

# Software Development II

## Lecture 8— Files and Testing

*Reading List : Java for everyone Ch8*

# From Last Week

## Classes and Objects

- State and behaviour
- Declaration and use
- Class constants and variables
- Constructors
- *this*
- Arrays with objects
- Class methods
- Methods declaration and use
- Public vs private
- Field modifiers
- Classes with other classes

# This week

## Files

- What is a file
- How to create a file
- How to write in a file
- How to read a file
- How to rename a file
- How to delete a file
- Directories

## Testing

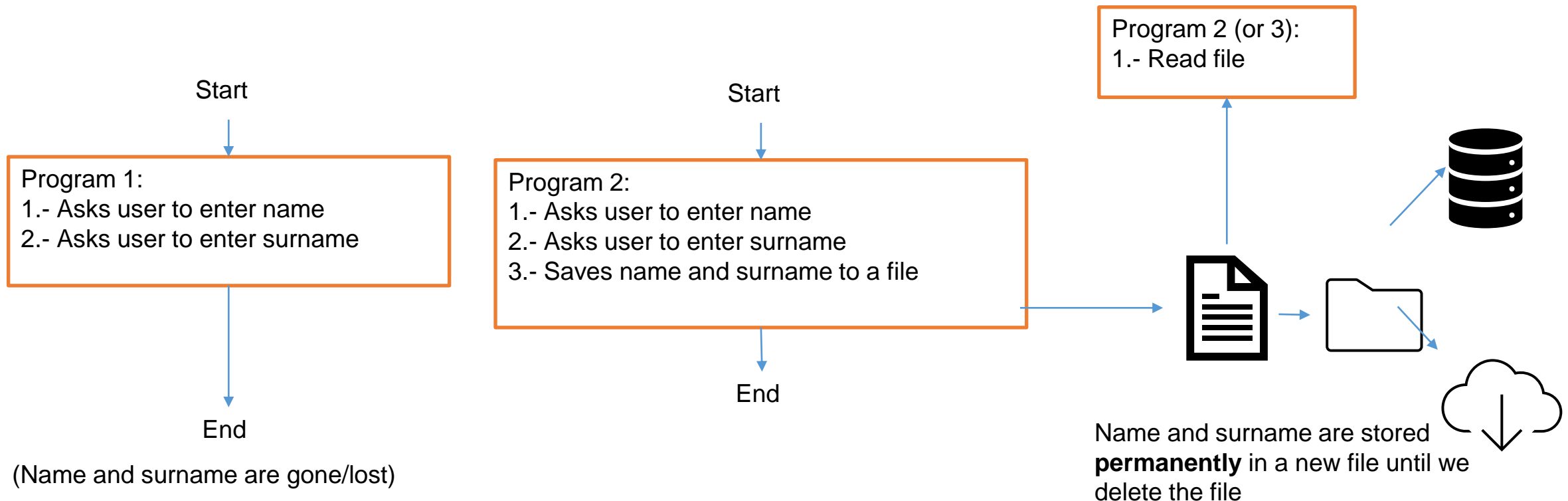
- What is testing
- Why do we need testing
- Types of testing
- Black box testing
- White box testing
- Integration testing

# File Handling Recap

- This section was available as a self study exercise in Blackboard during the Engagement week (GIS week) with a **self guided video lecture recording and slides**.
- **Please refer to Week 06 Blackboard content if you haven't referred it so far**
- Within this lecture, we will revise the file related concepts.

# What is a file?

- A file is a named location (document, folder) where you can **store** information.
- There are many file types: text file, Word document, picture, voice, etc.
- You can create, read, update and delete files (and directories) in Java.
- A directory is a location that can contain multiple files.



# Class File

We will use the class `File` from `java.io` to work with files:

```
import java.io.File; ← Import the File class
```

Create a *File* variable (this does NOT create a file):

```
File my_file = new File("text.txt"); ← Name of the file
```

```
File my_file_2 = new File("C:\\Users\\user_name\\text.txt"); ← Name and directory  
of the file
```

# Class File

The *File* class has several methods to work with files:

```
File file = new File("text.txt");
```

Method	Returns	Description	Example
<code>canRead()</code>	Boolean	Checks if the file is readable	<code>boolean readable = file.canRead();</code>
<code>canWrite()</code>	Boolean	Checks if the file is writable	<code>boolean writable = file.canWrite();</code>
<code>createNewFile()</code>	Boolean	Creates an empty file	<code>boolean var = file.createNewFile();</code>
<code>delete()</code>	Boolean	Deletes a file	<code>boolean deleted = file.delete();</code>
<code>exists()</code>	Boolean	Checks if the file exists	<code>boolean exists = file.exists();</code>
<code>getName()</code>	String	Returns the name of the file	<code>String name = file.getName();</code>
<code>getAbsolutePath()</code>	String	Returns the absolute pathname	<code>String path = file.getAbsolutePath();</code>
<code>length()</code>	Long	Returns the size of the file in bytes	<code>long length = file.length();</code>
<code>list()</code>	String[]	Returns a list of the files in a directory	<code>String[] list = file.list();</code>
<code>mkdir()</code>	Boolean	Creates a directory	<code>bool ok = file.mkdir();</code>

# Files: Create a file

```
import java.io.File;
import java.io.IOException;

public class Files {
    public static void main(String[] args) {

        try{
            File file = new File("text.txt");
            boolean file_created = file.createNewFile();
            if (file_created){
                System.out.println("File created: " + file.getName());
            }
            else{
                if (file.exists()){
                    System.out.println("File already exists.");
                }
                else{
                    System.out.println("Error while creating file: " + file.getName());
                }
            }
        }
        catch(IOException e){
            e.printStackTrace();
        }
    }
}
```

← Imports: *File* and *IOException* for the *try/catch* statement

← File handling always in a *try/catch* statement

← Try to create a file

← If file not created, we check if already existed

← File handling always in a *try/catch* statement



# Files: Write to a file

You can write in a file using a *FileWriter*:

```
import java.io.File;
import java.io.IOException;
import java.io.FileWriter; ← Import FileWriter

public class Files {
    public static void main(String[] args) {
        try {
            FileWriter file = new FileWriter("text.txt");
            file.write("Hello world!"); ← Write "Hello world!" in a file
            file.close(); ← We close the file
        } catch (IOException e) {
            System.out.println("Error while writing in a file.");
            e.printStackTrace();
        }
    }
}
```

Create a *FileWriter*

File handling always in a *try/catch* statement

# Files: Read a file

You can read from a file by using *Scanner* (similar to keyboard input):

```
import java.io.File;
import java.io.IOException;

public class Files {
    public static void main(String[] args) {

        try {
            File file = new File("text.txt");
            Scanner file_reader = new Scanner(file); ← Write create a Scanner passing file as parameter
            while (file_reader.hasNextLine()) {
                String text = file_reader.nextLine(); ← We read line by line in a while loop
                System.out.println(text); ← We print the text from the file
            }
            file_reader.close(); ← We close the file
        } catch (IOException e) {
            System.out.println("Error while reading a file.");
            e.printStackTrace();
        }
    }
}
```

} File handling always in a *try/catch* statement

# Rename a file

- You can rename a file by creating two variables `File`:

`File file = new File("MyFile.txt");` ← Create a *File* with the old name

`File file_newName = new File("MyNewNameFile.txt");` ← Create a *File* with the new name

`file.renameTo(file_newName);` ← Call the `renameTo` method on the first *File* with the second *File* as parameter.

# Files: Delete a file

You can delete files:

```
import java.io.File;
import java.io.IOException;

public class Files {
    public static void main(String[] args) {

        File file = new File("MyFile.txt");
        if (file.exists()) { ←————— Check if file exists
            file.delete(); ←————— Delete file
            System.out.println("File deleted.");
        }
        else{
            System.out.println("File does not exist.");
        }
    }
}
```

# Create a directory (folder)

- To create a directory:

```
File myDirectory = new File("../path/directory");  
  
if (!myDirectory.exists()) {  
    myDirectory.mkdirs();  
}
```

← Create a *File* with the directory we want to create

← Check if the directory exists

← Create the directory

# List files in a directory

- You can list all the files in a directory

```
File directory = new File("./");  
  
String[] files = directory.list();  
  
for(int i = 0; i < files.length; i++){  
    System.out.println(files[i]);  
}
```

← Create a *File* with the directory we want to list the files

← We obtain the list of filenames as Strings

← Print file name

- Once you have a list of filenames, you can:
  - Open them individually
  - Delete them
  - Rename them
  - Etc.

# Files: Delete a directory

You can also delete folders:

```
import java.io.File;
import java.io.IOException;

public class Files {
    public static void main(String[] args) {

        File folder = new File("C:\\Users\\user_name\\folder"); ← Folder name instead of
                                                                    file name
        if (folder.exists()) { ← Check if folder exists
            folder.delete(); ← Delete folder
            System.out.println("Folder deleted.");
        }
        else{
            System.out.println("Folder does not exist.");
        }
    }
}
```

# Exceptions (Recap)

- This section was discussed under loops and Debugging lecture.
- Will try to recap this section with File based exceptions as well.





# Catching exceptions

- Java programs have errors at running time and terminate the program.
- E.g., When we read an integer with `.nextInt()` but we enter some characters:

```
Exception in thread "main" java.util.InputMismatchException
    at java.base/java.util.Scanner.throwFor(Scanner.java:939)
    at java.base/java.util.Scanner.next(Scanner.java:1594)
    at java.base/java.util.Scanner.nextInt(Scanner.java:2258)
    at java.base/java.util.Scanner.nextInt(Scanner.java:2212)
    at com.mycompany.tutorial5.FilesAndTesting.main(FilesAndTesting.java:22)
Command execution failed.
```

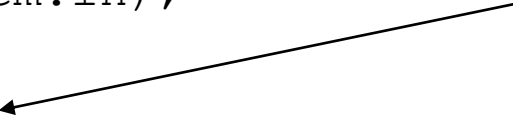
- We will use a try and catch statement.
- When there is an exception, we say an exception is thrown.
- When a matching exception-handling code is executed, the exception is caught.
- When the exception is thrown, it will say which type of exception.

# Catch Exception

- You can catch exceptions with a try and catch statement:

```
public static void main(String[] args) {  
    try {  
        System.out.println("Please enter a number: ");  
        Scanner input = new Scanner(System.in);  
        int number = input.nextInt();  
    } catch (InputMismatchException e) {  
        System.out.println("Error, wrong type of input, I was expecting an int.");  
    }  
}
```

This catch handles  
InputMismatchException






## Output:

```
Please enter a number:  
abc  
Error, wrong type of input, I was expecting an integer.
```

# File Exceptions

- You can use multiple catch statements to handle different Exceptions

```
try {  
    File file = new File("MyFile");  
    Scanner reader = new Scanner(file);  
    String text = reader.nextLine();  
  
} catch (FileNotFoundException e) {  This catch handles FileNotFoundException  
  
    System.out.println("I could not find the file");  
  
} catch (IOException e) {  This catch handles all IOException, except  
    FileNotFoundException  
  
    System.out.println("IOException");  
  
} catch (Exception e) {  This catch handles all Exceptions  
    except all IOExceptions  
  
    System.out.println("Some other Exception");  
}
```

# TESTING



# Royal Mail scandal

- [https://en.wikipedia.org/wiki/British\\_Post\\_Office\\_scandal](https://en.wikipedia.org/wiki/British_Post_Office_scandal)

*“The British Post Office scandal or Horizon scandal involved **faulty accounting software**, provided by Fujitsu and known as Horizon, creating **false shortfalls in the accounts of thousands of subpostmasters**. The prime minister, Rishi Sunak, described it as one of the **greatest miscarriages of justice in the history of the United Kingdom**. Between 1999 and 2015, over 900 subpostmasters were convicted of theft, fraud and false accounting based on faulty Horizon data, with about 700 of these prosecutions carried out by the Post Office. Other subpostmasters were prosecuted but not convicted, forced to cover Horizon shortfalls with their own money, or had their contracts terminated.* “

- <https://www.bbc.co.uk/news/business-56718036>



# What is testing?

- Is a procedure to ensure that the program works as expected.
- Useful at implementation time and after introducing changes.



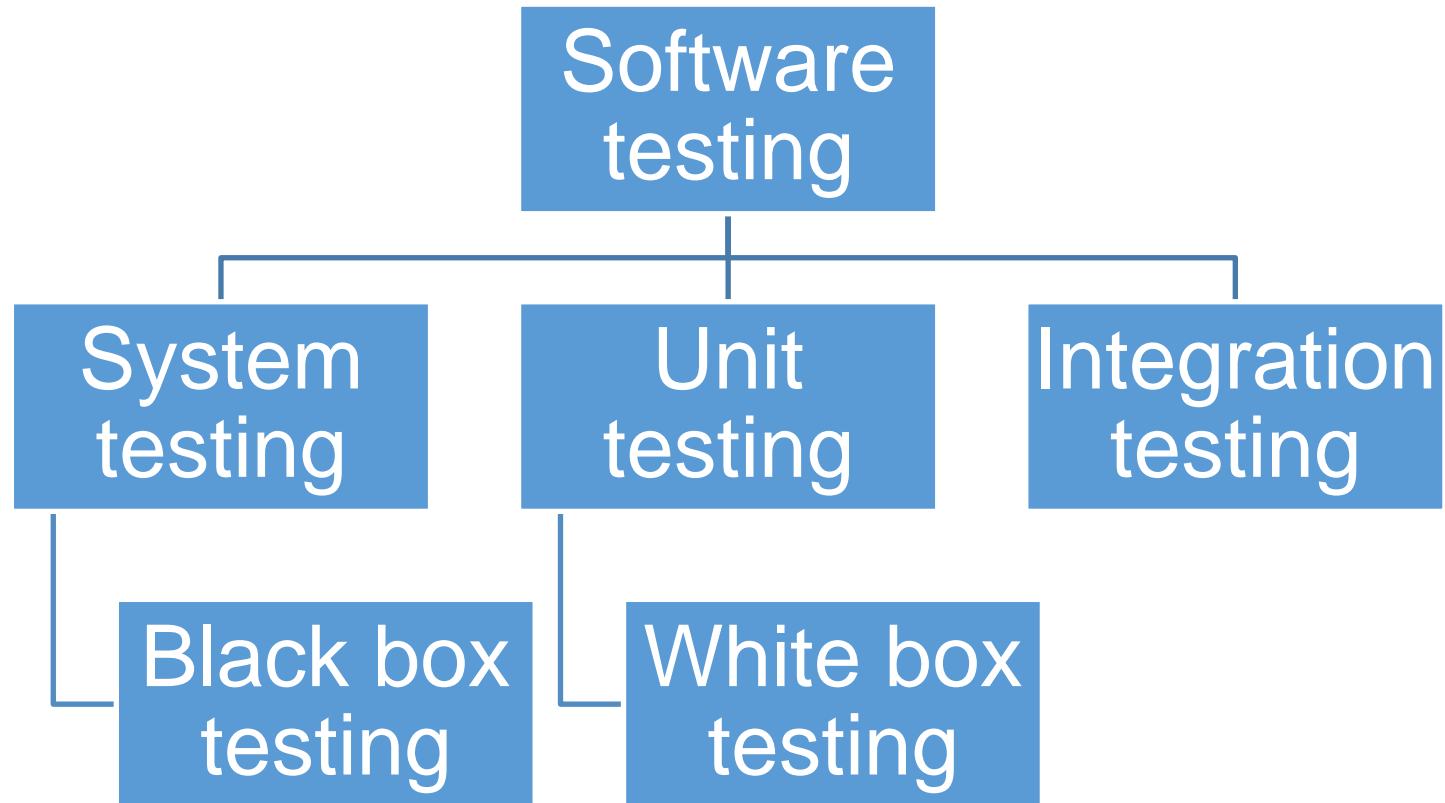
# Why do we need testing?

- Find errors during the implementation manually or automatically
- Ensure quality
- Good user experience
- Save money (less maintenance)



# Types of testing

- Some examples of testing methodologies are:



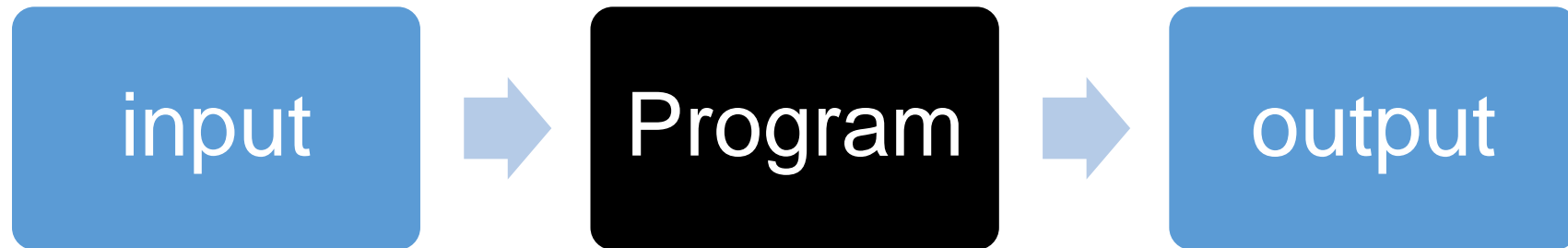


# How to test your program

- Two types of testing
  - Manual testing
  - Automatic testing
- Write a test to check a program functionality:
  - Identify the functionality to be tested
  - Find the correct/expected result
- Run the test
- Check if the result of the test is the expected result
- If the result is not the expected, find the error and correct it (e.g., by debugging your program)

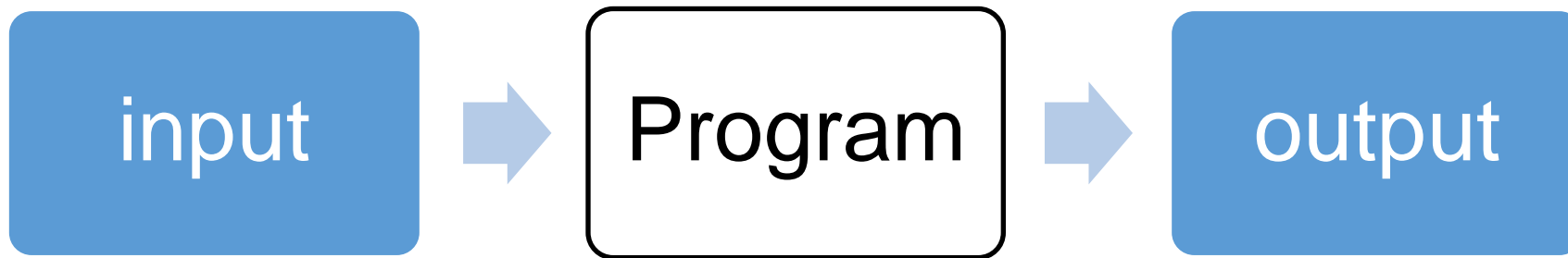
# System testing: Black box testing

- Design a set of tests based on the input and the expected output, without knowing how it works internally
- The code of the program is not available/visible.



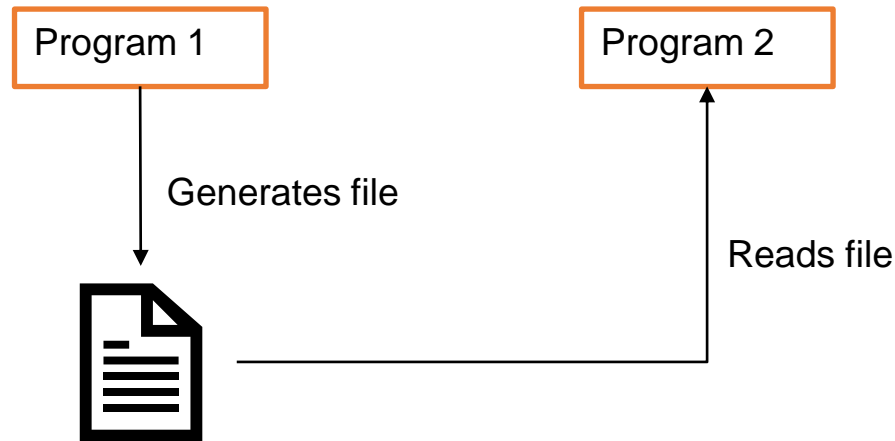
# Unit testing: White box testing

- Testing of a program with access to the code and design documents.
- Allows to identify issues that would be difficult to identify otherwise



# Integration testing

- Tests that multiple programs or methods work as expected.
- Example:



What happens in program 2 if program 1 fails to generate a file?

# Black box testing

You are given a simple program for a calculator, that adds two numbers and returns the result.

Which black box tests would you use to test the program?

# White box testing example

This is the code of the calculator program.

Which white box tests would you perform

```
public static void main(String[] args) {
    Scanner input = new Scanner(System.in);
    System.out.println("Please insert a number: ");
    double num_1 = input.nextDouble();
    System.out.println("Please insert a number: ");
    double num_2 = input.nextDouble();
    double sum = add(num_1, num_2);
    System.out.println(sum);
}

private static double add(double num_1, double num_2)
{
    double sum = num_1 + num_2;
    return sum;
}
```

# Coursework

After this lecture, you should be able to implement:

**Part B: tasks 12.**

**Complete the test plan.**

# Feedback: Formative test week 8 (Blackboard)

- In Learning Resources > Week 8 you will find a formative test to get feedback on the content of this lecture.



Thank You !!