UNIT-II
Interface

A programmer uses an abstract class when there are some common features shared by all the objects A programmer writes an interface when all the features have different implementations for different objects Interfaces are written when the programmer wants to leave the implementation to third party vendors An interface is a specification of method prototypes All the methods in an interface are abstract methods

- An interface is a specification of method prototypes
- An interface contains zero or more abstract methods
- All the methods of interface are public, abstract by default
- An interface may contain variables which are by default public static final
- Once an interface is written any third party vendor can implement it
- All the methods of the interface should be implemented in its implementation classes
- If any one of the method is not implemented, then that implementation class should be declared as abstract
- We cannot create an object to an interface
- We can create a reference variable to an interface
- An interface cannot implement another interface
- An interface can extend another interface
- A class can implement multiple interfaces

Program 1: Write an example program for

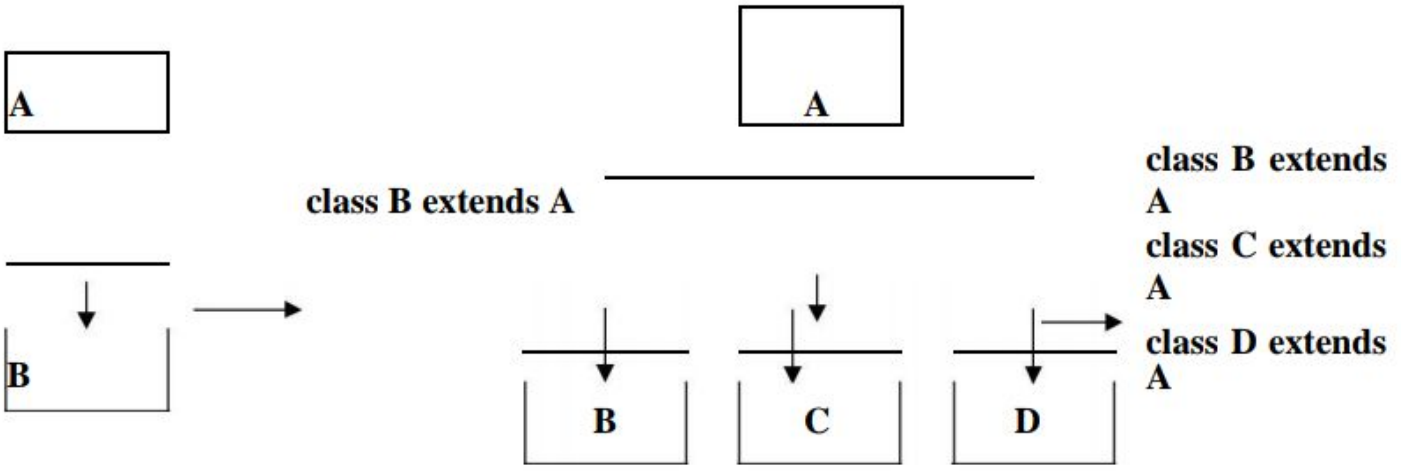
```
interface interface Shape
{void  area  ();  void
        volume ();
        double pi = 314;
}
class Circle implements Shape
{double r;
        Circle (double radius)
        {r = radius;
        }
        public void area ()
        {Systemoutprintln ("Area of a circle is : " + pi*r*r );
        }
        public void volume ()
        {Systemoutprintln ("Volume of  a circle is : " + 2*pi*r);
        }
}
class Rectangle implements Shape
{double l,b;
        Rectangle (double length, double breadth)
        {l  =  length;  b  =
                breadth;.}
          public void area ()
          {Systemoutprintln ("Area of a Rectangle is : " + l*b );
          }
```

```
        public void volume ()
        {Systemoutprintln ("Volume of  a Rectangle is : " + 2*(l+b));
        }
}
class InterfaceDemo
{public static void main (String args[])
        {Circle ob1 = new Circle (102);
                ob1area ();
                ob1volume ();
                Rectangle ob2 = new Rectangle (126,
                2355); ob2area ();
                ob2volume ();
        }
}
```

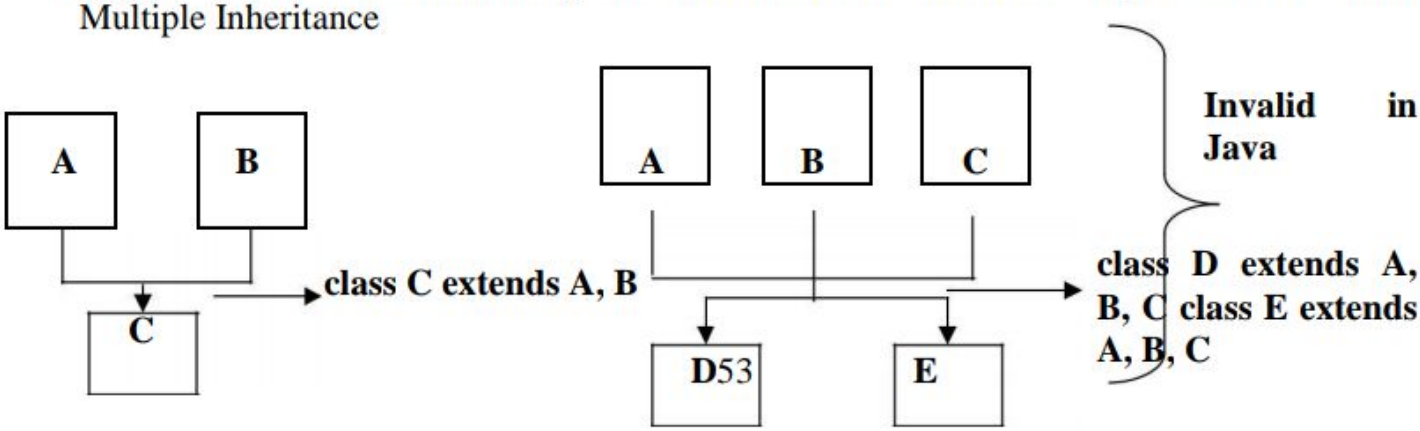
Output:

Types of inheritance:

Single Inheritance: Producing subclass from a single super class is called single inheritance



Multiple Inheritance: Producing subclass from more than one super class is called Multiple Inheritance



Java does not support multiple inheritance But multiple inheritance can be achieved by using interfaces

Program 2: Write a program to illustrate how to achieve multiple inheritance using multiple interfaces

```
//interface Demo
interface Father
{double   PROPERTY   =   10000;
        double HEIGHT = 56;
}
interface Mother
{double   PROPERTY   =   30000;
        double HEIGHT = 54;
}
class MyClass implements Father, Mother
{void show()
        { Systemoutprintln("Total property is :" +(FatherPROPERTY+MotherPROPERTY));
          Systemoutprintln ("Average height is :" + (FatherHEIGHT + MotherHEIGHT)/2 );
        }
}
class InterfaceDemo
{public static void main(String args[])
        {MyClass   ob1   =   new   MyClass();
                ob1show();
        }
}
```

Output:

Packages

A package is a container of classes and interfaces A package represents a directory that contains related group of classes and interfaces For example, when we write statemens like:

import javaio*;

Here we are importing classes of javaio package Here, java is a directory name and io is another sub directory within it The '*' represents all the classes and interfaces of that io sub directory We can create our own packages called user-defined packages or extend the available packages User-defined packages can also be imported into other classes and used exactly in the same way as the Built-in packages Packages provide reusability

General form for creating a package:

package packagename;

eg: package pack;

The first statement in the program must be package statement while creating a package

While creating a package except instance variables, declare all the members and the class itself as public then only the public members are available outside the package to other programs

Program 1: Write a program to create a package pack with Addition class

```
//creating a package
package pack;
public class Addition
{private double d1,d2;
        public Addition(double a,double b)
        {d1 = a; d2 =
                b;
        }
        public void sum()
        {Systemoutprintln ("Sum of two given numbers is : " + (d1+d2) );
        }
}
```

Compiling the above program:

The –d option tells the Java compiler to create a separate directory and place the class file in that directory (package) The () dot after –d indicates that the package should be created in the current directory So, out package pack with Addition class is ready

Program 2: Write a program to use the Addition class of package pack

```
//Using the package pack
import packAddition;
class Use

{public static void main(String args[])
        {Addition ob1 = new Addition(10,20);
                ob1sum();
        }
```

}

Output:

Program 3: Write a program to add one more class Subtraction to the same package pack

```
//Adding one more class to package pack:
package pack;
public class Subtraction
{private double d1,d2;
        public Subtraction(double a, double b)
        {d1 = a; d2 =
                b;
        }
        public void difference()
        {Systemoutprintln ("Sum of two given numbers is : " + (d1 - d2) );
        }
}
```

Compiling the above program:

Program 4: Write a program to access all the classes in the package pack

```
//To import all the classes and interfaces in a class using import pack*;
import  pack*;
class Use
{public static void main(String args[])
        {Addition   ob1   =   new   Addition(105,206);
                ob1sum();
                Subtraction          ob2          =          new
                Subtraction(302,4011); ob2difference();
        }
}
```

In this case, please be sure that any of the Additionjava and Subtractionjava programs will not exist in the current directory Delete them from the current directory as they cause confusion for the Java compiler The compiler looks for byte code in Additionjava and Subtractionjava files and there it gets no byte code and hence it flags some errors

Output:

If the package pack is available in different directory, in that case the compiler should be given information regarding the package location by mentioning the directory name of the package in the classpath The CLASSPATH is an environment variable that tells the Java compiler where to look for class files to import If our package exists in e:\sub then we need to set class path as follows:

We are setting the classpath to **e:\sub** directory and current directory () and %CLASSPATH% means retain the already available classpath as it is

Creating Sub package in a package: We can create sub package in a package in the format:

package packagenamesubpackagename;

eg: package pack1pack2;

Here, we are creating pack2 subpackage which is created inside pack1 package To use the classes and interfaces of pack2, we can write import statement as:

import pack1pack2;

Program 5: Program to show how to create a subpackage in a package

```
//Creating a subpackage in a package
package pack1pack2;
public class Sample
{public void show ()
        {
                Systemoutprintln ("Hello Java Learners");
        }
}
```

Compiling the above program:

public members of the class are available anywhere The scope of public members of the class is "GLOBAL SCOPE"

default members of the class are available with in the class, outside the class and in its sub class of same package It is not available outside the package So the scope of default members of the class is "PACKAGE SCOPE"

protected members of the class are available with in the class, outside the class and in its sub class of same package and also available to subclasses in different package also

Class Member Access	private	No Modifier	protected	public
Same class	Yes	Yes	Yes	Yes
Same package subclass	No	Yes	Yes	Yes
Same package non-subclass	No	Yes	Yes	Yes
Different package subclass	No	No	Yes	Yes
Different package non-subclass	No	No	No	Yes

Program 6: Write a program to create class A with different access specifiers

```
//create a package same
package same;
public class A
{private     int     a=1;
        public  int  b  =
        2;
        protected int c = 3;
        int d = 4;
}
```

Compiling the above program:

Program 7: Write a program for creating class B in the same package

```
//class B of same package
package same;
import sameA;
public class B
{
        public static void main(String args[])
        {
                A obj = new A();
                Systemoutprintln(obja);
                Systemoutprintln(objb);
                Systemoutprintln(objc);
                Systemoutprintln(objd);
        }
}
```

Exception Handling

The **exception handling in java** is one of the powerful *mechanism to handle the runtime errors* so that normal flow of the application can be maintained.

What is exception

In java, exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime.

Advantage of Exception Handling

The core advantage of exception handling is **to maintain the normal flow of the application**. Exception normally disrupts the normal flow of the application that is why we use exception handling.

Types of Exception

There are mainly two types of exceptions: checked and unchecked where error is considered as unchecked exception. The sun microsystem says there are three types of exceptions:

1. Checked Exception
2. Unchecked Exception
3. Error

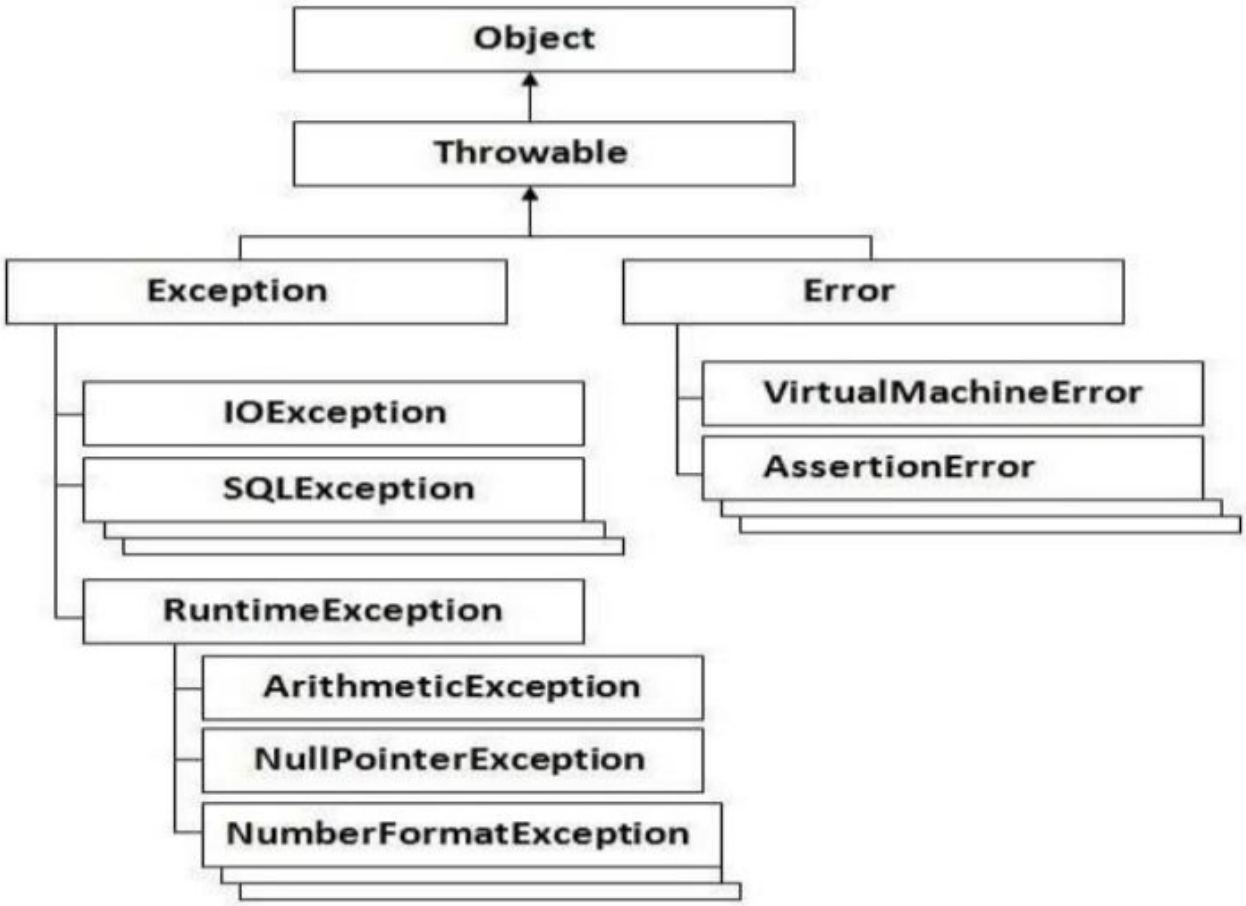
Difference between checked and unchecked exceptions

1) Checked Exception: The classes that extend Throwable class except RuntimeException and Error are known as checked exceptions e.g.IOException, SQLException etc. Checked exceptions are checked at compile-time.

2) Unchecked Exception: The classes that extend RuntimeException are known as unchecked exceptions e.g. ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc. Unchecked exceptions are not checked at compile-time rather they are checked at runtime.

3) Error: Error is irrecoverable e.g. OutOfMemoryError, VirtualMachineError, AssertionError etc.

Hierarchy of Java Exception classes



Checked and UnChecked Exceptions

Checked Exceptions	Unchecked Exceptions
• Exception which are checked at Compile time called Checked Exception • If a method throws a checked exception, then the method must either handle the exception or it must specify the exception using throws keyword	• Exceptions whose handling is NOT verified during Compile time. • These exceptions are handled at run-time i.e., by JVM after they occurred by using the try and catch block
• Examples: o IOException o SQLException o DataAccessException o ClassNotFoundException o InvocationTargetException o MalformedURLException	• Examples o NullPointerException o ArrayIndexOutOfBound o IllegalArgumentException o IllegalStateException

Java try block

Java try block is used to enclose the code that might throw an exception. It must be used within the method.

Java try block must be followed by either catch or finally block.

Syntax of java try-catch

```
1. try{
2. //code that may throw exception
3. }catch(Exception_class_Name ref){}
```

Syntax of try-finally block

```
1. try{
2. //code that may throw exception
3. }finally{}
```

Java catch block

Java catch block is used to handle the Exception. It must be used after the try block only.

You can use multiple catch block with a single try.

Problem without exception handling

Let's try to understand the problem if we don't use try-catch block.

```
public class Testtrycatch1{
  public static void main(String args[]){
    int data=50/0;//may throw exception
    System.out.println("rest of the code...");
} }
```

Output:

```
Exception in thread main java.lang.ArithmeticException:/ by zero
```

As displayed in the above example, rest of the code is not executed (in such case, rest of the code... statement is not printed).

There can be 100 lines of code after exception. So all the code after exception will not be executed.

Solution by exception handling

Let's see the solution of above problem by java try-catch block.

```
public class Testtrycatch2{
```

```
 public static void main(String args[]){
  try{
    int data=50/0;
  }catch(ArithmeticException e){System.out.println(e);}
  System.out.println("rest of the code...");
} }
```

1. Output:

```
Exception in thread main java.lang.ArithmeticException:/ by zero
rest of the code...
```

Now, as displayed in the above example, rest of the code is executed i.e. rest of the code... statement is printed.

Java Multi catch block

If you have to perform different tasks at the occurrence of different Exceptions, use java multi catch block.

Let's see a simple example of java multi-catch block.

```
1.  public class TestMultipleCatchBlock{
2.    public static void main(String args[]){
3.     try{
4.      int a[]=new int[5];
5.      a[5]=30/0;
6.     }
7.    catch(ArithmeticException e){System.out.println("task1 is completed");}
8.   catch(ArrayIndexOutOfBoundsException e){System.out.println("task 2 completed");
9. }
10.    catch(Exception e){System.out.println("common task completed");
11. }
12.    System.out.println("rest of the code...");
13. } }
```

```
Output:task1 completed
      rest of the code...
```

Java nested try example

Let's see a simple example of java nested try block.

```
class Excep6{
 public static void main(String args[]){
  try{
    try{
     System.out.println("going to divide");
     int b =39/0;
    }catch(ArithmeticException e){System.out.println(e);}

    try{
```

```
   int a[]=new int[5];
   a[5]=4;
   }catch(ArrayIndexOutOfBoundsException e){System.out.println(e);}
   System.out.println("other statement);
  }catch(Exception e){System.out.println("handeled");}
  System.out.println("normal flow..");
 }
1. }
```

Java finally block

Java finally block is a block that is used *to execute important code* such as closing connection, stream etc.

Java finally block is always executed whether exception is handled or not.

Java finally block follows try or catch block.

Usage of Java finally

Case 1

Let's see the java finally example where **exception doesn't occur**.

```
class TestFinallyBlock{
  public static void main(String args[]){
  try{
   int data=25/5;
   System.out.println(data);
  }
  catch(NullPointerException e){System.out.println(e);}
  finally{System.out.println("finally block is always executed");}
  System.out.println("rest of the code...");
  }
}
```

Output:5
finally block is always executed
rest of the code...

Java throw keyword

The Java throw keyword is used to explicitly throw an exception.

We can throw either checked or uncheked exception in java by throw keyword. The throw keyword is mainly used to throw custom exception. We will see custom exceptions later.

The syntax of java throw keyword is given below.

```
1. throw exception;
```

Java throw keyword example

In this example, we have created the validate method that takes integer value as a parameter. If the age is less than 18, we are throwing the ArithmeticException otherwise print a message welcome to vote.

```
1. public class TestThrow1{
     static void validate(int age){
     if(age<18)
     throw new ArithmeticException("not valid");
     else
     System.out.println("welcome to vote");
     }
     public static void main(String args[]){
     validate(13);
     System.out.println("rest of the code...");
     } }
```

Output:

```
Exception in thread main java.lang.ArithmeticException:not valid
```

Java throws keyword

The **Java throws keyword** is used to declare an exception. It gives an information to the programmer that there may occur an exception so it is better for the programmer to provide the exception handling code so that normal flow can be maintained.

Exception Handling is mainly used to handle the checked exceptions. If there occurs any unchecked exception such as NullPointerException, it is programmers fault that he is not performing check up before the code being used.

Syntax of java throws

```
1. return_type method_name() throws exception_class_name{
2. //method code
3. }
4.
```

Java throws example

Let's see the example of java throws clause which describes that checked exceptions can be propagated by throws keyword.

```
import java.io.IOException;
class Testthrows1{
  void m()throws IOException{
    throw new IOException("device error");//checked exception
```

```
}
void n()throws IOException{
 m();
}
void p(){
 try{
  n();
 }catch(Exception e){System.out.println("exception handled");}
}
public static void main(String args[]){
 Testthrows1 obj=new Testthrows1();
 obj.p();
 System.out.println("normal flow..."); } }
```

Output:

```
exception handled
normal flow...
```

Java Custom Exception

If you are creating your own Exception that is known as custom exception or user-defined exception. Java custom exceptions are used to customize the exception according to user need.

By the help of custom exception, you can have your own exception and message.

Let's see a simple example of java custom exception.

```
class InvalidAgeException extends Exception{
 InvalidAgeException(String s){
  super(s);
 } }
class TestCustomException1{
  static void validate(int age)throws InvalidAgeException{
    if(age<18)
     throw new InvalidAgeException("not valid");
    else
     System.out.println("welcome to vote");
  }
  public static void main(String args[]){
    try{
    validate(13);
    }catch(Exception m){System.out.println("Exception occured: "+m);}

    System.out.println("rest of the code...");
  } }
```

Output:Exception occured: InvalidAgeException:not valid rest of the code...