Java Programming

— Exception handling and File I/O —

Waseda University

Today's topic

- Command-Line arguments

  How to pass command line arguments to Java program

- Exception handling

  When an error occurs in a program, then the program can call the statements for an exception.

- File I/O

  Input or Output to a file

Command-Line arguments

Command-Line arguments

The program can accept any number of arguments from the command line.

Example

```
$ java␣SampleArgs␣aaa␣bbb␣ccc ↵
```

$$\Downarrow$$

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

- Command-line arguments are received in the program and assigned to the variable `args`.
- The data type of `args` is a `String`.

Example of command line

SampleArgs.java

```java
public class SampleArgs{
  public static void main(String[] args) {
    for(int i=0;i<args.length;i++){
      System.out.println("args[" + i + "] : " + args[i]);
    }
  }
}
```

```
$ java␣SampleArgs␣abc␣123␣4.5 ↵
args[0] : abc
args[1] : 123
args[2] : 4.5
```

- The program can accept any number of arguments, and the space character separates command-line arguments.
- The length of command-line arguments is `args.length`.

Convert String to number (1)

Aim	Method
Converting `s` to `int`	`Integer.parseInt(String s)`
Converting `s` to `double`	`Double.parseDouble(String s)`

```
String s1 = "123", s2 = "4.5";
int x1;
double x2;

x1 = Integer.parseInt(s1);            — Convert s1 to int
x2 = Double.parseDouble(s2);          — Convert s2 to int
System.out.println(x1);               — display x1
System.out.println(x2);               — display x2
```

Result

```
123
4.5
```

Convert String to number（2）

Example

Give two values to your program using command-line arguments and calculate the sum of the values.

SumOfDouble.java

```
public class SumOfDouble {
  public static void main(String[] args) {
    double x1, x2;

    if (args.length == 2) {
      x1 = Double.parseDouble(args[0]);
      x2 = Double.parseDouble(args[1]);
      System.out.println(x1 + x2);
    } else {
      System.out.println("error:two arguments are required.");
    }
  }
}
```

Exception handling (1)

If you access an array out of bounds....

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: **
        at ****.main(sample.java:**)
```

- Then, the exception occurs and the program ends.

$$\Downarrow$$

Exception handling

When an exception occurs in `try` block, then we can treat an exception in `catch` block.

Exception handling (2)

try-catch

```
try {
    An exception may occur in this block
}
catch (Exception e) {
    Exception handling
}
```

- First, Java executes statements in `try` block.
- If there is no exception in `try` block, then statements in `catch` block are not executed.
- If an exception is thrown in `try` block, then statements(Exception handling) in `catch` block are executed.
- An exception is assigned to `e`.

Exception handling (3)

inputInt method

```
public static int inputInt() {
  int a = 0;

  try{
    Scanner sc = new Scanner(System.in);
    a = sc.nextInt();
  }
  catch (Exception e) {
    e.printStackTrace();
    System.exit(1);
  }

  return a;
}
```

File Input (1)

SampleFile.java

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

public class SampleFile{
  public static void main(String[] args) {
    try {
        File file = new File(args[0]);
        Scanner sc = new Scanner(file);
        while( sc.hasNextLine() ){
            System.out.println(sc.nextLine());
        }
        sc.close();
    }
    catch (Exception e) {
        e.printStackTrace();
        System.exit(1);
    }
}
```

File Input (2)

`sc.nextLine()`

- `sc.nextLine()` returns the rest of the current line, excluding any line separator at the end.

`while( sc.hasNextLine() ){`

`hasNextLine()`
– returns **`true`** if there is another line after the current line in the file.
– returns **`false`** otherwise. This means that every contents in the file can be read.

File Output (1)

SampleFileOut.java

```
import java.io.*;

public class SampleFileOut {
  public static void main(String[] args) {
    String filename = "sample_number.txt";
    String line;
    try {
      FileWriter fw = new FileWriter(filename);
      BufferedWriter bw = new BufferedWriter(fw);
      PrintWriter pw = new PrintWriter(bw);

      for(int i=1;i<=10;i++){
        System.out.println(i*100);
        pw.println(i*100);
      }
      pw.close();
    }
    catch (Exception e) {
      System.out.println(e);
      System.exit(1);
    }
  }
}
```

File Output (2)

```
String filename = "sample_number.txt";
```

- Assign `sample_number.txt`” to `filename`

```
FileWriter fw = new FileWriter(filename);
BufferedWriter bw = new BufferedWriter(fw);
PrintWriter pw = new PrintWriter(bw);
```

- Preparation for reading the sample_number.txt.

```
pw.println(argument);
```

- This method writes an argument to a file.