Programming 1

Lecture 9 – File Input/Output (1)

Faculty of Information Technology
Hanoi University

Contents

- **Basic file operations**
  - `java.io.File`
- **Reading text files**
  - `java.util.Scanner`
- **Writing text files**
  - `java.io.PrintWriter`

Types of Input/Output

- Input
  - Input devices (keyboard/mouse/mic…)
  - Text files
  - Binary files (zip, images, serialized data…)
  - Network (e.g. database system, web page, network stream)
- Output
  - Output devices (screen/speakers/printer…)
  - Files
  - Network

Types of computer memory

- Data stored inside a program's variables will be lost when the program terminates.
  - A running program is called a process.
  - The data of a process is stored in RAM, which is a type of volatile (short-term) memory.
- We often need to save data as files…
  - Examples: text documents, images, audio, video…
  - Using a non-volatile (long-term) memory such as HDD, SSD, CD, DVD, Floppy Disk, USB Flash Memory…

The `java.io.File` class

- A basic class to represent a file.
  - A File object only stores information about a file or directory. E.g. path,
  - It does not contain the file's data.
- A File object has methods to perform basic operations.
  - `boolean` **`createNewFile()`**: create a file from the information stored in the object.
    It may throw an IOException (needs to be caught)
  - `boolean` **`mkdir()`**: create a directory from the information stored in the object.

`File` path

- Two types: *absolute* and *relative*.
  - **Absolute:** the full path to the file/folder.
  - **Relative:** takes program's working directory as base. *(relative path is recommended)*
- Working directory is usually the project's root.
  To find out the current effective working directory:

```
File f = new File("someFileName.ext");
System.out.println(f.getAbsolutePath());
```

```
C:\Users\Quan\IdeaProjects\F2022_PR1\someFileName.ext
```

working directory

java.io.File example

- Create an empty file at the working directory

```
import java.io.File;
import java.io.IOException;
public class CreateFile {
    public static void main(String[] args) {
        File f = new File("myfile.txt");
        try {
            f.createNewFile();
        } catch (IOException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

`java.io.File` example

- Create a directory at the working directory

```
import java.io.File;
public class CreateDir {
    public static void main(String[] args) {
        File f = new File("mydir");
        f.mkdir();
    }
}
```

- Although `mkdir()` does throws `SecurityException`, it's not compulsory to handle it.

`java.io.File` example

- Create a directory at the working directory
  - Print a success or failure message.
  - `mkdir()` returns `false` when it fails.

```
import java.io.File;
public class CreateDir2 {
    public static void main(String[] args) {
        File f = new File("mydir");
        if (f.mkdir()) {
            System.out.println("Directory created!");
        } else {
            System.out.println("Failed to create directory!");
        }
    }
}
```

The `java.io.File` class

- `File` methods (continue)
  - `String` **`getName()`**: the file's name
  - `String` **`getAbsolutePath()`**: the file's full path
  - `boolean` **`isDirectory()`**: return `true` if the path stored in the object is a directory (folder)
  - `boolean` **`isFile()`**: return `true` if the path stored in the object is a file

`java.io.File` example

- Check if a path is a directory or file

```
import java.io.File;
public class FileOrDir {
    public static void main(String[] args) {
        File f = new File("mydir");
        System.out.print(f.getAbsolutePath() + " is ");
        if (f.isDirectory()) {
            System.out.println("a directory.");
        } else if (f.isFile()) {
            System.out.println("a file.");
        } else {
            System.out.println("non-existant.");
        }
    }
}
```

The `java.io.File` class

- `File` methods (continue)
  - `boolean renameTo(File dest)`: rename/move a file to a new path
  - `boolean exists()`: see if a file/directory exists
  - `boolean delete()`: delete the file itself
  - `long length()`: the file size in bytes

java.io.File example

- Rename a file

```
import java.io.File;
public class RenameFile {
    public static void main(String[] args) {
        File src = new File("a.txt");
        File dest = new File("b.txt");
        if (!dest.exists()) {
            if (src.renameTo(dest)) {
                System.out.println("Rename successful");
            } else {
                System.out.println("Rename failed");
            }
        } else {
            System.out.println("Destination exists");
        }
    }
}
```

java.io.File example

- Move a file

```
public class MoveFile {
    public static void main(String[] args) {
        File src = new File("a.txt");
        File dest = new File("some_dir/a.txt");
        if (!dest.exists()) {
            if (src.renameTo(dest)) {
                System.out.println("Move successfully");
            } else {
                System.out.println("Failed to move file");
            }
        } else {
            System.out.println("Destination exists");
        }
    }
}
```

`java.io.File` summary

- What it can do:
  - Create/delete/rename/move files and directories
  - Get full path, file size, permissions (read/write/execute)
  - Recognize file and directory
  - Check for existance
- What it's not capable of:
  - Read file's content
  - Write content to a file.

Read files with `java.util.Scanner`

- Other than `System.in`, `Scanner` accepts a `File` object as its input.
- Example file's content:

File: **data.txt**

```
15
16
FIT-PR1
2.4
```

Read files with `java.util.Scanner`

- Other than `System.in`, `Scanner` accepts a `File` object as its input.

File: **data.txt**

```
15↵
16↵
FIT-PR1↵
2.4
```

```
import java.util.Scanner;
public class ScannerRead {
    public static void main(String[] args) throws IOException {
        File f = new File("data.txt");
        Scanner sc = new Scanner(f);
        int a = sc.nextInt(); // get first integer
        int b = sc.nextInt(); // get second integer
        sc.nextLine(); // clears a newline character
        String s = sc.nextLine(); // "FIT-PR1"
        double d = sc.nextDouble();
        System.out.println(a + "\n" + b + "\n" + s + "\n" + d);
    }
}
```

Read files with `java.util.Scanner`

- Read the whole file to a String.

```
import java.io.File;
import java.io.IOException;
import java.util.Scanner;
public class ScannerReadText {
    public static void main(String[] args) throws IOException {
        Scanner sc = new Scanner(new File("text"));
        String s = "";
        while (sc.hasNext()) {
            s += sc.nextLine() + "\n";
        }
        System.out.println(s);
    }
}
```

Read files with `java.util.Scanner`

- Read text file to a String using StringBuilder for better performance.

```
public class ScannerReadText2 {
    public static void main(String[] args) throws Exception {
        Scanner sc = new Scanner(new File("some.txt"));
        StringBuilder sb = new StringBuilder();
        while (sc.hasNextLine()) {
            sb.append(sc.nextLine());
            sb.append(System.lineSeparator());
        }
        String s = sb.toString();
        System.out.println(s);
    }
}
```

`java.util.Scanner` summary

- `boolean` **`hasNext()`**: return `true` if there is any token in the input buffer.
- `boolean` **`hasNextLine()`**: return `true` if there is any line in the input buffer.
- `boolean` **`hasNextInt()`**: return `true` if there is any integer left in the input buffer.

The `java.io.PrintWriter` class

- Contains methods necessary for writing formatted data to files.
- Syntax to create a PrintWriter object:

```
PrintWriter out = new PrintWriter("data.txt");
```

- **Attention:** If the file exists, its content will be erased.
- The statement to create a `PrintWriter` object may throw `FileNotFoundException` and needs to be surrounded with `try…catch` (or declared to be thrown)

The `java.io.PrintWriter` class

- Methods in the PrintWriter class:
  - `void` **`print()`**: write text or other values to the output file.
  - `void` **`println()`**: similar to **`print()`** but adds a newline character at the end.
  - `void` **`write()`**: write a single character, a char array or a String to the file.
  - `void` **`printf()`**: similar to `System.out.format()`.
  - `void` **`format()`**: the same as **`printf()`**.

The `java.io.PrintWriter` class

- Methods in the PrintWriter class:
  - `void` **`flush()`**: transfer the content from the buffer to the file.
    - Without flushing, texts are not written to the file.
  - `void` **`close()`**: flush and close the connection to the file, as well as release any system resources being used by the PrintWriter object.
    - Typically, you only need to invoke **`close()`** when you have finished writing everything to the file.

Example

Use PrintWriter to write some content to a text file.

```
public class PrintWriterDemo {
    public static void main(String[] args) {
        try {
            PrintWriter pw = new PrintWriter("info.txt");
            pw.println("My info:");
            pw.println("Name: Dang Dinh Quan");
            pw.println("Organization: Hanoi University");
            pw.close();
        } catch (FileNotFoundException e) {
            System.out.println("Cannot write to file!");
        }
    }
}
```