UNIVERSITY OF CALGARY

Department of Computer Science

Tutorial (Week-2)

CPSC 319 - Data Structures, Algorithms, and Their Applications

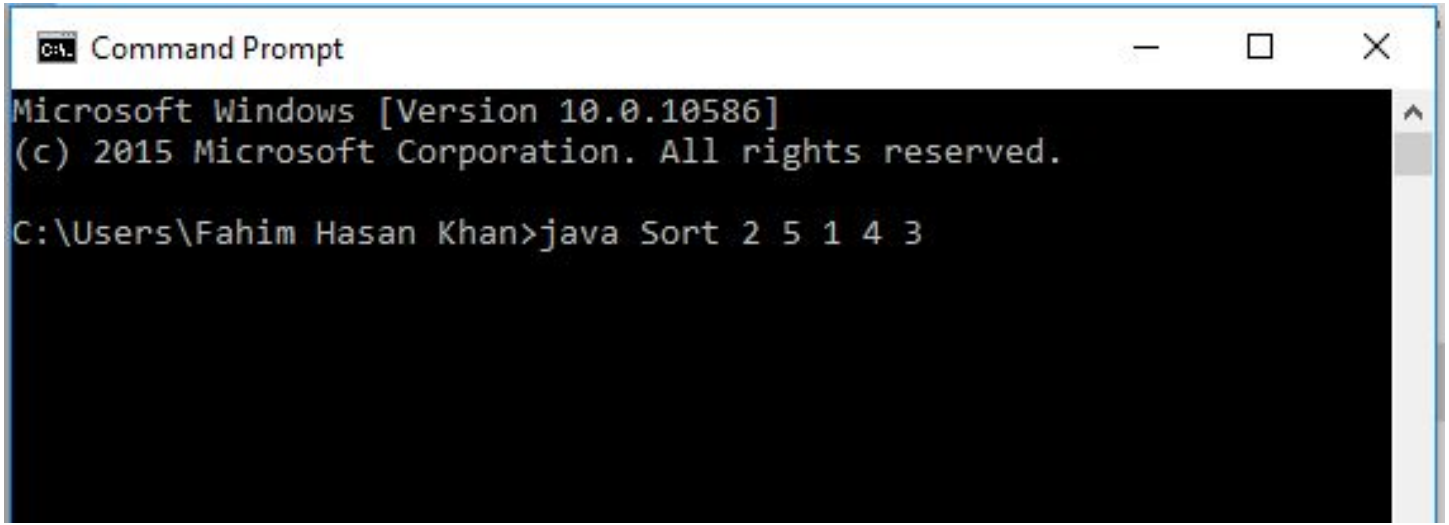
Contents

Command-line Arguments

- A Java application can accept any number of arguments from the command-line.
- Command-line arguments allow the user to affect the operation of an application.
- The user enters command-line arguments when invoking the application and specifies them after the name of the class to run.

Command-line Arguments

For example, suppose you have a Java application, called Sort, that sorts five numbers, you run it like this:



Note: The arguments are separated by spaces.

How Java Application Receive Command-line Arguments

In Java, when you invoke an application, the runtime system passes the command-line arguments to the application's main method via an array of Strings.

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

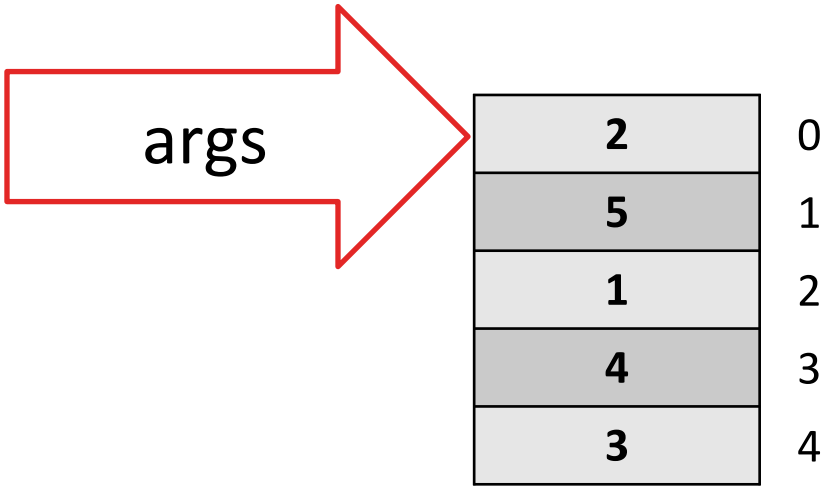
Each String in the array contains one of the command-line arguments.

args[ ] array

Given the previous example where we run:

```
java Sort 2 5 1 4 3
```

the arguments are stored in the args array of the main method declaration.



Command-line Arguments Example

To print the array of arguments, we write:

```
public class CommandLineSample {

    public static void main( String[] args ){

        for(int i=0; i<args.length; i++){
            System.out.println( args[i] );
        }
    }
}
```

Conversion of Command-line Arguments

- If your program needs to support a numeric command-line argument, it must convert a String argument that represents a number, such as "34", to a number.
- Here's a code snippet that converts a command-line argument to an integer,

```
int firstArg = 0;
if (args.length > 0){
    firstArg = Integer.parseInt(args[0]);
}
```

- The parseInt() method in the Integer class throws a NumberFormatException (ERROR) if the format of args[0] isn't valid (not a number).

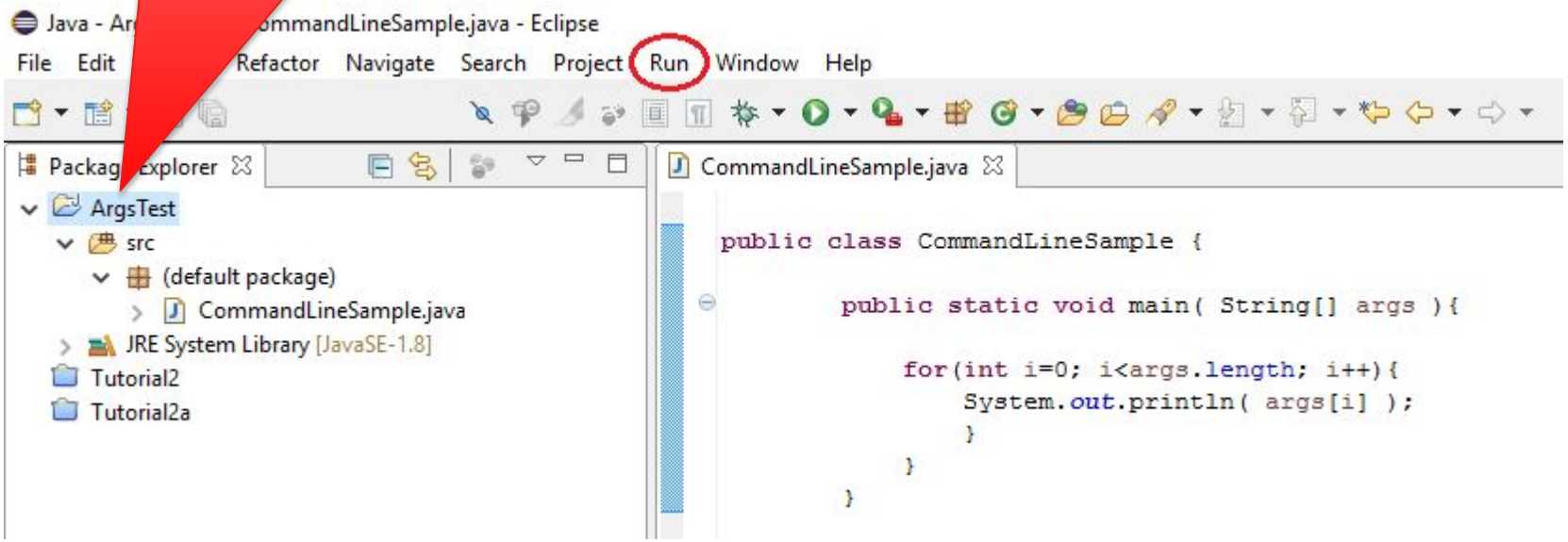
Command-line Arguments: Coding Guidelines

- Before using command-line arguments, always check the number of arguments before accessing the array elements
- so that there will be no exception generated. For example, if your program needs the user to input 5 arguments,

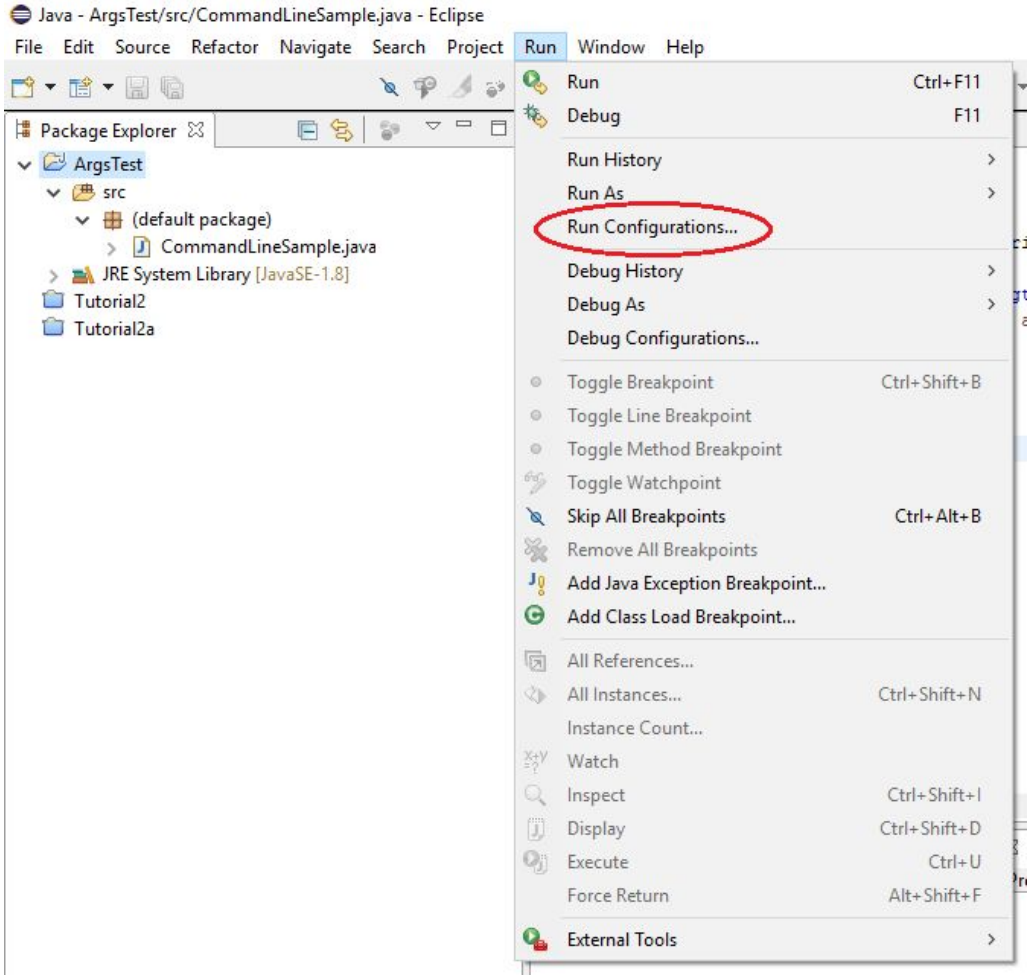
```
if( args.length!= 5 ){
      System.out.println("Invalid number of arguments");
      System.out.println("Please enter 5 arguments");
      }
else{
      //some statements here
}
```

Command-line Arguments in Eclipse

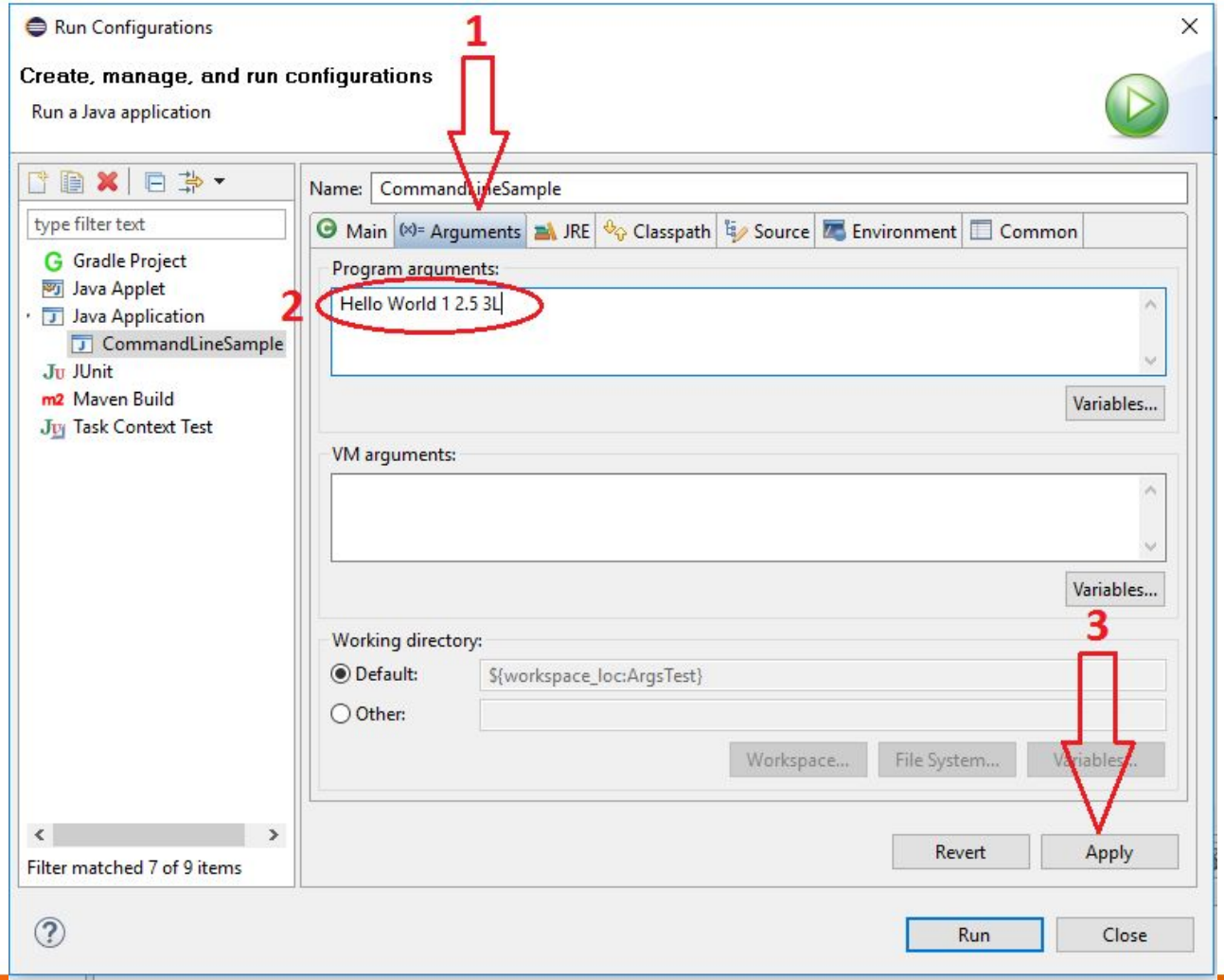
Make sure that the right project is selected



Command-line Arguments in Eclipse



Command-line Arguments in Eclipse



The `Random` class

- A `Random` object generates pseudo-random numbers.
  - Class `Random` is found in the `java.util` package.

```
import java.util.*;
```

Method name	Description
`nextInt()`	returns a random integer
`nextInt(`***max***`)`	returns a random integer in the range [0, *max*) in other words, 0 to *max*-1 inclusive
`nextDouble()`	returns a random real number in the range [0.0, 1.0)

  - Example:

```
Random rand = new Random();
int randomNumber = rand.nextInt(10);    // 0-9
```

Generating random numbers

- Common usage: to get a random number from 1 to *N*

```
int n = rand.nextInt(20) + 1;    // 1-20 inclusive
```

- To get a number in arbitrary range [*min*, *max*] inclusive:

  name `.nextInt(`***size of range***`) + ` ***min***

  - Where ***size of range*** is (***max*** `-` ***min*** `+ 1`)

  — Example: A random integer between 4 and 10 inclusive:

```
int n = rand.nextInt(7) + 4;
```

Random questions

- Given the following declaration, how would you get:

```
Random rand = new Random();
```

  - A random number between 1 and 47 inclusive?

```
int random1 = rand.nextInt(47) + 1;
```

  - A random number between 23 and 30 inclusive?

```
int random2 = rand.nextInt(8) + 23;
```

  - A random even number between 4 and 12 inclusive?

```
int random3 = rand.nextInt(5) * 2 + 4;
```

Generating random numbers (Alternative)

If you need to generate numbers from Min to Max, you write

```
Min + (Math.random() * (Max - Min))

Min + (int)(Math.random() * ((Max - Min) + 1))
```

Example

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

    int[] numlist = new int[1000];

    for(int i=0;i<numlist.length;i++){
            numlist[i] =(int )(Math.random() * ((120 - 5) +
   1));
            System.out.println(numlist[i]);
    }
}
```

Calculating the Running Time of a Code Segment

Two simple options,

Code Format	Remarks

long startTime = System.currentTimeMillis();
 	
//Code Segment
 	
long estimatedTime = System.currentTimeMillis() - startTime;
	Time in Millisecond

long startTime = System.nanoTime();
 	
//Code Segment
 	
long estimatedTime = System.nanoTime() - startTime;
	Time in Nanosecond

Example

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

            int[] numlist = new int[1000];

            long startTime = System.currentTimeMillis();
            long startTime1 = System.nanoTime();

            for(int i=0;i<numlist.length;i++)
                numlist[i] =(int )(Math.random() * ((120 - 5) + 1));

            long estimatedTime = System.currentTimeMillis() - startTime;
            long estimatedTime1 = System.nanoTime() - startTime1;

    System.out.println("Running Time: "+ estimatedTime +" Milli Seconds");
    System.out.println("Running Time: "+ estimatedTime1 +" Nano Seconds");
        }
}
```

Practice

Print a random permutation of args in Java