Unit 2: Work With Inheritance and Handling Exceptions
Most Essential Learning Competencies: *At the end of the course, you must be able to:*

1. Discuss and define class inheritance
2. Handle errors and catch exceptions

Reading Activity

What is Inheritance?

Inheritance can be defined as the process where one class acquires the properties ***methodsandfields*** of another. With the use of inheritance the information is made manageable in a hierarchical order.

The class which inherits the properties of other is known as subclass ***derivedclass***, ***childclass*** and the class whose properties are inherited is known as superclass ***baseclass***, ***parentclass*****.**

The extends Keyword

extends is the keyword used to inherit the properties of a class. Below given is the syntax of extends keyword.

```
class Super{
.....
.....
}
class Sub extends Super{
.....
.....
}
```

Java classes are organized in a hierarchical inheritance structure. Subclasses are derived from superclasses. The class Object is the highest-level superclass; it has no parent class above it. All other classes are either subclasses of Object or subclasses of subclasses of Object. This is illustrated in Figure 9-10.1.

Inheritance represents “is a” relationship. For example, a Student is a Person, so the class Student could be a subclass of the Person class. An Employee is a Person, too, so Employee could be another subclass of Person.

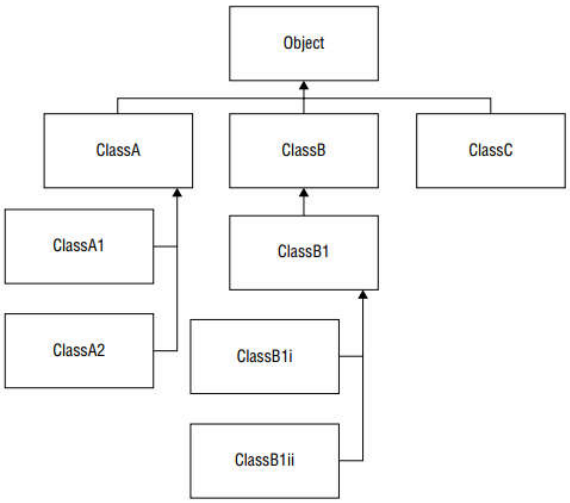

Figure 9 -10.1

To indicate that a class is a subclass of another class, use the extends keyword in the class declaration, for example, public class Employee extends Person{ }. Variables and methods that are shared by all types of Persons can be placed in the Person class, and then each subclass can have specialized variables and methods that are only used by that type or its subclasses. For example,

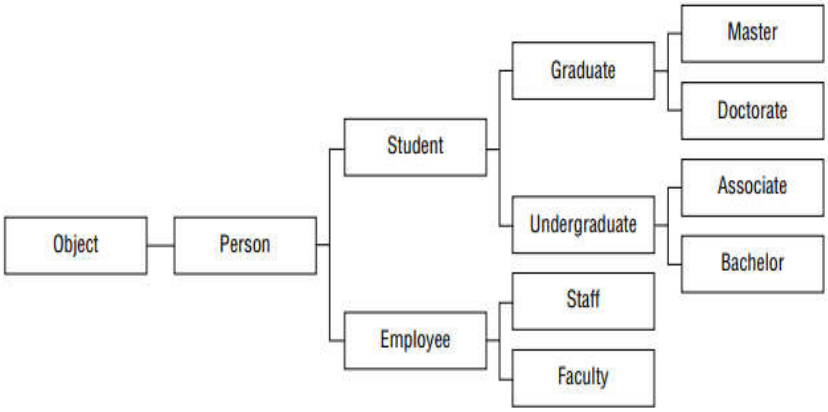

Figure 9 - 10.2

the Person class might have a String variable called name, which all Person objects will inherit whether they are graduate students or faculty employees. The Student class might have a double variable for their grade point average, which both graduate and undergraduate students inherit, but employees do not. A graduate student might have a String variable to store the title of their thesis. This would then be inherited by master and doctorate students. At the lowest level of the hierarchy, a Doctorate student might have a Faculty instance as their advisor, but with no subclasses, this will not be inherited by any other class.

The Super Keyword

Another keyword introduced in this chapter is super. Super is a way of referring to the superclass. It's not so different than the this keyword, covered earlier in this chapter. It is used in constructors to invoke the constructor of the superclass, like the this keyword was used to invoke another constructor of the same class. It is also used to access the variables and methods of the superclass, like the this keyword was used to access the variables of the instance calling the method.

Consider a possible implementation of two of the classes from the Person example: Person and Employee. Employee is the subclass of Person, and Person is the superclass of Employee.

```
public class Person {
        private String name;

        public Person(String name){
                this.setName(name);
        }

        public String getName(){
                return this.name;
        }

        public void setName(String name){
                this.name = name;
        }
}
```

Recognizing Error Types

Errors are almost unavoidable when programming. Just as when you are writing an essay and sometimes make typos or misuse words, you will make occasional mistakes when programming in Java. These mistakes can be referred to as bugs, errors, or exceptions. Here you will find them classified into three main categories: syntax errors, runtime errors, and logical errors. In general, syntax errors are the easiest to find and correct while logical errors are the most difficult.

Identifying Syntax Errors

Syntax errors are the programming equivalent of spelling and grammar mistakes in natural languages. Syntax errors include the following examples:

- Misspelled class, variable, or method names
- Misspelled keywords
- Missing semicolons
- Missing return type for methods

- Out of place or mismatched parentheses and brackets
- Undeclared or uninitialized variables
- Incorrect format of loops, methods, or other structures

In the following code example, see how many syntax errors you can spot:

```
public class errors {

    public static vod main(String[] args) {
        age = 30;
        int retirementFund = 10000;
        int yearsInRetirement = 0;
        String name = "David Johnson",
        for (int i = age; <= 65; ++){
            recalculate(retirementFund,0.1);
        }
        int monthlyPension = retirementFund/yearsInRetirement/12
        System.out.printline(name + " will have $" + monthlyPension
                + " per month for retirement."];
    }

    public static recalculate(fundAmount, rate){
        fundAmount = fundAmount*(1+rate);
    }
  }
}
```

You probably can spot several just by reading the code. The keyword void is misspelled, the declaration of the string name should end with a semicolon instead of a comma, and the print statement should close with a parenthesis followed by a semicolon.

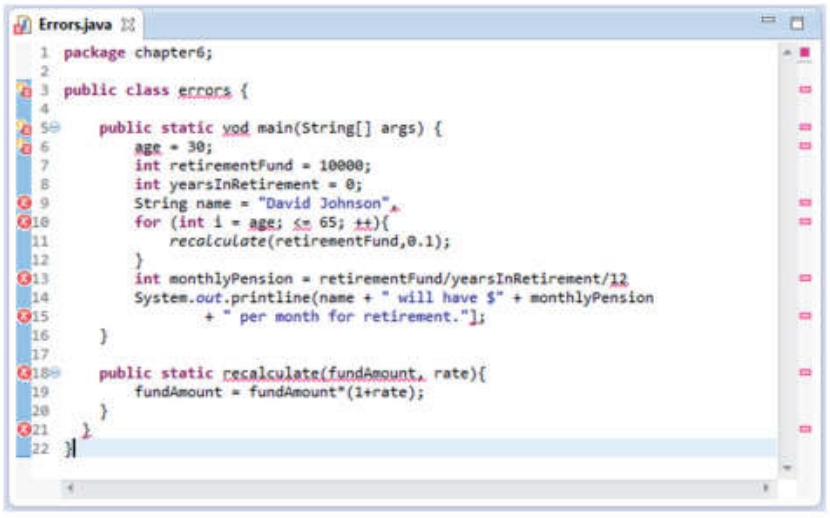

Figure 9 - 10.3 Codes written in Eclipse IDE showing line errors

Following this approach, you can resolve all of the syntax errors in this code.

- Line 3: Rename type to Errors (class name must match .java filename)
- Line 5: Change to void (main method return type is always void)
- Line 9: Change the comma to a semicolon
- Line 10: Change the for loop to for (int i = age; i <= 65; i++)
- Line 18: Set the method return type to void (recalculate method doesn't return any value)
- Line 18: Add the parameter data types to double fundAmount, double rate (this also solves the bug indicated at line 10)
- Line 13: Add a semicolon at the end
- Line 14: Change to println (printline is not correct syntax)
- Line 15: Change the square bracket to a parenthesis
- Line 22: Delete the last closing curly bracket (there was one too many)

After all of these corrections, the syntax errors are resolved and your program can be executed. The resulting code looks like this:

```
public class Errors {

    public static void main(String[] args) {
        int age = 30;
        int retirementFund = 10000;
        int yearsInRetirement = 0;
        String name = "David Johnson";
        for (int i = age; i <= 65; i++){
            recalculate(retirementFund,0.1);
        }
        int monthlyPension = retirementFund/yearsInRetirement/12;
        System.out.println(name + " will have $" + monthlyPension
                + " per month for retirement.");
    }

    public static void recalculate(double fundAmount, double rate){
        fundAmount = fundAmount*(1+rate);
    }
}
```

Identifying Runtime Errors

If you tried running the program after fixing all the syntax errors, you probably already found your first runtime error.

Division by zero is a classic runtime error example, because the syntax is correct and it is only during runtime that a problem occurs. Other typical examples that you will no doubt encounter, if you haven't already, include null pointer exception, index out of bounds exception, and file not found exception.

There are clues in the error message that can help you sort out where and why it happened. First note the type of exception: **java.lang.ArithmeticException**. Already from the name, you can see it must be something related to the calculations. Then, specifically it states "/ by zero," so somewhere your

program has tried to divide by zero, an operation that's undefined and therefore cannot be computed by Java.

Identifying Logical Errors

If you've run the code that you just fixed, did you notice anything strange? There shouldn't be any syntax or runtime errors, so your program should run normally. You should see David Johnson will have $41 per month for retirement. printed to the console. At first glance, this seems fine, but on further inspection, the number is lower than you would expect. In fact, 10,000 divided by 20 years divided by 12 months is 41 dollars per month. But the program is supposed to be calculating an annual return rate of 10% for all the years between age 30 and age 65. So what's going on here?

This is an example of a logical error. The code compiles and executes without error, but logically produces the wrong result. Logical errors are perhaps the most difficult to spot, because there's nothing to suggest there's an error unless you know what to expect and compare the actual results to the expected results. If you are not careful to review what is happening, you may easily miss logical errors lurking in your code.

Exceptions

Exceptions are more specific to programming as they are events that disrupt the execution of a program. Exceptions can be indications that something went wrong, and they can happen automatically as a result of something Java is unable to complete or can be explicitly thrown when certain conditions are met. In the following sections, you'll be introduced to some common exceptions, and you'll get to see how to handle them in your own programs.

There are some common exceptions that occur often enough, not just with new programmers, but even with experienced developers. In this short section, you'll get to see two in particular that come up all the time: **null pointer exceptions** and **index out of bounds exceptions**.

1. **Null Pointer Exceptions** indicate that the program is trying to access an object that doesn't exist yet. Before you can reference a primitive data type, like int, you need to initialize it, which is setting the value.
2. **Index Out Of Bounds** occurs when you have an indexed object, such as an array, and you try to access an element outside the limits of the array.

Two other common exceptions you might encounter are **StackOverFlowError** and **OutOfMemoryError**. These occur when the program you are running demands more memory than your machine allows for Java or your IDE. The stack is the part of your memory allocated for parameters and local variables. This can overflow when you are calling a method recursively or when two methods call each other. The heap is where objects are allocated in memory. Creating too many objects, often within an infinite loop, can quickly consume all the available memory.

Catching Exceptions

Now that you've been introduced to the three main error categories and some common exceptions, it's time to start learning how to handle them when you do encounter them. The first step is a new structure called a **try/catch block**. This essentially allows you to *try* executing a piece of code to see if an exception is thrown. If none is thrown, the program will proceed normally, but if one is thrown, you can *catch* it and specifically indicate what should be done next. This prevents your program from crashing and at least allows you to recover some information before it terminates.

The general form of a try/catch block looks like this:

```
try {
// execute some statements
} catch (Exception exc){
// statements to handle the exception
} finally {
// no matter what, do this
}
```

Read Additional Resources

1. **Inheritance**
   Week9-10_Inheritance.pdf
2. **Handling Exception**
   Week9-10_Handling Exceptions.pdf

Watch Video Resources

1. **Inheritance**
   Week9-10_Inheritance.mp4
2. **Handling Exceptions**
   Week9-10_Handling Exception.mp4
3. **Types of Common Errors in Java Programming**
   Week9-10_Common Errors in Java Programming.mp4
4. **Errors – Intro to Java Programming**
   Week9-10_Errors Intro to Java Programming.mp4

Laboratory Activity

Laboratory Activity 9 -10.1

Java Class Inheritance

1. Open Netbeans (or Eclipse) IDE
2. Copy and paste the program given below in a file with name **My_Calculation.java**

```
class Calculation{
        int z;
        public void addition(int x, int y){
                z=x+y;
        System.out.println("The sum of the given numbers:"+z);
        }

        public void Substraction(int x,int y){
                z=x-y;
        System.out.println("The difference between the given numbers:"+z);
```

```
        }
}

public class My_Calculation extends Calculation{
        public void multiplication(int x, int y){
                z=x*y;
                System.out.println("The product of the given numbers:"+z);
        }

        public static void main(String args[]){
                int a=20, b=10;
                My_Calculation demo = new My_Calculation();
                demo.addition(a, b);
                demo.Substraction(a, b);
                demo.multiplication(a, b);
        }
}
```

3. Compile and execute the above code as shown below

```
javac My_Calculation.java
java My_Calculation
```

After executing the program it will produce the following result.

```
The sum of the given numbers:30
The difference between the given numbers:10
The product of the given numbers:200
```

Laboratory Activity 9 - 10.2

The super Keyword

1. Open Netbeans (or Eclipse) IDE
2. Copy and paste the program in a file with name **Sub_class.java**

```
class Super_class{
        int num=20;
        //display method of superclass
        public void display(){
                System.out.println("This is the display method of superclass");
        }
}
public class Sub_class extends Super_class {
        int num=10;
```

```
        //display method of sub class

        public void display(){
                System.out.println("This is the display method of subclass");
        }
        public void my_method(){
                //Instantiating subclass
                Sub_class sub=new Sub_class();
                //Invoking the display() method of sub class
                sub.display();
                //Invoking the display() method of superclass
                super.display();
                //printing the value of variable num of subclass
                System.out.println("value of the variable named num in sub class:"+ sub.num);
                //printing the value of variable num of superclass
                System.out.println("value of the variable named num in super class:"+ super.num);
        }
        public static void main(String args[]){
                Sub_class obj = new Sub_class();
                obj.my_method();
        }
}
```

3. Compile and execute the above code using the following syntax.

```
javac Super_Demo
java Super
```

On executing the program you will get the following result:

```
This is the display method of subclass
This is the display method of superclass
value of the variable named num in sub class:10
value of the variable named num in super class:20
```

Laboratory Activity 9 - 10.3

Invoking Superclass Constructor

1. Open Netbeans (or Eclipse) IDE
2. Copy and paste the below given program in a file with name Subclass.java

```
class Superclass{
        int age;
```

```
        Superclass(int age){
                this.age=age;
        }

         public void getAge(){
                System.out.println("The value of the variable named age in super class is: " +age);
        }
}

public class Subclass extends Superclass {
        Subclass(int age){
                super(age);
        }

        public static void main(String argd[]){
                Subclass s= new Subclass(24);
                s.getAge();
        }
}
```

3. Compile and execute the above code using the following syntax.

```
javac Subclass
java Subclass
```

On executing the program you will get the following result:

```
The value of the variable named age in super class is: 24
```

Self-Check

Quiz 9-10.1

Instructions: Write your answer on the Answer Sheet (AS) provided in this module.

1. Enumerate and define the different types of errors and exception (2-point each).
2. Write down the syntax of extends keywords (5-points).
3. Write down the syntax of try-catch methods (5-points).

Internet References

1. https://www.youtube.com
2. https://www.tutorialspoint.com
3. https://www.youtube.com/watch?v=RI_6AYD8RZ
4. https://www.youtube.com/watch?v=LrrqSbVP4bo