Module 3:

Class, Inheritance, Exception, Packages and Interfaces

CLASSES

A class defines structure and behavior (data and code) that will be shared by a set of objects. We can say it defines a new data type which can be used to create objects of that type. Class is a template for an object while an object is an instance of class.

A class in java can contain:

- **data member**
- **method**
- **constructor**
- **block**
- **class and interface**

General form/Declaring class:

```
class  c_name
{
type var1;
type var2;

type method1 (parameter_list)
{
//body of method
}
public static void main(String ar[])
{

}
}
```

Member Access

Accessing data/code:

The data or methods of a class can be accessed from another class only using its objects. Using the objects, its own data/code are accessed using dot(.) operator. Example:

r.length=10; r.area();

Constructors

- A constructor is a method with the same name as that of the class, which is invoked automatically during the creation of an object. It does not have a return type and but can have access specifiers.
- There are basically two rules defined for the constructor.
    1. Constructor name must be same as its class name
    2. Constructor must have no explicit return type
- **Advantages of constructors:**
    1. A constructor eliminates placing the default values.
    2. A constructor eliminates calling the normal or ordinary method implicitly.
- **Types of java constructors**

  There are two types of constructors:
    1. Default constructor (no-arg constructor)
    2. Parameterized constructor.

Default constructor (no-arg constructor):

A constructor is said to be default constructor if and only if it never take any parameters.

If any class does not contain at least one user defined constructor than the system will create a default constructor at the time of compilation it is known as **system defined default constructor**.

Syntax

```
class className
{
className ()
{
Block of statements;  // Initialization
}
.....
```

```
}
```

Example:

```
class A
{
        A()
        {
            int a=10,b=20;
            System.out.println("I am from default Constructor...");
            System.out.println("Value of a: "+a);
            System.out.println("Value of b: "+b);
        }
        public static void main(String ar[])
        {
                A a1=new A();
        }
}
```

Parameterized Constructor:

If any constructor contain list of variable in its signature is known as paremetrized constructor. A parameterized constructor is one which takes some parameters.

Types of parameter passing to constructor

1. Variable as parameters
2. Object as parameters

Syntax

```
class ClassName
{
ClassName(list of parameters) //parameterized constructor
{
.....
}
```

.....

}

Example:

```
class A
{
        int a=10,b=20;
        A(int a,int b)    // variable as a parameter
        {
                System.out.println("Parameter as variable to Constructor...");
                System.out.println("Value of a: "+a);  // Value of a:2
                System.out.println("Value of b: "+b); // Value of b:3
        }

        public static void main(String ar[])
        {
                A a1=new A(2,3);
                A a2=new A(a1);
        }
}
```

Rules or properties of a Constructor

- Constructor will be called automatically when the object is created.
- Constructor name must be similar to name of the class.
- Constructor should not return any value even void also. Because basic aim is to place the value in the object. (if we write the return type for the constructor then that constructor will be treated as ordinary method).
- Constructor definitions should not be static. Because constructors will be called each and every time, whenever an object is creating.
- Constructor should not be private provided an object of one class is created in another class (Constructor can be private provided an object of one class created in the same class).

- Constructors will not be inherited from one class to another class (Because every class constructor is create for initializing its own data members).
- The access specifier of the constructor may or may not be private.

Constructor Overloading:

- **Constructor overloading** is a technique in Java in which a class can have any number of constructors that differ in parameter lists. The compiler differentiates these constructors by taking the number of parameters, and their type.
- Example:

```
class A
{
        int a=10,b=20;
        A(int a,int b)
        {
          System.out.println("Value of a: "+a);
          System.out.println("Value of b: "+b);
        }
        A(int a,int b,int c)
        {
          System.out.println("Value of a: "+a);
          System.out.println("Value of b: "+b);
          System.out.println("Value of b: "+c);
        }
          public static void main(String ar[])
          {
                  A a1=new A(2,3);
                  A a2=new A(2,3,4);
          }
}
```

Difference between constructor and method in java

There are many differences between constructors and methods. They are given below.

Java Constructor	Java Method
Constructor is used to initialize the state of an object.	Method is used to expose behaviour of an object.
Constructor must not have return type.	Method must have return type.
Constructor is invoked implicitly.	Method is invoked explicitly.
The java compiler provides a default constructor if you don't have any constructor.	Method is not provided by compiler in any case.
Constructor name must be same as the class name.	Method name may or may not be same as class name.

Method Overloading/Function overloading/Polymorphism:

- The method having same name, but different numbers of arguments, type of parameter and return type is called function overloading
- In Java, it is possible to define two or methods at same time. This is known as overloading methods and such methods are known as overloaded methods. It is only possible if all the overloaded methods differ in either type of argument/number of arguments.
- It is one of the ways through which Java implements polymorphism.
- When Java encounters a call to an overloaded method, it simply executes that version of the method whose parameters match exactly to the arguments passed in the call.

Example:

```
class A
{
        void ArithmaticOperation()
        {
                int a=10,b=20;
                int c=a+b;
                System.out.println("sum="+c);
        }
        void ArithmaticOperation(int a,int b)
        {
                int c=a*b;
                System.out.println("multiplication="+c);
        }
        void ArithmaticOperation(float a,int b)
        {
                int c=a-b;
                System.out.println("sum="+c);
        }
        public static void main(String ar[])
        {
                A a1=new A();
                a1. ArithmaticOperation();
                a1. ArithmaticOperation(2,3);
                a1. ArithmaticOperation(3.0,2.0);
        }
}
```

Instance Variable Hiding:

- There can be situations where the instance variable names are same as the local variable name in a method. In such cases, the local variables of the method overlap instance variable thereby hiding them are called **instance variable hiding.**
- The above problem can be overcome using 'this' keyword.
- 'this' is always a reference to the object that has invoked the method that contains 'this' keyword.

E.g.:-

```
class A
{
    int a=0;
   void dis(int a)
   {
     int a=2;
    System.out.println("Value of a:"+a);// local variable hides a instance variable. Value of a:2
   System.out.println("Value of a:"+this.a); //instance variable hide local variable Value of a:0
   }
}
```

Usage of this keyword

1. this keyword can be used to refer current class instance variable.
2. this() can be used to invoke current class constructor.
3. this can be used to constructor reusing

Note: refer class program for this keyword.

Super keyword /Super class

- The **super** keyword in java is a reference variable that is used to refer immediate parent class object.
- Whenever you create the instance of subclass, an instance of parent class is created implicitly i.e. referred by super reference variable.
- Super() key word should be first statement in subclass constructors.
- **Usage of java super Keyword**
    1. super is used to refer immediate parent class instance variable.
    2. super() is used to invoke immediate parent class constructor.
    3. super is used to invoke immediate parent class method.

Example:

```
class A
{
        int a=10;
        void display()
        {
                System.out.println("super class method");
        }
        A()
        {
                System.out.println("A class constructors");
        }
        A(int a,int b)
        {
                System.out.println("sum="+(a+b));
        }
}
```

```
Class B extends A
{
        B()
        {
                Super(); // calls super class constructor
                Super.display(); // calls super class method
                Int c=Super.a; // calls super class instance variable

                System.out.println("A class constructors"+ c);
        }
        B(int a,int b)
        {
                super(3,3);
                System.out.println("sum="+(a+b));
        }
        Public static void main(String ar[])
        {
        B b1=new B();
        B b2=new B(2,3);
        }
}
```

output:

A class constructors

super class method

A class constructors:10

sum=6

sum=5

Static keyword:

- Static is a keyword that are used for share the same variable or method of a given class. This is used for a constant variable or a method that is the same for every instance of a class.
- The main method of a class is generally labeled static.
- No object needs to be created to use static variable or call static methods,
- To access static variable or methods from class to another class ,uses class name before the static variable or method to use them.
- Static method can not call non-static method.

- **In java language static keyword can be used for following**
    1. variable (also known as class variable)
    2. method (also known as class method)
    3. block

1) static variable/class variable

If any variable we declared as static is known as static variable or class variable.

The static variable allocate memory only once in class area at the time of class loading.

Example:

```
class A
{
        static int a=10;
        public static void main(String ar[])
        {
                System.out.println("static variable a="+a);
        }
}
```

2)static method:

If you apply static keyword with any method, it is known as static method.

- A static method belongs to the class rather than object of a class.
- A static method can be invoked without the need for creating an object of a class.

- static method can access static data member and can change the value of it.
- **Restrictions for static method**

1.The static method cannot use non static data member or call non-static method directly.

2.this and super cannot be used in static context.

Example:

```
class A
{
        static void add(int a,int b)
        {
                System.out.println("static method sum="+(a+b));
        }
        public static void main(String ar[])
        {
                A a1=new A();
                add(2,3); // static method invoked without object
        }
}
```

3) Java static block

- Is used to initialize the static data member.
- It is executed before main method at the time of classloading.

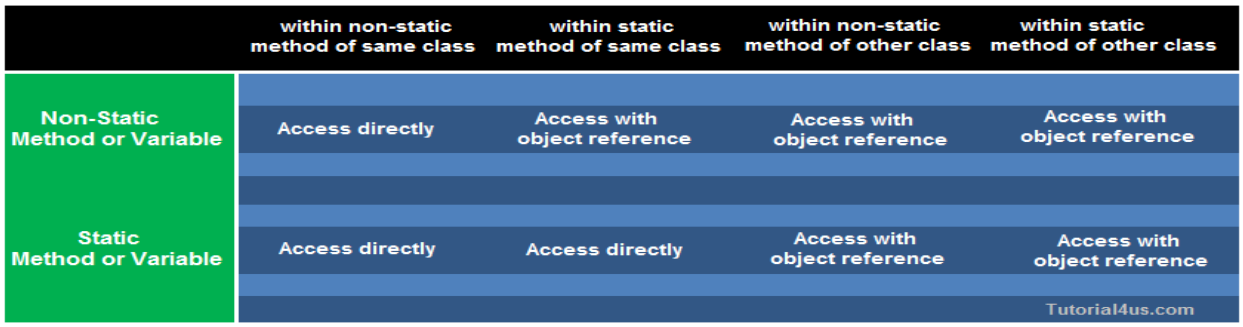
Example:

```
class A
{
        static
        {
                System.out.println("static block");
        }
}
```

Difference between non-static and static Method

	Non-Static method	Static method
	1.These method never be preceded by static keyword Example: void fun1() { .... }	These method always preceded by static keyword Example: static void fun2() { .... }
2	Memory is allocated multiple time whenever method is calling.	Memory is allocated only once at the time of class loading.
3	It is specific to an object so that these are also known as instance method.	These are common to every object so that it is also known as member method or class method.
4	These methods always access with object reference Syntax: **Objref.methodname();**	These property always access with class reference Syntax: **className.methodname();**
5	If any method wants to be execute multiple time that can be declare as non static.	If any method wants to be execute only once in the program that can be declare as static .

Accessing static methods and non static methods in different cases.



	within non-static method of same class	within static method of same class	within non-static method of other class	within static method of other class
Non-Static Method or Variable	Access directly	Access with object reference	Access with object reference	Access with object reference
Static Method or Variable	Access directly	Access directly	Access with object reference	Access with object reference

Tutorial4us.com

Inner Classes/nested class:

- If one class is existing within another class is known as inner class or nested class
- Inner class properties can be accessed in the outer class with the object reference but not directly.
- Outer class properties can be access directly within the inner class.
- Inner class properties can't be accessed directly or by creating directly object.
- Syntex:

```
Class A
{
Class B
{
}
}
```

Examlple:

```
class A
{
        int a=10;
        void add()
        {
                int a=10,b=20;
                int c=a+b;
                System.out.println("sum="+c);
        }

        class B
        {
                void display()
                {
                        System.out.println("outer class variable a="+a);
                        add();  //inner class can access the properties of outer class
                }
        }
```

```
        public static void main(String ar[])
        {
                A a1=new A();
                a1.add();
                B b1;
                b1=a1.new B();
                b1.display();  // to access the inner class properties

        }
}
```

Garbage Collection:

Memory deallocation is done automatically in Java. Garbage collection only occurs sporadically (if at all) during the execution of the program.

The finalize( ) Method:

Sometimes an object will need to perform some action when it is destroyed.

For example, if an object is holding some non-Java resource such as a file handle or character font, then we might want to make sure these resources are freed before an object is destroyed.

To handle such situations, Java provides a mechanism called *finalization.* By using finalization, we can define specific actions that will occur when an object is just about to be reclaimed by the garbage collector.

To add a finalizer to a class, we simply define the finalize( ) method. The Java run time calls that method whenever it is about to recycle an object of that class. Inside the finalize( ) method, we will specify those actions that must be performed before an object is destroyed.

The garbage collector runs periodically, checking for objects that are no longer referenced by any running state or indirectly through other referenced objects. Right before an asset is freed, the Java run time calls the finalize( ) method on the object.

It is important to understand that finalize( ) is only called just prior to garbage collection. It is not called when an object goes out-of-scope.

The **finalize( )** method has this general form:

```
protected void finalize( )
{
        // finalization code here
}
```

Here, the keyword **protected** is a specifier that prevents access to **finalize( )** by code defined outside its class.

```
Ex.
class Garbage
{
	public void sayHello()
	{
		System.out.println("Hello Welcome");
	}

	protected void finalize()
	{
		System.out.println("Inside finalize method..");
	}

}

public class GarbageCollection {

	public static void main(String[] args) {
		Garbage g = new Garbage();
		Garbage g1 = new Garbage();
		g.sayHello();

		g=null;
		g1=null;

		System.gc();
	}

}
```

Inheritance

- The process of obtaining the data members and methods from one class to another class is known as **inheritance**. It is one of the fundamental features of object-oriented programming.
- **Syntax of Java Inheritance**

```
class Subclass-name extends Superclass-name
{
  //methods and fields
}
```

The **extends keyword** indicates that you are making a new class that derives from an existing class.

Important points

- In the inheritance the class which is give data members and methods is known as base or super or parent class.
- The class which is taking the data members and methods is known as sub or derived or child class.
- The data members and methods of a class are known as features.
- The concept of inheritance is also known as re-usability or extendable classes or sub classing or derivation.

Why use Inheritance ?

- For Method Overriding
- For method overloading
- For code Re-usability

Types of Inheritance

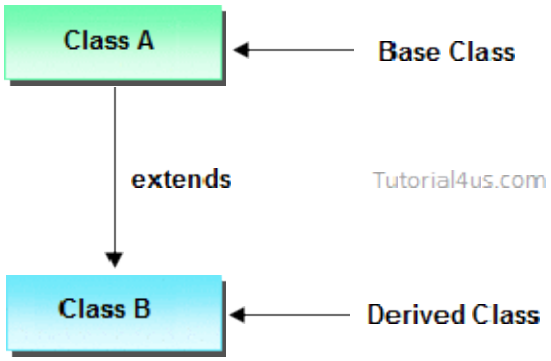
Based on number of ways inheriting the feature of base class into derived class we have five Inheritance type they are:

- Single inheritance
- Multiple inheritance(Interface)
- Hierarchical inheritance
- Multilevel inheritance
- Hybrid inheritance

1. Single inheritance

In single inheritance there exists single base class and single derived class.

Property of base class inherited into sub class and sub class having its own property.



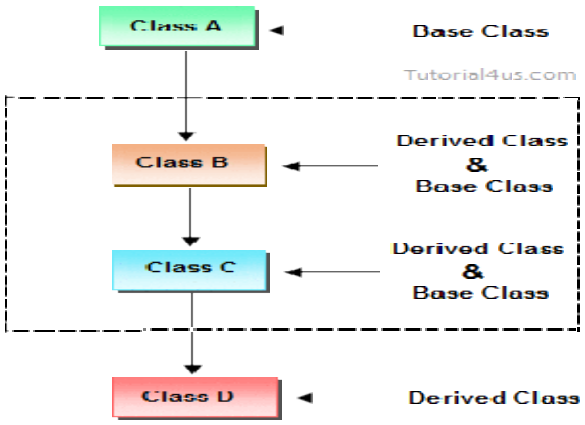
Example:

```
class A
{
        void display()
        {
                System.out.println("base class method");
        }
}
class B extends A
{
        void display2()
        {
                System.out.println("sub class methods");
```

```
    }
    public static void main(String ar[])
    {
        B a1=new B();
        a1.display();
        a1.display2();
    }
}
```

2.Multilevel inheritances

- In Multilevel inheritances there exists single base class, single derived class and multiple intermediate base classes.
- An intermediate base class is one in one context with access derived class and in another context same class access base class.



Example:

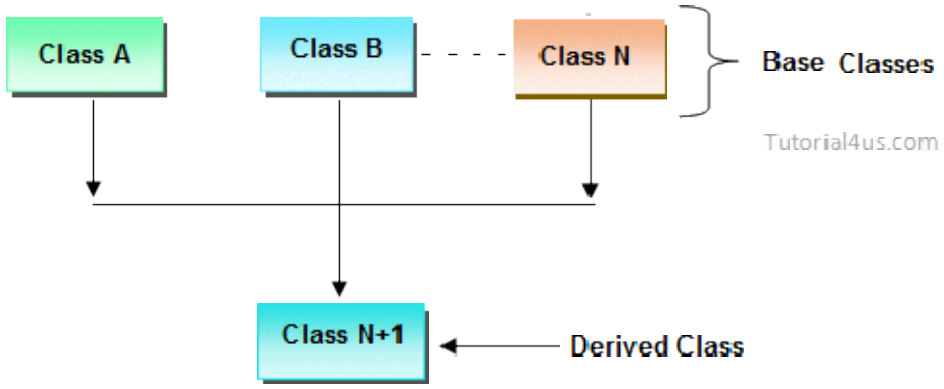
```
class A
{
    void display()
```

```
        {
        System.out.println("A class method");
        }
}
class B extends A
{
        void display1()
        {
                System.out.println("B class methods");
        }

class C extends B
{
        void display2()
        {
                System.out.println("C class methods");
        }
        public static void main(String ar[])
        {
                C a1=new C();
                a1.display();
                a1.display1();
                a1.display2();
        }
}
```

3.Multiple inheritance

In multiple inheritance there exist multiple classes and singel derived class.



The concept of multiple inheritance is not supported in java through concept of classes but it can be supported through the concept of **interface**.

Overloading:

Whenever same method name is exiting multiple times in the same class with different number of parameter or different order of parameters or different types of parameters is known as **method overloading**.

Example:

```
class A
{
        void add(int a,int b)
        {
                int c=a+b;
                System.out.println("Sum of two nos="+c);
        }
}
class B extends A
{
        void add(float a,int b)
        {
                float sum=a+b;
                System.out.println("Sum ="+sum);
        }
```

```
        public static void main(String ar[])
        {
                B a1=new B();
                a1.add(2,3);
                a1.add(4.0 , 2);
        }
}
```

Overriding:

- Whenever same method name is existing in both base class and derived class with same types of parameters or same order of parameters is known as **method Overriding**.
- **Rules for Method Overriding**
  1) method must have same name as in the parent class.
  2) method must have same parameter as in the parent class.
  3) must be IS-A relationship (inheritance).

Example:

```
class A
{
        void add(int a,int b)
        {
                int c=a+b;
                System.out.println("Sum of two nos="+c);
        }
}
class B extends A
{
        void add(int a,int b)  //overriding method
        {
                float sum=a+b;
                System.out.println("Sum ="+sum);
```

```
        }
        public static void main(String ar[])
        {
                B a1=new B();
                a1.add(2,3);
                a1.add(4.0 , 2);
        }
}
```

Advantage of Java Method Overriding

- Method Overriding is used to provide specific implementation of a method that is already provided by its super class.
- Method Overriding is used for Runtime Polymorphism

Final Keyword

In java language final keyword can be used in following way.

- Final at variable level
- Final at method level
- Final at class level

1. **Final at variable level**

Final keyword is used to make a variable as a constant. This is similar to const in other language. A variable declared with the final keyword cannot be modified by the program after initialization.

Example:

```
class A
{
        final  int a=10;
        public static void main(String ar[])
        {
                System.out.println("static variable a="+a1.a); // no error
```

```
        System.out.println("static variable a="+a1.a++); //final variable cannot be
    modified (error)
    }
}
```

2.Final at method level

- It makes a method final, meaning that sub classes cannot override this method. The compiler checks and gives an error if you try to override the method.
- When we want to restrict overriding, then make a method as a final.
- Example:

```
class A
{
    final void add()
    {
        System.out.println("sum="+(2+3));
    }
}
class B extends A
{
    void add()   // error because final method cannot override
    {
        System.out.println("sum="+(2+3));
    }
    public static void main(String ar[])
    {
        A a1=new A();
        a1.add();
```

```
    }
}
```

3. Final at class level

It makes a class final, meaning that the class cannot be inheriting by other classes. When we want to restrict inheritance then make class as a final.

Example:

```
final class A
{
    void add()
    {
        System.out.println("sum="+(2+3));
    }
}
class B extends A  // error because final class cannot inherited.
{
    public static void main(String ar[])
    {
        A a1=new A();
        a1.add();
    }
}
```

Abstract class

- A class that is declared with abstract keyword, is known as **abstract class**.
- An abstract class is one which is containing:
    - Final variable

- Function/method declaration
- Function /method definition.

- An abstract method is one which contains only declaration or prototype but it never contains body or definition.
- In order to make any undefined method as abstract whose declaration is must be predefined by abstract keyword.
- Any class that contains 1 or more abstract methods must be declared as abstract class.
- Abstract classes cannot have objects.
- A subclass must always define the abstract methods of its superclass or else must be declared as 'abstract'
- Syntex:

```
abstract class class_name
{
    final datatype v1,v2;
    Void fun();
    Void fun1()
    {
    }
}
```

Example:

```
abstract class A
{
    final int a=10;
    abstract void add();
    void add(int a,int b)
    {
        int c=a+b;
        System.out.println("Sum="+c);
    }
}
class B extends A
```

```
{
        void add()
        {
                int a=10,b=20;
                int c=a+b;
                System.out.println("sum="+c);
        }
        public static void main(String ar[])
        {
                B a1=new B();
                a1.add();
                a1.add(2,3);
        }
}
```

Java does not support multiple inheriatance:

- **Java does not support the multiple inheritance due to** reduce the complexity and simplify the language, multiple inheritance is not supported in java.
- Consider a scenario where A, B and C are three classes. The C class inherits A and B classes. If A and B classes have same method and you call it from child class object, there will be ambiguity to call method of A or B class.
- Since compile time errors are better than runtime errors, java renders compile time error if you inherit 2 classes. So whether you have same method or different, there will be compile time error now.
- **Problem of Multiple inheritance can be solved by using interface**

Interface

- is similar to class which is collection of public static final variables (constants) and abstract methods.
- The interface is a mechanism to achieve fully abstraction in java. There can be only abstract methods in the interface. It is used to achieve fully abstraction and multiple inheritance in Java.

- Syntex:

  interface interface_name

  {

  final variable;

  Method declaration;

  }

  - One interface can extends many interface
  - Class can implements many interface but one extends class

Example:

```
interface A
{
   int a=10;
   void add();
}
interface B
{
   void add(int ,int);
}
class C implements A,B
{
   public void add()
   {
            qint a=10,b=20;
            int c=a+b;
            System.out.println("sum="+c);
   }
   public void add(int a,int b)
   {
            int c=a*b;
        System.out.println("product="+c);
   }
```

```
    public static void main(String ar[])
    {
            C c1=new C();
            c1.add();
            c1.add(2,3);
    }
}
```

Why we use Interface ?

- It is used to achieve fully abstraction.
- By using Interface, you can achieve multiple inheritance in java.
- It can be used to achieve loose coupling.

properties of Interface

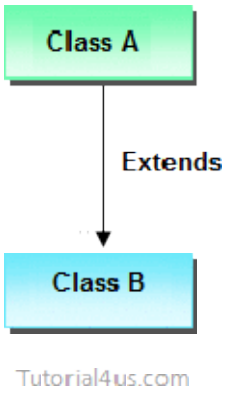
- An interface is implicitly abstract. So we no need to use the abstract keyword when declaring an interface.
- Each method in an interface is also implicitly abstract, so the abstract keyword is not needed.
- Methods in an interface are implicitly public.
- All the data members of interface are implicitly public static final.

Rules for implementation interface

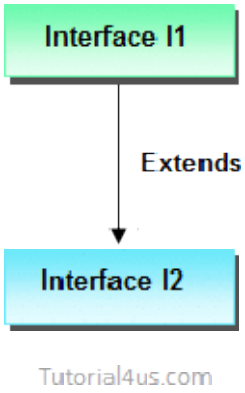
- A class can implement more than one interface at a time.
- A class can extend only one class, but implement many interfaces.
- An interface can extend another interface, similarly to the way that a class can extend another class.

Relationship between class and Interface

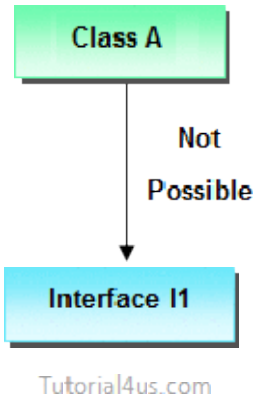
- Any class can extends another class



- Any Interface can extends another Interface.



- Any class can Implements another Interface



Difference between Abstract class and Interface

	Abstract class	Interface

1	It is collection of abstract method and concrete methods.	It is collection of abstract method.
2	There properties can be reused commonly in a specific application.	There properties commonly usable in any application of java environment.
3	It does not support multiple inheritance.	It support multiple inheritance.
4	Abstract class is preceded by abstract keyword.	It is preceded by Interface keyword.
5	Which may contain either variable or constants.	Which should contains only constants.
6	The default access specifier of abstract class methods are default.	There default access specifier of interface method are public.
7	These class properties can be reused in other class using extend keyword.	These properties can be reused in any other class using implements keyword.
8	Inside abstract class we can take constructor.	Inside interface we can not take any constructor.
9	For the abstract class there is no restriction like initialization of variable at the time of variable declaration.	For the interface it should be compulsory to initialization of variable at the time of variable declaration.
10	There are no any restriction for abstract class variable.	For the interface variable can not declare variable as private, protected, transient, volatile.
11	There are no any restriction for abstract class method modifier that means we can use any modifiers.	For the interface method can not declare method as strictfp, protected, static, native, private, final, synchronized.

Exception Handling

The process of converting system error messages into user friendly error message is known as **Exception handling**. This is one of the powerful feature of Java to handle run time error and maintain normal flow of java application.

Exception

An **Exception** is an event, which occurs during the execution of a program, that disrupts the normal flow of the program's Instructions.

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

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.

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

Error

Error is irrecoverable e.g. OutOfMemoryError, VirtualMachineError, AssertionError etc

Common scenarios where exceptions may occur

There are given some scenarios where unchecked exceptions can occur. They are as follows:

1) Scenario where ArithmeticException occurs

If we divide any number by zero, there occurs an ArithmeticException.

```
1. int a=50/0;//ArithmeticException
```

---

2) Scenario where NullPointerException occurs

If we have null value in any variable, performing any operation by the variable occurs an NullPointerException.

```
1. String s=null;
2. System.out.println(s.length());//NullPointerException
```

---

3) Scenario where NumberFormatException occurs

The wrong formatting of any value, may occur NumberFormatException. Suppose I have a string variable that have characters, converting this variable into digit will occur NumberFormatException.

```
1. String s="abc";
2. int i=Integer.parseInt(s);//NumberFormatException
```

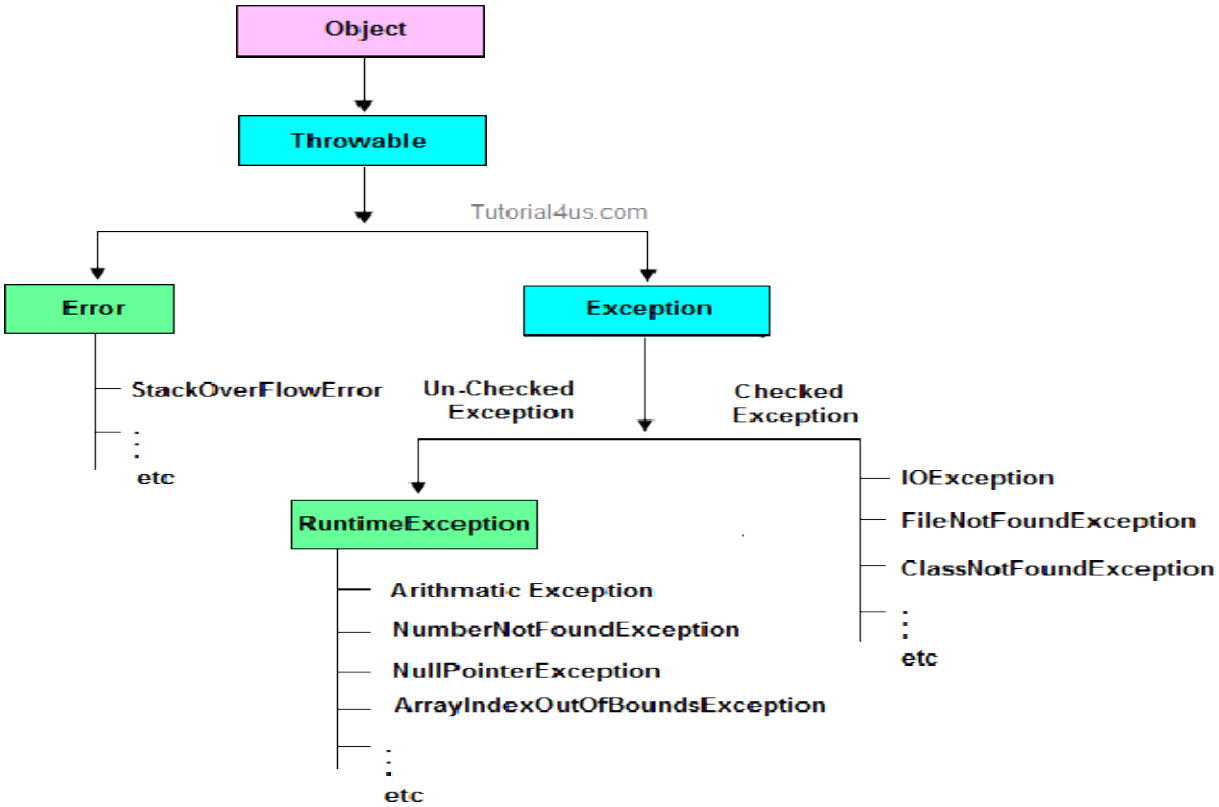
---

4) Scenario where ArrayIndexOutOfBoundsException occurs

If you are inserting any value in the wrong index, it would result ArrayIndexOutOfBoundsException as shown below:

```
1. int a[]=new int[5];
2. a[10]=50; //ArrayIndexOutOfBoundsException
```

Hierarchy of Exception classes



Handling the Exception

- Handling the exception is nothing but converting system error generated message into user friendly error message in others word whenever an exception occurs in the java application.
- JVM will create an object of appropriate exception of sub class and generates system error message, these system generated messages are not understandable by user so need to convert it into user-friendly error message.
- **Use Five keywords for Handling the Exception**
    - try
    - catch
    - finally
    - throws
    - throw

Syntax for handling the exception

```
try
{
  // statements causes problem at run time
}
catch(type of exception-1 object-1)
{
  // statements provides user friendly error message
}
catch(type of exception-2 object-2)
{
  // statements provides user friendly error message
}
finally
{
  // statements which will execute compulsory
}
```

try block

Inside **try block** we write the block of statements which causes executions at run time in other words try block always contains problematic statements.

Important points about try block

- If any exception occurs in try block then CPU controls comes out to the try block and executes appropriate catch block.

- After executing appropriate catch block, even through we use run time statement, CPU control never goes to try block to execute the rest of the statements.
- Each and every try block must be immediately followed by catch block that is no intermediate statements are allowed between try and catch block.

Syntax

```
try
{
  .....
}
catch()
{
  ....
}
```

catch block

Inside **catch** block we write the block of statements which will generates user friendly error messages.

catch block important points

- Catch block will execute exception occurs in try block.
- You can write multiple catch blocks for generating multiple user friendly error messages to make your application strong. You can see below example.
- At a time only one catch block will execute out of multiple catch blocks.
- in catch block you declare an object of sub class and it will be internally referenced by JVM.

finally block

Inside **finally**block we write the block of statements which will relinquish (released or close or terminate) the resource (file or database) where data store permanently.

finally block important points

- Finally block will execute compulsory
- Writing finally block is optional.
- You can write finally block for the entire java program
- In some of the circumstances one can also write try and catch block in finally block

Example:

```
Class A
{
        A()
        {
                try
```

```
                {
                        int a=b/0;
                }
                catch(Exception e)
                {
                        System.out.println(e);
                }
                finally
                {
                        System.out.println("finally block always executes");
                }
        }
        public static void main(String ar[])
        {
                A a1=new A();
        }
}
```

Multiple catch block

You can write multiple catch blocks for generating multiple user friendly error messages to make your application strong.

Single try block can have multiple catch block, Complier calls the appropriate catch block depending upon the exception generated from the try block.

Syntex:

```
try{
}
catch(Exception_type1 obj)
{
}
 catch(Exception_typen obj)
{
}
```

Example:

```
Class A
{
        A()
        {
                try
                {
                        int a=b/0;
                }
                catch(ArrayIndexOutOfBoundException e)
                {
                        System.out.println(e);
                }
                catch(ArithmaticException e)
                {
                        System.out.println(e);
                }
```

```
        }
        public static void main(String ar[])
        {
                A a1=new A();
        }
}
```

throw

throw is a keyword in java language which is used to throw any user defined class exception to the same signature of method in which the exception is raised.

Syntax

```
class A
{
try
{
throw new MyClass(string);
}
finally
{
}
}
```

Example:

```
class A
{
        A()
        {
                try
                {
                        int a=2,b=3;
                        If(a<b) throw new MyClass("a is smaller then B");
                }
                finally
                {
                        System.out.println("finally block execute");
                }
        }
        public static void main(String ar[])
        {
                A a1=new A();
        }
}
```

```
class MyClass
{
        MyClass(String a)
        {
                System.out.println(a);
        }
}
```

throws

throws is a keyword in java language which is used to throw the exception which is raised in the called method to it's calling method throws keyword always followed by method signature.

Syntex:

```
returnType methodName(parameter)throws Exception_class....
{
.....
}
```

Example:

```
class A
{
        public void dis() throws IllegelAccessException
        {
                new MyClass(); // if my class is not exit then illegelAccessException will occurs
        }
        public static void main(String ar[])
        {
                A a1=new A();
                try{
                        a1.dis();
                }
                catch(IllegelAccessException e)
                {
                        System.out.println(e);
                }
```

```
    }
```

Nested Try block:

- Try within another try block is called as nested try block
- Inner try block will execute weather exception occurs or may not occurs.
- Each try block should contain either catch block or finally block
- Syntex:

```
try
{
try
{
}
catch(Exception_type obj)
{
}
}
catch(Exception_type obj)
{
}
```

Example:

```
class A
{
        A()
        {
                try
                {
                        int a=5/0;
                        try
                        {
                                int a[]=new int[5];
                                A[10]=10;
```

```
                }


                catch(Exception e)
                {
                        System.out.println(e);
                }
        }
        catch(Exception e)
        {
                System.out.println(e);
        }
    }
    public static void main(String ar[])
    {
        A a1=new A();
    }
}
```

Difference between throw and throws

	Throw	Throws
1	throw is a keyword used for hitting and generating the exception which are	throws is a keyword which gives an indication to the specific method to place the common

	occurring as a part of method body	exception methods as a part of try and catch block for generating user friendly error messages
2	The place of using throw keyword is always as a part of method body.	The place of using throws is a keyword is always as a part of method heading
3	When we use throw keyword as a part of method body, it is mandatory to the java programmer to write throws keyword as a part of method heading	When we write throws keyword as a part of method heading, it is optional to the java programmer to write throw keyword as a part of method body.

Packages

- *Packages* are containers for classes that are used to keep the class name space compartmentalized.
- For example, a package allows you to create a class named **List**, which you can store in your own package without concern that it will collide with some other class named **List** stored elsewhere.
- Packages are stored in a hierarchical manner and are explicitly imported into new class definitions.

Defining a Package

- To create a package is quite easy: simply include a package command as the first statement in a Java source file. Any classes declared within that file will belong to the specified package.
- The package statement defines a name space in which classes are stored. If you omit the package statement, the class names are put into the default package, which has no name.
- This is the general form of the package statement

```
package pkg;
package MyPackage;
```

// A simple package

```
package MyPack;

class Balance

{
        String name;
        double bal;


    Balance(String n, double b)

    {           name = n;
                bal = b;
    }
    void show()

    {
         if(bal<0)
        System.out.print("--> ");
        System.out.println(name + ": $" + bal);
    }

}

class AccountBalance

{
          public static void main(String args[])

            {
                Balance b1 = new Balance("XYZ", 4.5);
               b1.show();

            }
}
```

Access Protection

- Packages add another dimension to access control.

	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

Import Packages

- Since classes within packages must be fully qualified with their package name or names, it could become tedious to type in the long dot-separated package path name for every class you want to use.
- Java includes the **import** statement to bring certain classes, or entire packages, into visibility. Once imported, a class can be referred to directly, using only its name.
- In a Java source file, import statements occur immediately following the package statement (if it exists) and before any class definitions.

  Ex.. import java.util.Date;

  import java.io.*