Chulalongkorn University
International School of Engineering
Department of Computer Engineering
2140105 Computer Programming Lab.

Name _____
Student ID _____
Station No. _____
Date _____

Lab 2 - Using Java API documents, command-line arguments, and file I/O.

Objectives:

- Learn to use Java API documents.
- Understand command-line arguments and how to write Java programs to get them.
- Understand Java file input/output, how to read from a file and write to a file.

1. Using Java API documents

The Java API (Application Programming Interface) is like a super dictionary of the Java language. It has a list of all Java packages, classes, and interfaces; along with all of their methods, fields and constructors; and also with information on how to use them. You can look at Java API documents at http://java.sun.com/javase/6/docs/api/

Each API is organized into three frames: two small ones on the left listing the available packages and the interfaces and classes in that package, and a large one on the right showing details on the current selection as shown in Figure 1. There is also an alphabetical index listing all classes, interfaces, fields, methods, and constructors.

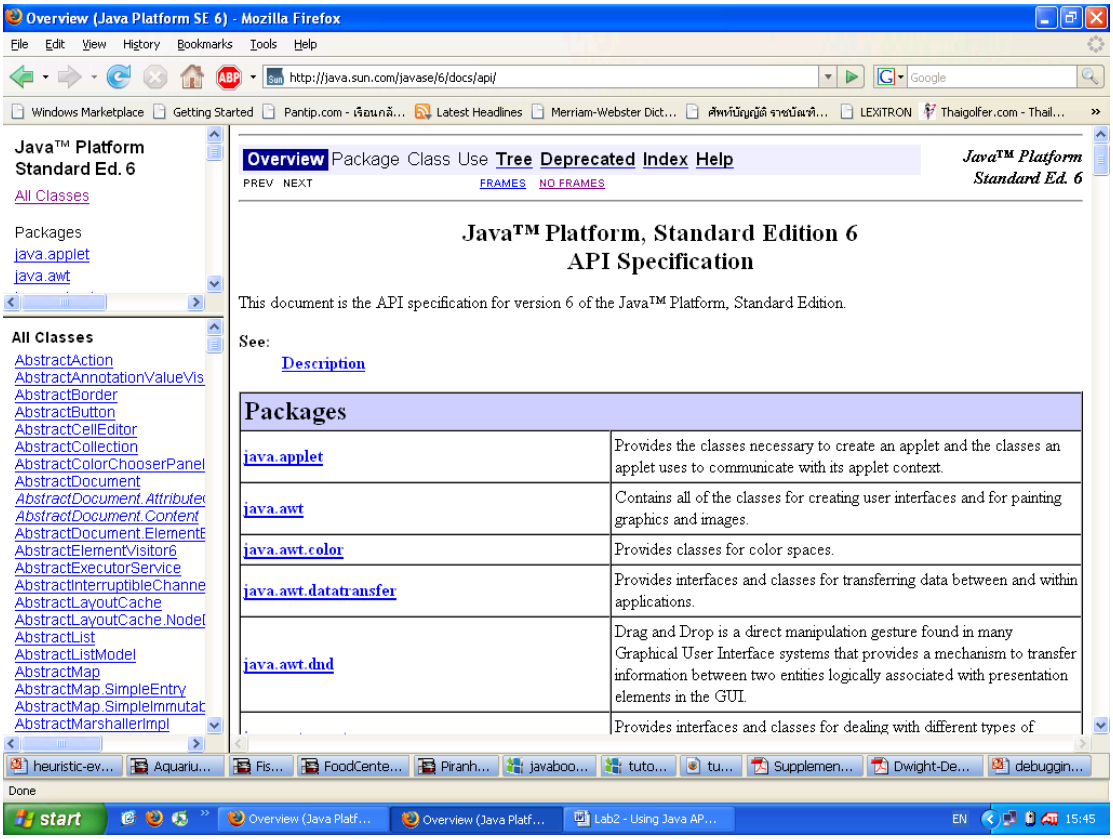

Figure 1 - Java API Document

There is Java API document in HTMLHelp format which can be downloaded from http://www.allimant.org/javadoc.

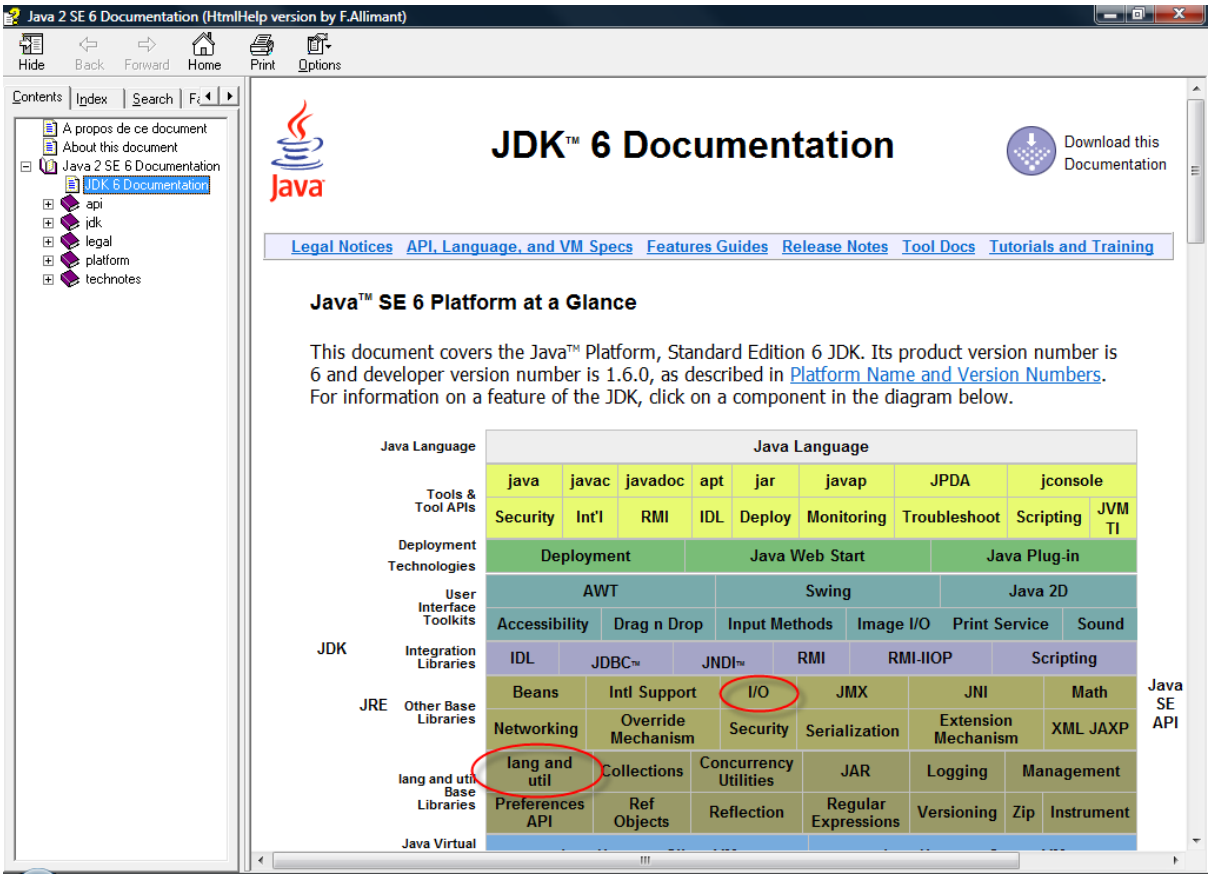

Figure 2 - Java API in HTMLHelp format

Figure 2 shows Java API document in HTMLHelp format. There are a lots of packages and classes. But we will need to look at the Java API about input/output (I/O), lang, util; which are **`java.lang`**, **`java.io`**, and **`java.util`** package.

> ***Your turn*** ①
>
> Find the API document for creating a new empty file in the specified directory.
> Write a Java statement to create **`tempfile.txt`** in **`C:\Temp`**

2. Command-Line Arguments

A Java application can accept any number of arguments from the command line. This allows the user to specify configuration information when the application is launched.

The user enters command-line arguments when invoking the application and specifies them after the name of the class to be run. For example, suppose a Java application called **`Sort`** sorts lines in a file. To sort the data in a file named **`friends.txt`**, a user would enter:

```
java Sort friends.txt
```

2.1 Parsing String Command-Line Arguments

Arguments are passed as a String array to the main method of a class. The first element (element 0) is the first argument passed not the name of the class.

> ***Your turn*** ②
>
> Create a new class called **`CommandLineArgument.java`** and copy the following Java statements as shown in Figure 3.
> Follow step 1) to 4) to run **`CommandLineArgument`** with the arguments given in Eclipse.

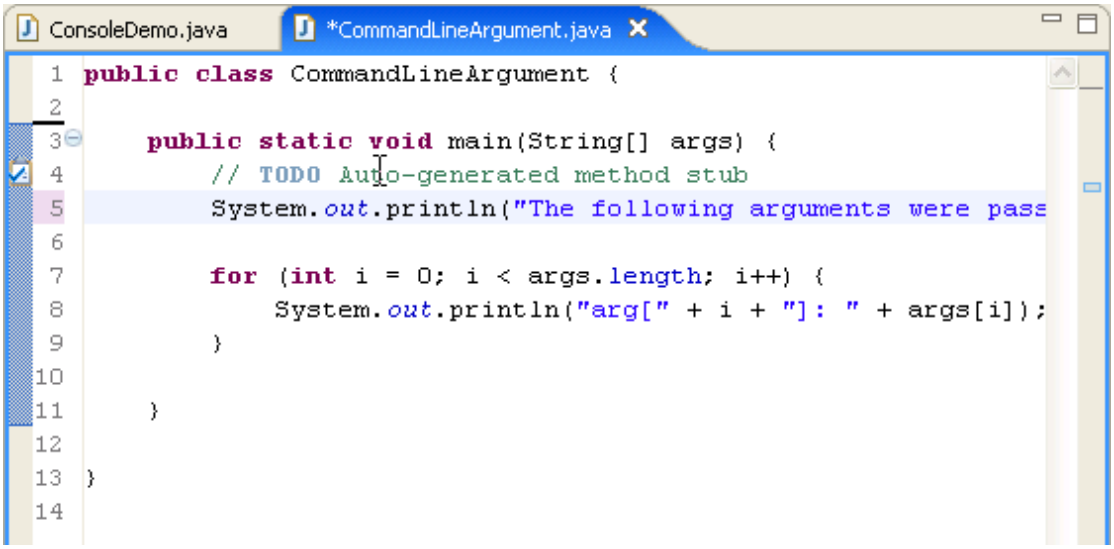

Figure 3 - CommandLineArgument.java

With the following command line, the output shown is produced as shown in Figure 4.

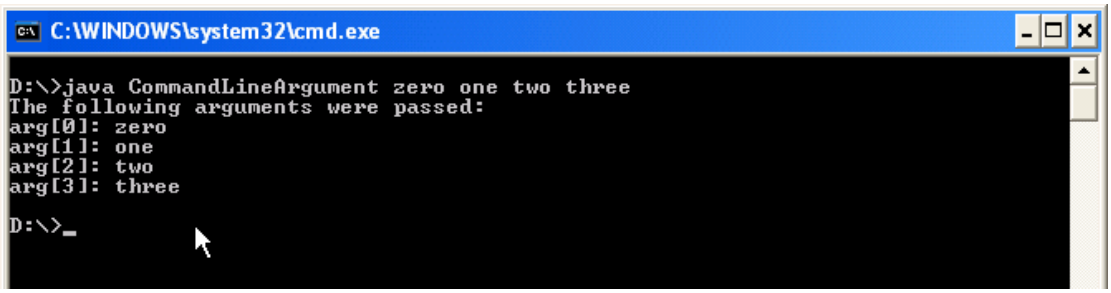

Figure 4 - Running a program in console window.

You can enter command line argument when running your Java program in Eclipse as following:

1) Select menu **Run > Run…** as in Figure 5. A new window will be displayed, called Run configuration window as shown in Figure 6.

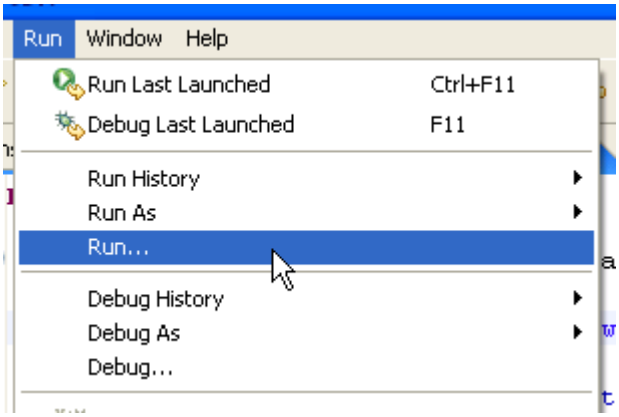

Figure 5 - Run menu

2) Click on **Arguments tab**

3) Enter your arguments in Program arguments: box

4) Click **Run**

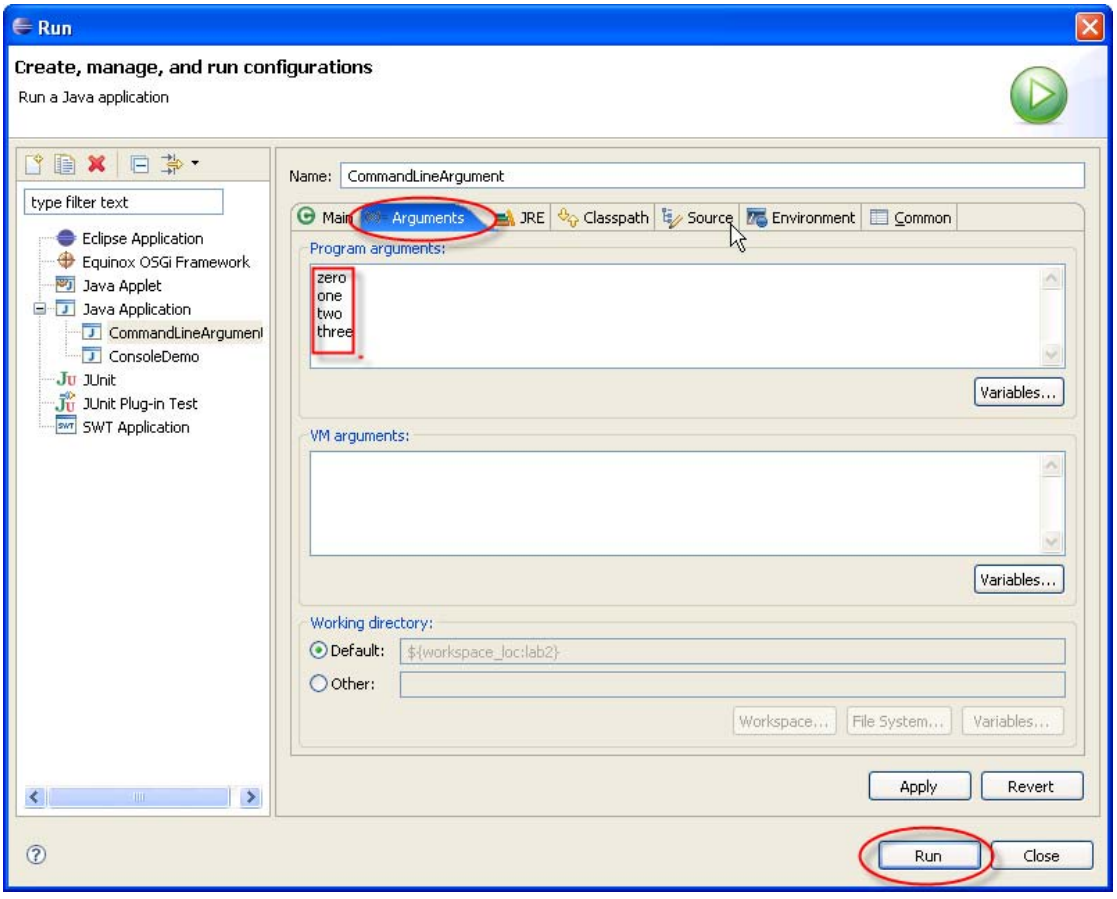

Figure 6 - Run configuration window

Figure 7 shows the output of the program **`CommandLineArgument`** in console view.

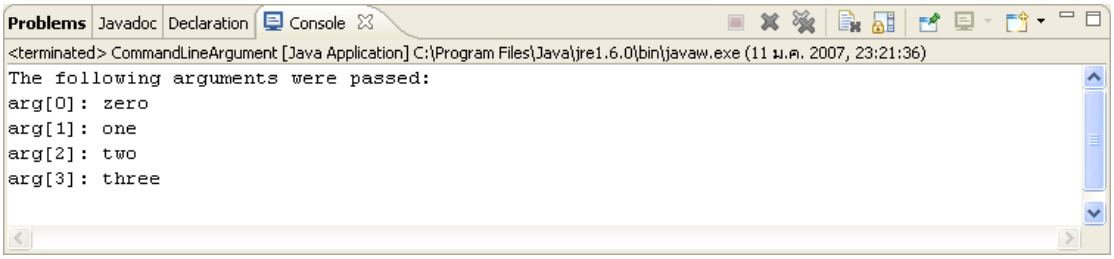

Figure 7 - Eclipse's Console View

2.2 Parsing Numeric Command-Line Arguments

If an application needs to support a numeric command-line argument, it must convert a String argument that represents a number, such as "**34**", to a numeric value. Here is a code snippet that converts a command-line argument to an **int**:

```
int firstArg;
if (args.length > 0) {
    try {
        firstArg = Integer.parseInt(args[0]);
    } catch (NumberFormatException e) {
        System.err.println("Argument must be an integer");
        System.exit(1);
    }
}
```

parseInt throws a **NumberFormatException** if the format of **args[0]** is not valid. All of the **Number** classes — **Integer**, **Float**, **Double**, and so on — have **parseXXX** methods that convert a **String** representing a number to an object of their type.

3. File I/O and Streams

You can write data to a file instead of the computer screen. You can write certain data to a file while still putting other data on the screen. Or you may need access to multiple files simultaneously. Or you may want to query the user for input rather than accepting it all on the command line. Or maybe you want to read data out of a file that's in a particular format. In Java all these methods take place as streams. The **System.out.println()** statement we've been using all along is an implementation of Streams.

3.1 A program that writes a string to a file

In order to use the Java file classes, we must import the Java input/output package (**java.io**) in the following manner

```
import java.io.*;
```

Add **throws IOException** after **main** method header.

Inside the main method of our program,

1. We must declare and create a **`FileOutputStream`** object that takes as its constructor the a **`String`** indicating the file we wish to be the output.

2. Create a new **`PrintStream`** object that takes as it constructor the existing **`FileOutputStream`**.

3. Now, any data we send from **`PrintStream`** will be passed to the **`FileOutputStream`**, and to disk. We the make a call or calls to the **`print`** or **`println`** method of the **`PrintStream`**, passing it a string.

4. When you are done, close the connection

Your turn ③

Create a new class called **`FileOutput.java`** and copy the following Java statements. Run and see the output in Eclipse.

The following listing is an example of writing strings to a file.

```
/*
* FileOutput
* Demonstration of FileOutputStream and PrintStream classes
*/
import java.io.*;
public class FileOutput {

  public static void main(String args[]) throws IOException {
    // 1) declare and create a file output object
    //    connected to "myfile.txt"
    FileOutputStream out = new FileOutputStream("myfile.txt");

    // 2) create a new print stream object and connect
    //    to the output stream
    PrintStream p = new PrintStream(out);

    // 3) write to PrintStream and file
    p.println("This is written to a file myFile.txt");
    p.println("It is very easy, isn't it?");

    // 4) close the connection
    p.close();
  }
}
```

To see what happen in Eclipse:

Right-click at your project in package explorer view and select "**Refresh**" or press **F5**

You will see that there will be a file named **myFile.txt** added to your project. **Double-click** on the file to see the content.

Try to change the **print** or **println** in step 3 and see the output again.

3.2 Reading a text file

Now that we know how to write a text file, let's try reading one.

1) **import java.io.*;**

2) **import java.util.Scanner;**

3) Add **throws IOException** after **main** header

4) Declare and create a **File** object that takes as its constructor's argument a **String** indicating the file we are about to read.

5) Declare and create a **Scanner** object that takes as its constructor's argument the **File** object we create in step 4).

6) Use method **hasNext** of the **Scanner** object created in step 5) to check whether there is something to read or not. If there is something to read **hasNext** returns **true**, otherwise it returns **false**.

7) You can read one line at a time by calling method **nextLine** of the **Scanner** object. This method will return a string for the current line (exclude the newline (**'\n'**) character).

Your turn ④

Create a new class called **PrintFiles.java** and copy the following Java statements. Run and see the output in Eclipse.

The following code accepts a series of file names on the command line and then prints those filenames to the standard output in the order they were listed.

```
// read a file and print out

import java.io.*;
import java.util.Scanner;

public class PrintFiles {

  public static void main (String args[]) throws IOException {

    //Loop across the arguments
    for (int i=0; i < args.length; i++) {

      //Open the file for reading
      File fin = new File(args[i]);
```

```
      Scanner sc = new Scanner(fin);

      while (sc.hasNext()) {
        String thisLine = sc.nextLine();
        System.out.println(thisLine);
      } // while loop end
  } // main ends here

}
```

3.3 Getting keyboard input

Java has been very difficult to get keyboard input. In the past we use BufferedReader to get keyboard input like this

```
BufferedReader kb = new BufferedReader(
                        new InputStreamReader(System.in));
String input = kb.readLine();
```

The input you entered will always be a String. You have to use **`XXXX.parseXXXX`** to change to numeral types yourself.
Since Java 5.0, there is a new class called **`Scanner`** in package **`java.util`** that will make keyboard input a lot easier.

1) **`import java.util.Scanner;`**
2) Create a **`Scanner`** object that takes as its constructor's argument, **`System.in`**.
3) Use the following methods
   - **`nextLine()`** to get a String
   - **`nextInt()`** to get an integer
   - **`nextDouble()`** to get a double

Your turn ⑤

Create a new class called **`KeyboardInput.java`** that takes integer, double, and String inputs and then prints them out in reverse order.

Lab 2 Exercise

Your turn

1) Write a program called **`Sort.java`** that takes three command line arguments and then print them out in alphabetical order (number comes before String). **Hint:** look at Java API document about class **`String`**. There are methods that will help you.

   Make sure that you test you program with the following test cases:

Arguments	***Results***
`one two three`	`one three two`
`one 2 3`	`2 3 one`
`computing compute computer`	`compute computer computing`
`Sunday Monday Tuesday`	`Monday Sunday Tuesday`

2) Write a program called **`TempConversion.java`** to convert temperature either from degree Celsius to Fahrenheit or from Fahrenheit to Celsius. It takes three arguments: the first argument is the temperature to be converted; the second and third arguments show the direction of the conversion. For example,

   `java TempConversion 20.5 c f`
   will convert 20.5 degree Celsius to degree Fahrenheit.
   While
   `java TempConversion 96.4 f c`
   will convert 96.4 degree Fahrenheit to degree Celsius.

3) Import **`lab2data.txt`** into your project. Write a program called **`ReadFile.java`** to read the imported file and do the following tasks:
   - Print out total number of lines.
   - Print out total number of sentences.
   - Print out total number of words.
   - Print out total number of each letter ‘a’ to ‘z’ regardless of case.

Lab 2 - Using Java API documents, command-line arguments, and file I/O.

Task	Description	Result	Note
1	Create a file using Java API doc		
2	CommandLineArgument.java		
3	FileOutput.java		
4	PrintFiles.java		
5	KeyboardInput.java		
6	Sort.java		
7	TempConversion.java		
8	ReadFile.java		