OBJECT ORIENTED ANALYSIS & DESIGN
DATA STRUCTURES & ALGORITHMS

File Handling

LearnOA
To The Next Level...

Why use files?

In all the programs we have written so far, any data we need has been either

- hard coded into the program

```
String name = "Cathy";
```

- or read in from the keyboard using a Scanner object

```
Scanner kybd = new Scanner(System.in);
System.out.println("What is your name?");
name = kybd.next();
```

It is tedious to type in all the data to be processed each time the program is run

It is not very useful if the data and results cannot be saved

- imagine a payroll program where all information about the employees has to be entered each time the program is run

Opening a file to read from

A `Scanner` object can be set up to read from a file

- so far all our `Scanner` objects have read from the keyboard

The name of the file to read from is required

First create a `File` object corresponding to this file

- need to import `java.io.*`
- this is a library of java classes for input and output

Then create a `Scanner` object using the `File` object as the source of the input

- instead of `System.in`

Reading from a file

```
import java.util.*;
import java.io.*;
public class Payroll
{
  public static void main(String args[])
  {
    String fName = "payroll.txt";
    Scanner inFile = new Scanner(new File(fName));
    ……
   or
    Scanner inFile =
       new Scanner(new File("payroll.txt"));
```

Reading from `Scanner` objects

Once the `Scanner` object is created, we can use its methods to read from the file

- just like when reading from the keyboard

```
String name = inFile.next();
double hourlyPay = inFile.nextDouble();
```

Multiple `Scanner` objects can be created, as long as they are given different names

- for example, to read from two different files
- or from a file and the keyboard

```
Scanner kybd = new Scanner(System.in);
Scanner inFile = new Scanner(new File(fName));
```

Reading from a file

The `hasNext()` method is useful when you do not know how much data is in the file

- returns `true` if there is more data to read
- returns `false` if you have reached the end of the file

It can be used in a `while` loop to process all the data in the file

Imagine a text file is available where each line contains information about one employee:

- name (as a `String`)
- followed by hourly pay (as a `double`)

Reading all the data in a file

```
while (inFile.hasNext())
{
  name = inFile.next();
  hourlyPay = inFile.nextDouble();
  System.out.println("Hours worked by "+name+"?");
  hoursWorked = kybd.nextInt();
  double pay = hourlyPay * hoursWorked;
  System.out.println("Pay is " + pay);
}
inFile.close();
```

Closing a file

When you have finished reading from a file, you should close it

A file is closed by calling the `close()` method of the `Scanner` object that was set up to read the file

```
inFile.close();
```

Tips for Input Files

Decide on the format of the data

- repeating rows of
  - name (as a `String`)
  - followed by hourly pay (as a `double`)

Make sure the information is

- in the correct order
- of the correct type

to match the input statements in your program

Any text editor can be used to create and edit the file

or it could be output from a program

- which writes to the file using the correct format

Writing to a file

The name of the file to write to is required

First create a `PrintWriter` object for this file

- need to import `java.io.*`
- same library of java classes as `FileReader`

Typically the output file will be in the same directory as the Java program

If a file of this name already exists

- it will be opened
- all the data currently in the file will be lost

If the file does not already exist, a new one will be created

```
PrintWriter pw = new PrintWriter("Payroll.txt");
```

Files program structure

A program which reads data from a file may do a lot of processing on it

- do calculations (totals, averages)
- add to it (input by user)
- delete some of it
- sort it
- search all of it for a particular data value

It is awkward to search for and retrieve only the required data from a sequential file for each process

and to write changes back to the original file

Files program structure

It is sometimes better to

- open the input file
- read all the data in the file into an appropriate data structure
  - such as an array, or several arrays
- close the input file

Do all the processing in memory, then write the final version of the data to a file

- either with the same name as the input file
  - original data is lost
- or a new file

Most of the program (data structures, processing) is the same as when all the data is entered via the keyboard

- just add the file-reading code at the beginning of the program
- and the file-writing code at the end

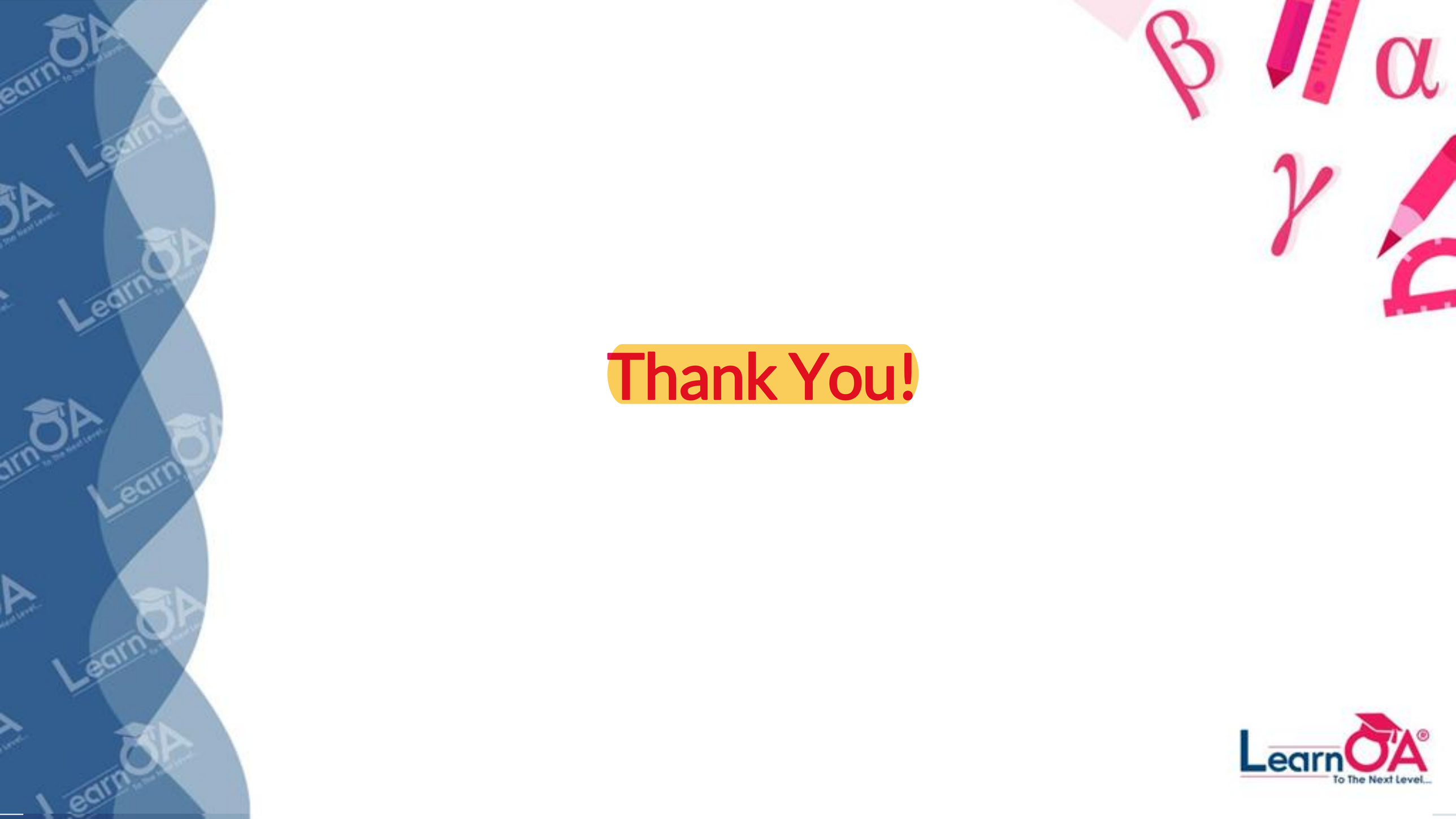
Files and Exceptions

File handling is one area that is prone to things going wrong

- A file may not exist
- A file may not be accessible
- The format of the data in the file may be incorrect

Whenever dealing with files it is best to make use of try catch blocks to handle any exceptions

- Try to anticipate what could go wrong


Thank You!