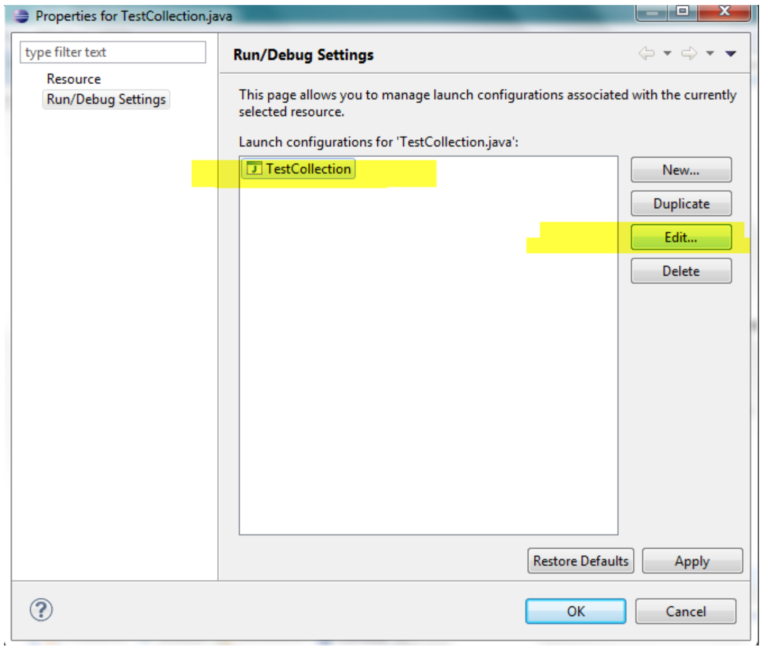
```
Bineets-MBP:JavaBeginnersProjects BineetSharma$ cd /Users/BineetSharma/Documen
/workspace/JavaBeginnersProjects/
Bineets-MBP:JavaBeginnersProjects BineetSharma$ ls -ltr
total 24
-rwxrwxrwx  1 BineetSharma  staff  339 May 14  2013 input_final_rename.txt
drwxr-xr-x  3 BineetSharma  staff  102 Aug 15 12:19 META-INF
-rw-r--r--  1 BineetSharma  staff  407 Sep 29 16:50 output_inal.txt
-rw-r--r--  1 BineetSharma  staff  140 Nov 12 15:37 output_final.txt
drwxr-xr-x  8 BineetSharma  staff  272 Nov 13 19:26 src
drwxr-xr-x  8 BineetSharma  staff  272 Nov 13 22:55 bin
Bineets-MBP:JavaBeginnersProjects BineetSharma$
```

Either way check your directories. You have to make sure that your input file is in the level of directories along with src and bin folders. So, by default the Eclipse looks at files in the project level folder.

- Back to Eclipse. Write click in your java file for which you want to set a command line argument and then chose properties



- Click on the file name and then press edit button in the next window

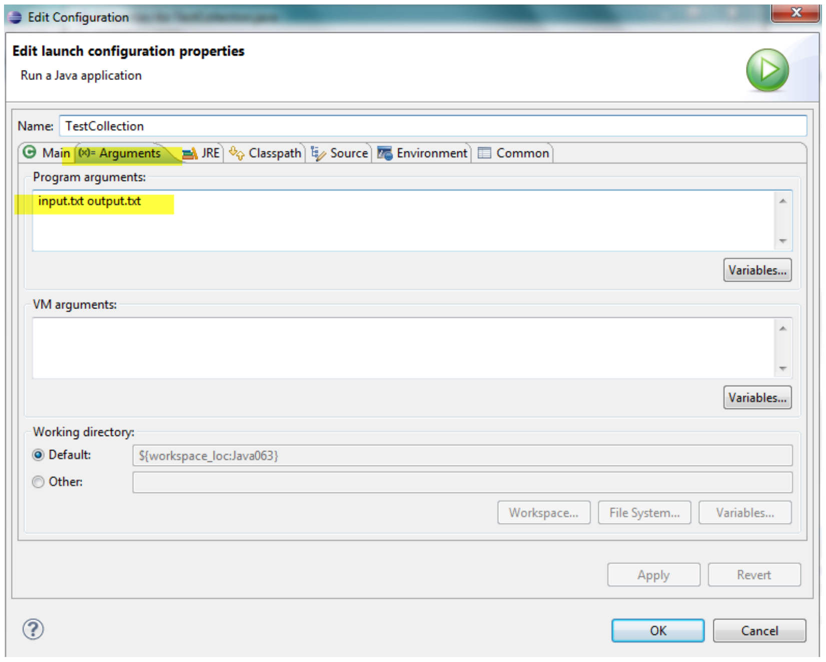


This page allows you to manage launch configurations associated with the currently selected resource.

Launch configurations for 'TestCollection.java':

TestCollection

- Click on the arguments tab and then write all the arguments you want to be passed to this program separated by space between each of them.



Name: TestCollection

Program arguments:

input.txt output.txt

VM arguments:

Working directory:

(●) Default: ${workspace_loc:Java063}

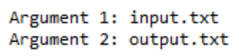
( ) Other:

Press Ok, to save your settings.

- You can write the code in your source file like this to extract these two command line arguments given by the user at run time, one by one ("input.txt" and "output.txt") like this:

  ```
  public class TestCollection {
          public static void main(String[] args) { //args has those two words
                  for (int i = 0; i < args.length; i ++) {
                          System.out.printf("Argument %d: %s\n", i+1, args[i]);
                  }
          }
  }
  ```

  This will yield the output like below. Of course, this code example only shows how to extract the command lines in your real application you will use these arguments for different purposes, e.g., input file names, output file names, an input for a method, or, some configuration values, database names, user name and password etc.

  ```
  Argument 1: input.txt
  Argument 2: output.txt
  ```