COM1102

**Programming and**

**Data Structures**

**Object Oriented Programming using Java**

**Part C**

**Handling Exceptions**

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

**int** i = 0;

String spanish [] = { "Uno", "Dos","Tres" };

System.out.println("Please enter a number (1-3)");

Scanner in = new Scanner(System.in);

int num = in.nextInt();

System.out.println("Your number in Spanish is " + spanish[num-1]);

}

* What happens if the user enters other numbers (e.g., 4)?
* What happens if the user enters a non-number?

**Handling Exceptions**

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

**int** i = 0;

String spanish [] = { "Uno", "Dos","Tres" };

System.out.println("Please enter a number (1-3)");

**try** {

Scanner in = new Scanner(System.in);

**Normal flow**

int num = in.nextInt();

System.out.println("Your number in Spanish is " + spanish[num-1]);

}

**catch** (ArrayIndexOutOfBoundsException e){

**Error occured!**

System.out.println("Error! Number must be between 0 and 3 ");

}

**catch** (InputMismatchException e){

**Error occured!**

System.out.println("Error! Please enter a number (0-3)");

}

System.out.println("Good Bye!");

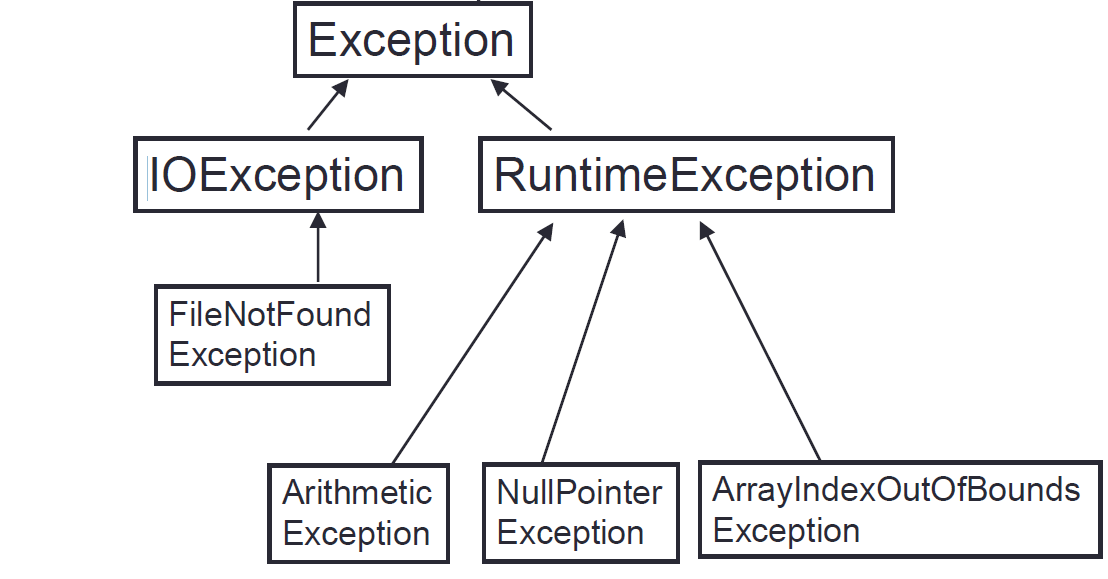
}

Common Exceptions (provided by Java)

• ArithmeticException

• NullPointerException

• ArrayIndexOutOfBoundsException



Define your own Exception (Reference)

/\* Define the TeacherNotHappyException \*/

**Exception defined**

**public class** **TeacherNotHappyException** **extends** RuntimeException {

**public** **TeacherNotHappyException**() {

**super**("Oh no! CK is not happy. Program must stop! ");

}

}

/\*A method that *may* throw the Exception \*/

**public void** MyMethod() **throws** **TeacherNotHappyException** {  
 if (CK.mood != “Happy”)

**Exception throwed**

**throw** **new** **TeacherNotHappyException();**

}

**public void** AnotherMethod(){ /\*Another method that callsMyMethod, and may get the exception \*/  
 **try** {myObject.MyMethod(); **}**

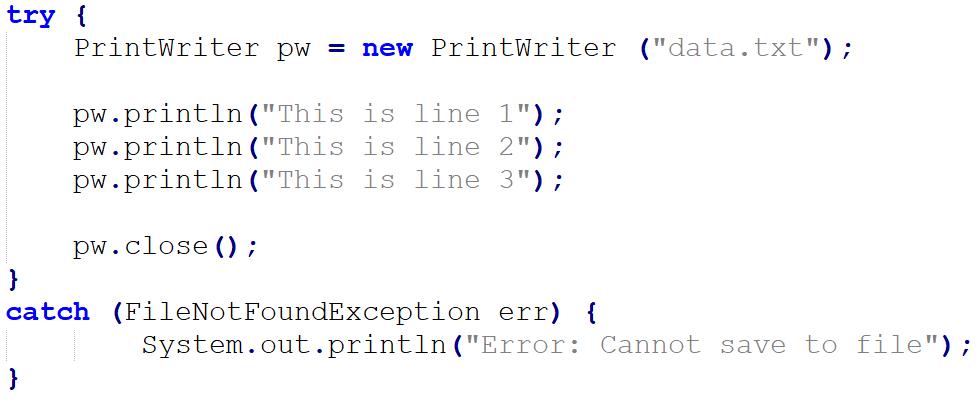
**Exception caught**

**catch (TeacherNotHappyException e) {** System.out.println**(“**Error Detected:**” +** e.getMessage()**); }**

}

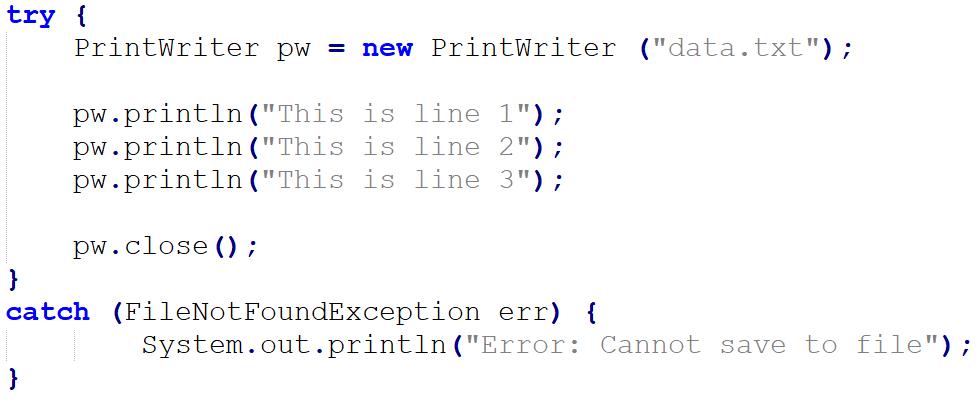
**File IO  
Writing to a file using PrintWriter**

PrintWriter is defined in java.io

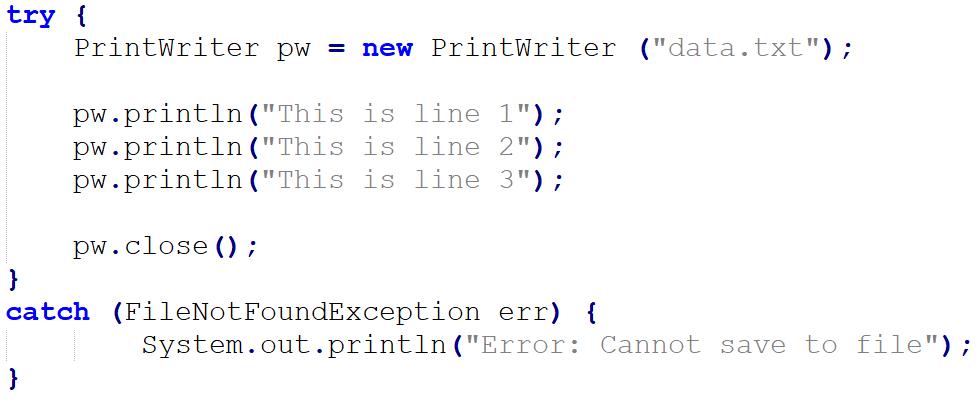


**The try part is executed to the end if everything is OK …**

1. Create a PrintWriter Object for writing to the file



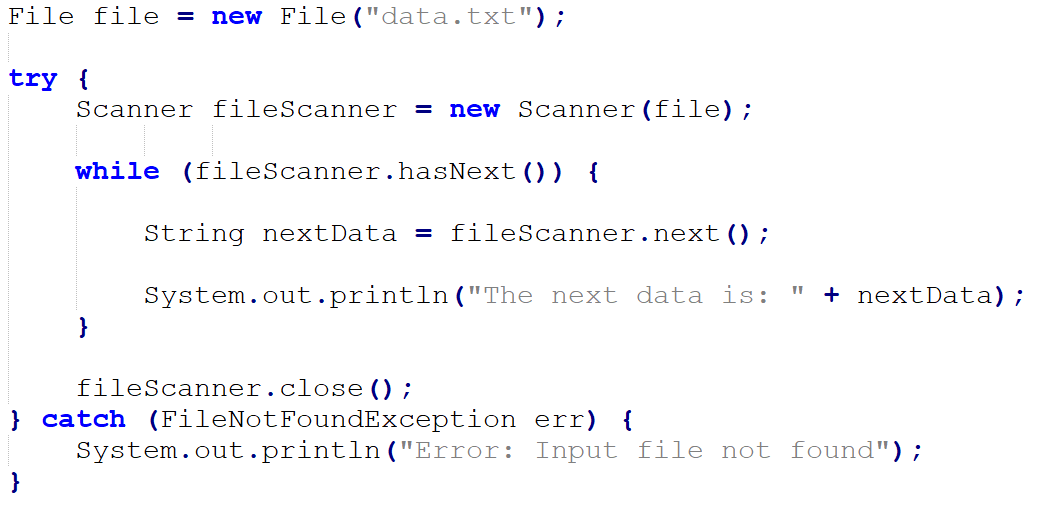
1. Close the file before exiting from the program
2. Write to the file



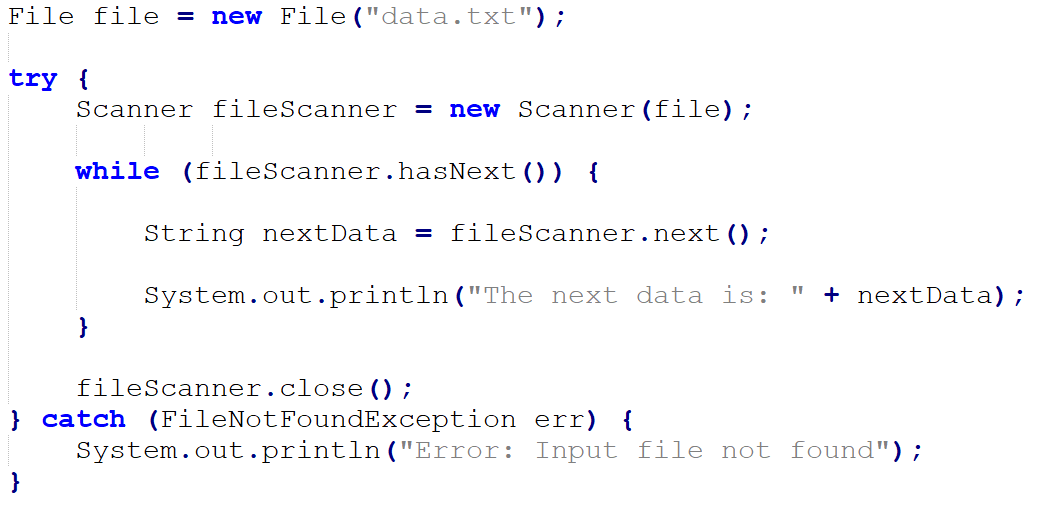
**The catch part is executed if the File cannot be saved to.**

**File IO  
Reading a file using Scanner.**

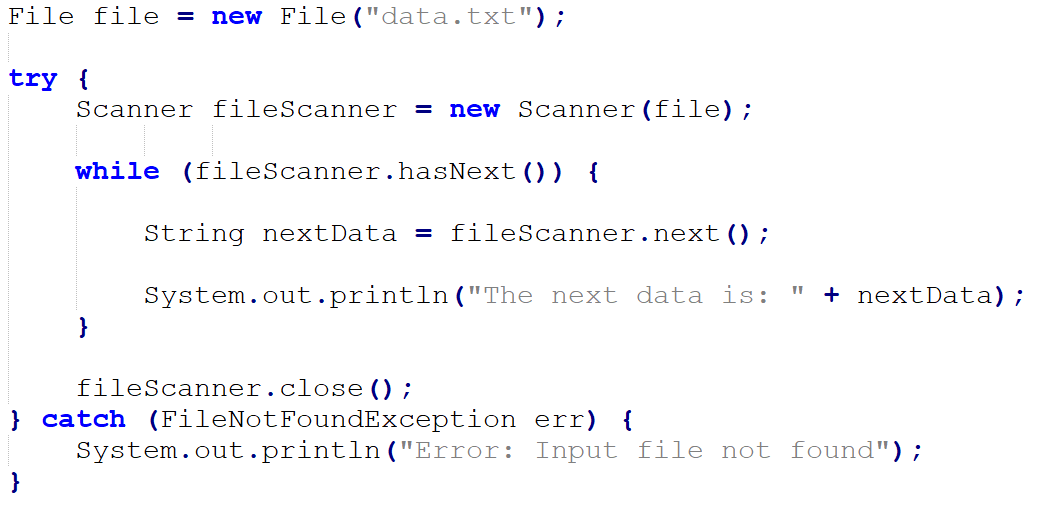
It is similar to reading from the keyboard, except that (i) now data come from a file and (ii) we need to handle possible file exceptions using try…catch



Scanner is defined under java.util  
FileIOException is under java.io

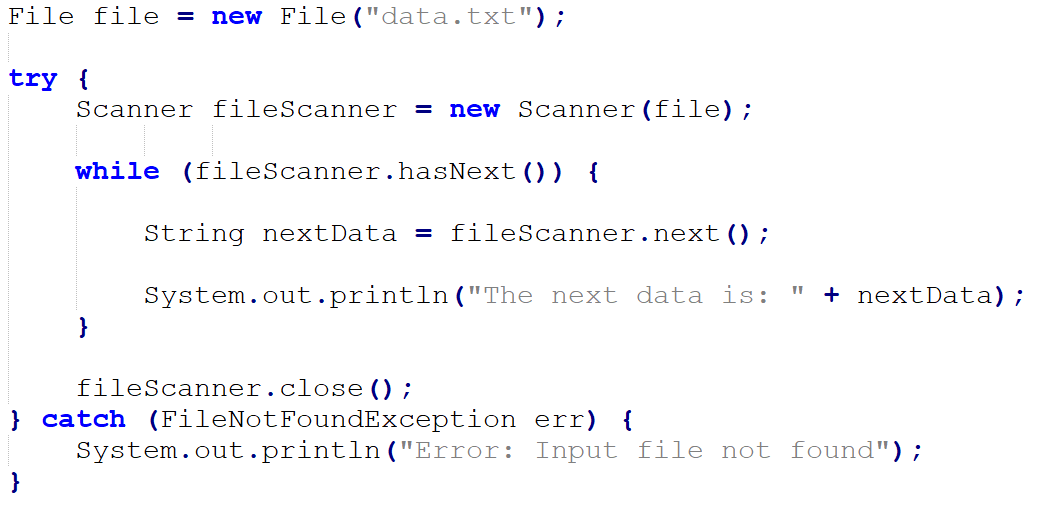
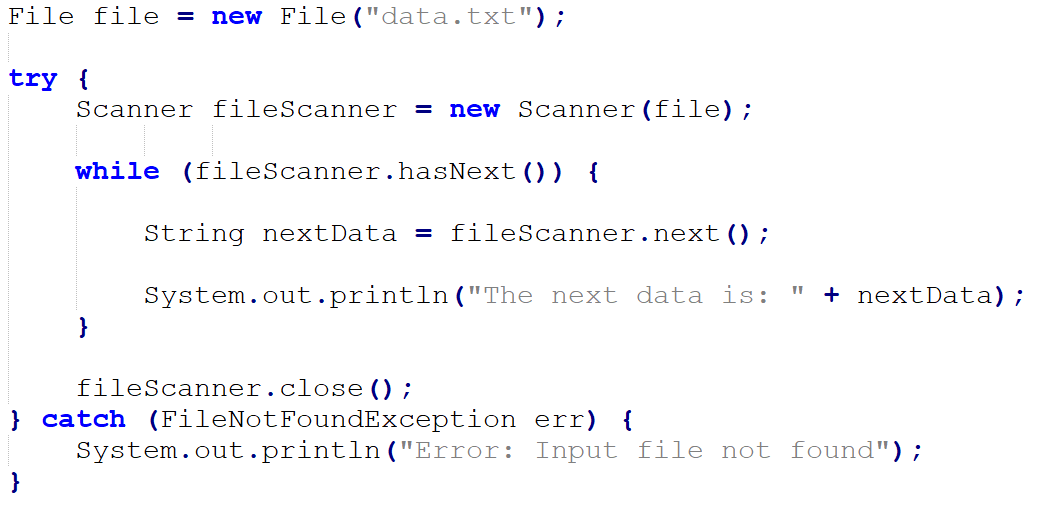


Create Scanner object to read from the file (“data.txt”)



Read the input in the same way that you read from the keyboard (System.in)

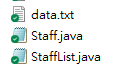
(Other available methods: nextLine(), nextInt() etc)



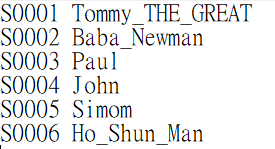
**The catch part is executed if the File cannot be opened for reading.**

**Sample Program: StaffList**

* Demo program: StaffList.java and Staff.java



* This program first load some staff data (id and name) from a file on the computer to the program.



data.txt

(text file on the computer)

* And let the user view or update the data (in the program) by typing some commands.

PRESS 1 to view a record

PRESS 2 to edit a record

PRESS 3 to exit

>2

Enter the Staff ID to be edited:

>S0002

Please enter new name:

>Baba OLDMan

Record updated!

PRESS 1 to view a record

PRESS 2 to edit a record

PRESS 3 to exit

>1

Enter a Staff ID:

>S0001

Staff ID: S0001

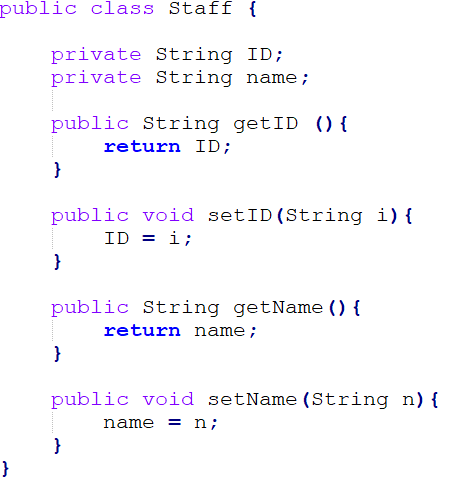
Staff Name: Tommy\_THE\_GREAT

View a record EditView a record

* Save the (updated) data back to the file at the end of the program

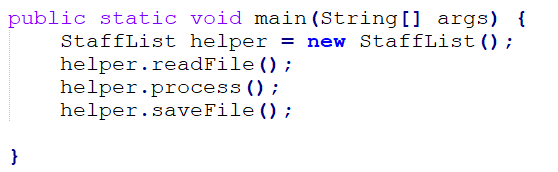
The **Staff** class

This class defines the Staff objects for storing an ID a name of a staff member.



**Methods for accessing the attributes**

The **main** method



* Our main method would not do the works itself!
* Instead, it creates an helper object to help it!
* It then asks the helper to:
  1. Open the data file and **read the file contents** into the program
  2. Start **processing the command**s from the user
  3. **Save the data back to the file** at the end.



Helper! Please read the staff data from the file, handle the commands from the user accordingly, and save the updated data back to the file at the end!

main()



helper  
The **StaffList** class

The StaffList class is like a “helper”, who knows how to do a few things for us.

Methods it can do:

* Read the staff data from a file into the program. (**readFile**)
* **Process** the user’s questions, and modify the Staff data in the program. i.e.,

Ask what the user want to do, and either

Display a staff record (**displayData**), or

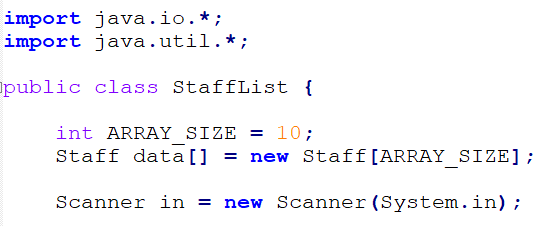
Edit a staff record (**doEdit**)

* Save the updated Staff data from the program back to the file (**saveFile**)

The **StaffList** class (**attributes**)

Required for File I/O

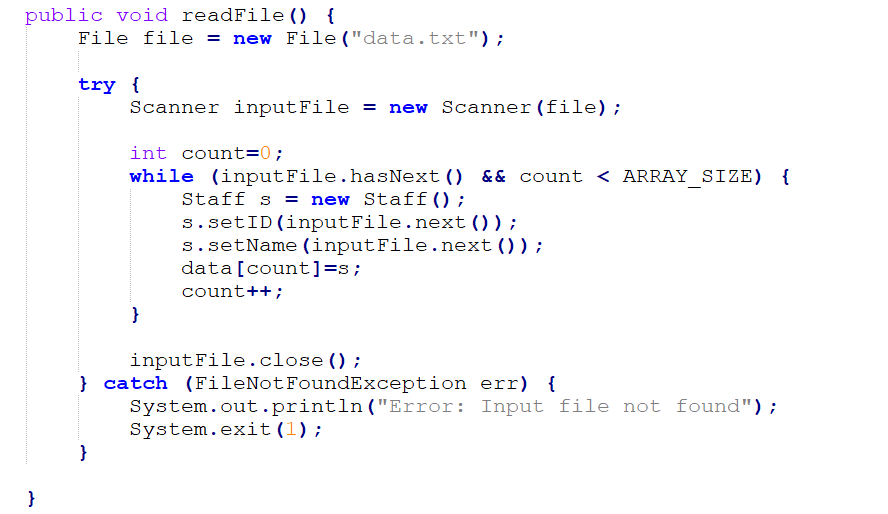
Required for useful utlities, such as Scanners



The **StaffList** class (**readFile()** method)

For reading user input from the keyboard

This is where the data are stored in the program (after reading from the file)



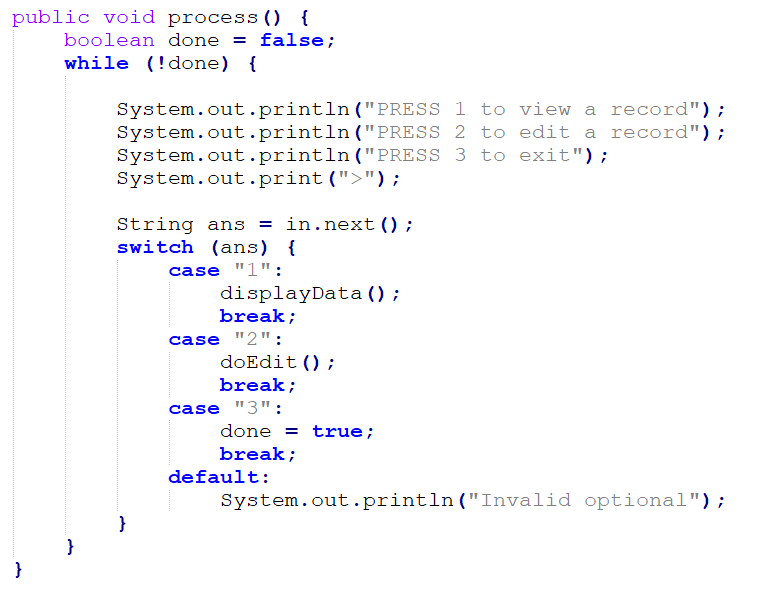
data

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |

s

Create a Staff object for each staff record, and put into the data[] array.

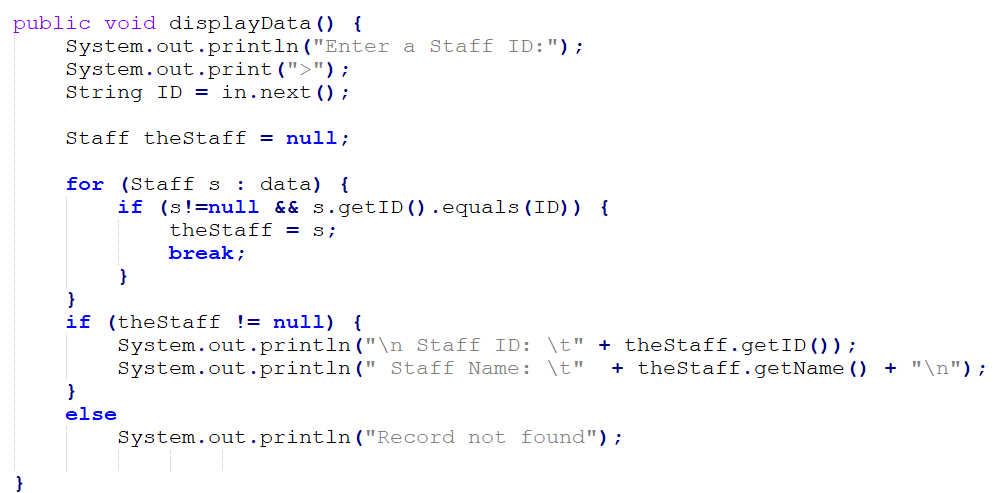
The **StaffList** class (**process()** method)



It will keep doing the following repeatedly unless the user choose to exit:

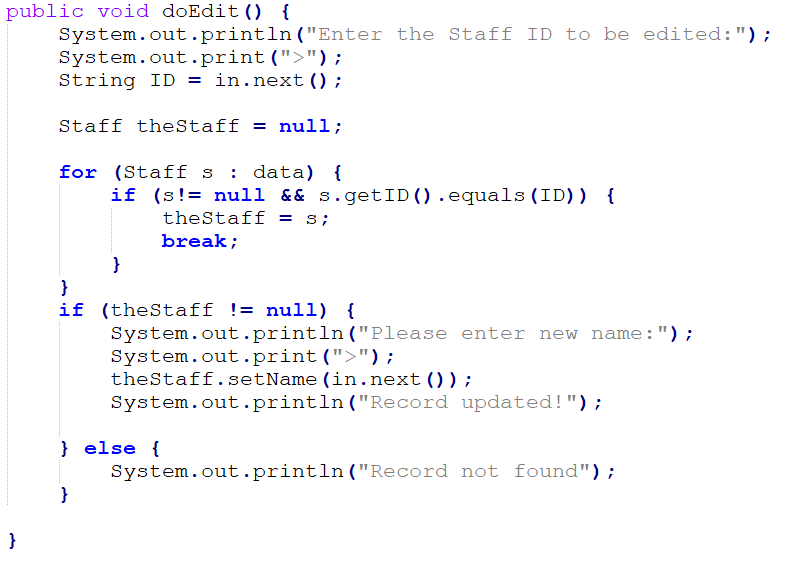
1. Display a menu, and ask user to select an option
2. If the user chooses 1, perform “displayData”
3. If the user chooses 2, perform “doEdit”
4. If the user choosed 3, exit (break) from the loop!

The **StaffList** class (**displayData()** method)



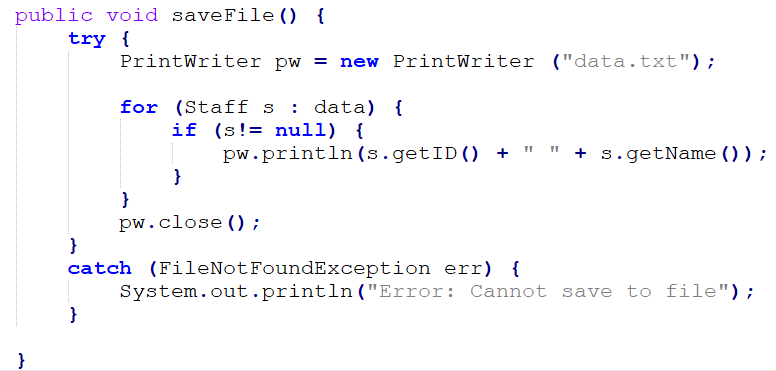
* Ask the user to enter the Staff ID of the person he is looking for.
* Go through each Staff object in the data[] array.
* If a staff ID matches the one entered by the user, it is the object we are looking for. Call it *theStaff*.
* Print out the details of *theStaff*

The **StaffList** class (**doEdit()** method)



* Ask the user to enter the Staff ID of the person he is looking for.
* Go through each Staff object in the data[] array.
* If a staff ID matches the one entered by the user, it is the object we are looking for. Call it *theStaff*.
* Ask the user to enter a new name for *theStaff*

The **StaffList** class (**saveFile()** method)



Note: the main method is NOT part of a StaffList object. In fact, we can place the main method in any class.

**Inheritance**

* Given the following Employee class

**public class** Employee {

String name;

Date hireDate;

Date dateOfBirth;

}

* Now, suppose that we also have a special type of employees called the *managers*.
* A manager has all the attributes of an ordinary employee (name, hireDate, dateOfBirth), plus a few others attributes.

**public class** Manager { // Version 1

String name;

Date hireDate;

Date dateOfBirth;

**String department;**

**Employee subordinates[];**

}

* What is bad with this approach?

**Inheritance**

* A better approach:

**public class** Employee {

String name;

Date hireDate;

Date dateOfBirth;

}

**public class** Manager **extends** Employee {

String department;

Employee subordinates [];

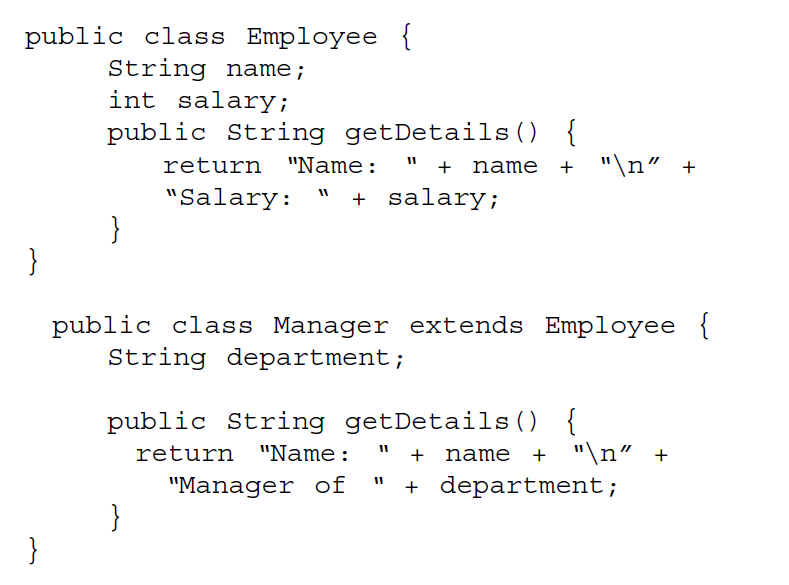
}

* Manager is a *subclass* of Employee. It *inherits* all the attributes and methods of Employee.[[1]](#footnote-1)
* Employee is the *superclass* (or parent) of Manager.

Employee emp = **new** Employee();  
emp.name= “Chow Sun Yuk”;

Manager mgr = **new** Manager();  
mgr.name = “Ng Fong Gong”;  
mgt.department = “H.R.”;

**Overriding Methods in subclass**



Exercise (Inheritance )**The Object Class**

Given the class definitions on the left and the following declarations.

ClassP p = new ClassP();  
ClassC c = new ClassC();

For each of the following statements, if it is valid, state the output.   
If it is not valid (illegal), say so.

p.m1(); Yes 🡺 m1 Parent

c.m1(); Yes 🡺 m1 Parent

p.m2(); Yes 🡺 m2 Parent

c.m2(); Yes 🡺 m2 Child

p.m3(); NOT LEGAL

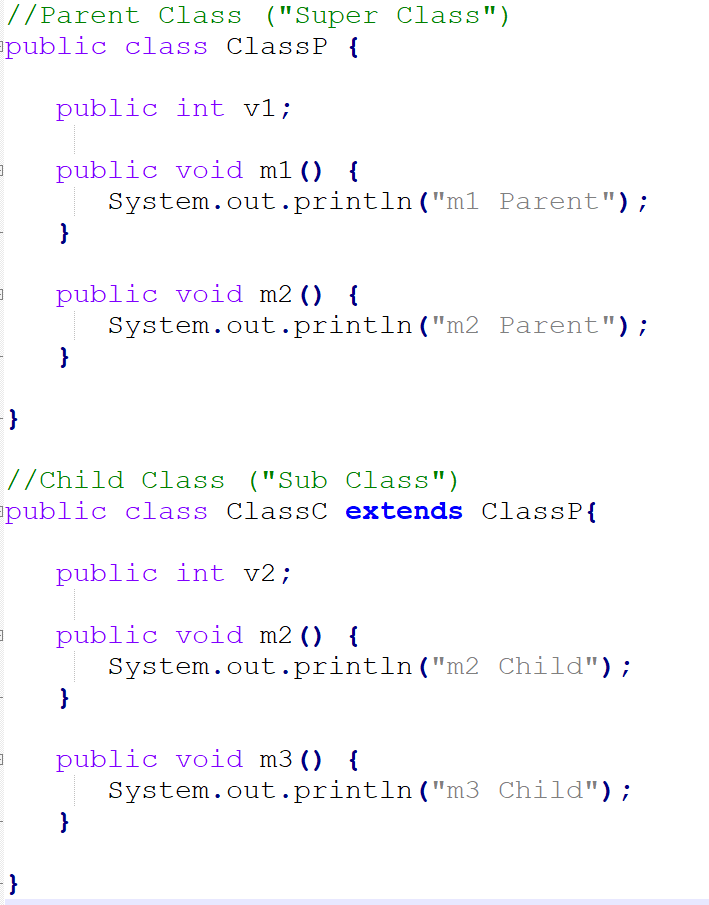
c.m3();

System.out.println(p.v1);

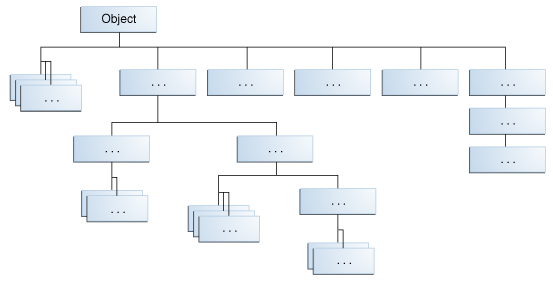
System.out.println(c.v1);

System.out.println(p.v2);

System.out.println(c.v2);



* In Java, any may inherit (extend) exactly one other class (it’s parent).
* And the parent may extend yet another class, and so on.
* The overall picture may look like a tree
* The top-most class is simply called the class Object (the name is confusing, as this is the name of a class)



**Class (static) Variables**

•Shared among all instances of a class

**public class** Count {

**public static int** **counter** = 0;

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

counter++;

Count.counter++;

}

}

**public class** Count {

**public int** **counter** = 0;

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

Count peopleCount = new Count();

peopleCount.counter++;

}

}

**Interface**

* In some cases, we want to show what methods are available in some classes, without showing the actual implemention.

//Interface declaration: by first user

**interface** Drawable{

**void** draw();

}

//An implementation

**class** Rectangle **implements** Drawable{

**public** **void** draw(){System.out.println("drawing rectangle");}

public double getWidth() { /\* some other codes \*/ };

}

//Another implementation

**class** Circle **implements** Drawable{

**public** **void** draw(){System.out.println("drawing circle");}

public double getRadius() { /\* some other codes \*/ };

}

//How they are used:

**class** TestInterface1{

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

Drawale d=**new** Circle();

d.draw();

}  
}

**Interface**

* An **Interface** lists out (some) of the methods that can be used, as well as what instance variables are available.
* More than one class may implement the same Interface (and possibly in different ways). And a class may implement multiple interfaces.

1. Except constructors, which are not inherited. [↑](#footnote-ref-1)