throw keyword in Exception Handling

- We can create exceptions explicitly. It is also used for User defined exception.
- To create an object of any exception type. We used throw.

throw Program:

```
package com.exception;
import java.io.IOException;
public class DemoThrow {
	public static void main(String[] args) {
		DemoThrow obj=new DemoThrow();
		obj.display();
	}
	void display()
	{
		try{
			throw new IOException("Explicit IOException created by user");
		}
		catch(IOException e){
			System.out.println("Exception is "+e);
}	}	}
```

Output:

Exception is java.io.IOException: Explicit IOException created by user

throws keyword in Exception Handling

- throws keyword used when programmer know that method may cause exception, but method unable to handle exception that why programmer directly throws the exception.
- Throws keyword used with method declaration.

throws Program:

```
package com.exception;
public class DemoThrows {
	public static void main(String[] args) throws Exception {
		int x=5,y=0;
		int c=x/y;
		System.out.println("The result is="+c);
}	}
```

Output:

Exception in thread "main" java.lang.ArithmeticException: / by zero
at com.exception.DemoThrows.main(DemoThrows.java:7)

Topic: Inheritance

Definition:

- One class derived from another one.
- One class acquires properties (Methods + Data Members) of another class called as Inheritance

Inheritance in java is a mechanism in which one object acquires all the properties and behaviors of parent object.

The idea behind inheritance in java is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of parent class, and you can add new methods and fields also.

Inheritance represents the **IS-A relationship**, also known as *parent-child* relationship.

Why use inheritance in java

- For Method Overriding (so runtime polymorphism can be achieved).
- For Code Reusability.

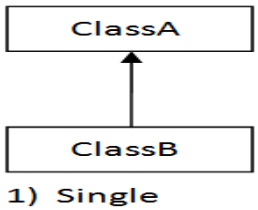
Syntax of Java Inheritance

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

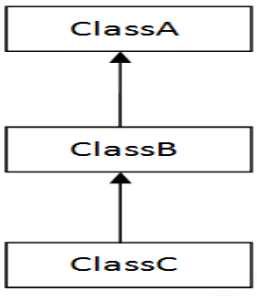
Types of inheritance in java

On the basis of class, there can be three types of inheritance in java: single, multilevel and hierarchical.

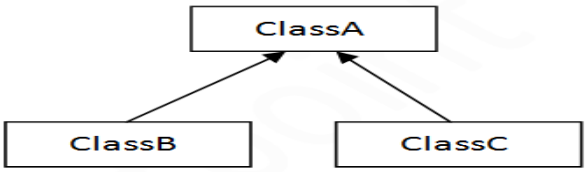
In java programming, multiple and hybrid inheritance is supported through interface only. We will learn about interfaces later.



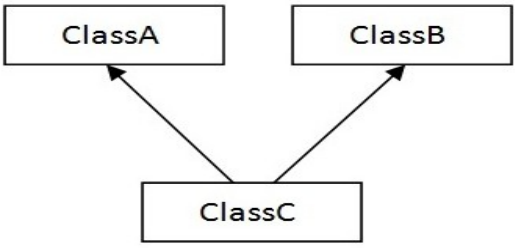
1) Single



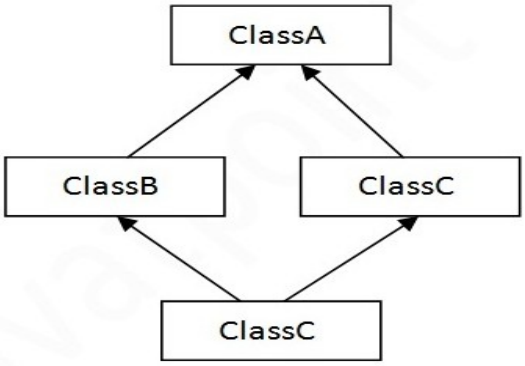
2) Multilevel



3) Hierarchical



4) Multiple



5) Hybrid

Q) Why multiple inheritance is not supported in java?

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.

Example:

```
class A{
void msg(){System.out.println("Hello");}
}
class B{
void msg(){System.out.println("Welcome");}
}
class C extends A,B{//suppose if it were

 Public Static void main(String args[]){
   C obj=new C();
   obj.msg();//Now which msg() method would be invoked?  Ambiguity.
}
}
```

Method Overriding in Java

If subclass (child class) has the same method as declared in the parent class, it is known as **method overriding in java**.

In other words, If subclass provides the specific implementation of the method that has been provided by one of its parent class, it is known as method overriding.

Usage 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

Rules for Java 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).

Program Method Overriding

```
class Vehicle{
void run(){System.out.println("Vehicle is running");}
}
class Bike2 extends Vehicle{
void run(){System.out.println("Bike is running safely");}

public static void main(String args[]){
Bike2 obj = new Bike2();
obj.run();
}
```

Output:Bike is running safely

Note:
static method cannot be overridden.

Difference between method overloading and method overriding in java

No.	Method Overloading	Method Overriding
1)	Method overloading is used *to increase the readability* of the program.	Method overriding is used *to provide the specific implementation* of the method that is already provided by its super class.
2)	Method overloading is performed *within class*.	Method overriding occurs *in two classes* that have IS-A (inheritance) relationship.
3)	In case of method overloading, *parameter must be different*.	In case of method overriding, *parameter must be same*.
4)	Method overloading is the example of *compile time polymorphism*.	Method overriding is the example of *run time polymorphism*.
5)	In java, method overloading can't be performed by changing return type of the method only. *Return type can be same or different* in method overloading. But you must have to change the parameter.	*Return type must be same or covariant* in method overriding.

Simple example of Inheritance

```
class Parent
{
    public void p1()
    {
        System.out.println("Parent method");
    }
}
public class Child extends Parent {
    public void c1()
    {
        System.out.println("Child method");
    }
    public static void main(String[] args)
    {
        Child cobj = new Child();
        cobj.c1();   //method of Child class
        cobj.p1();   //method of Parent class
    }
}
```

Output

```
Child method
Parent method
```

Multilevel Inheritance example program in Java

```
Class X
{
   public void methodX()
   {
     System.out.println("Class X method");
   }
}
Class Y extends X
{
public void methodY()
{
System.out.println("class Y method");
}
}
Class Z extends Y
{
   public void methodZ()
   {
     System.out.println("class Z method");
   }
   public static void main(String args[])
   {
     Z obj = new Z();
     obj.methodX(); //calling grand parent class method
     obj.methodY(); //calling parent class method
     obj.methodZ(); //calling local method
  }
}
```

super keyword in java

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 superreference variable.

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.

1) super is used to refer immediate parent class instance variable.

Problem without super keyword

```
class Vehicle{
  int speed=50;
}
class Bike3 extends Vehicle{
  int speed=100;
  void display(){
   System.out.println(speed);//will print speed of Bike
   }
  public static void main(String args[]){
   Bike3 b=new Bike3();
   b.display();
}  }
```

Output:100

Solution by super keyword

```
//example of super keyword

class Vehicle{
  int speed=50;
}

class Bike4 extends Vehicle{
  int speed=100;

  void display(){
   System.out.println(super.speed);//will print speed of Vehicle now
   }
```

```
    public static void main(String args[]){
     Bike4 b=new Bike4();
     b.display();
     } }
```

Output:50

2) super is used to invoke parent class constructor.

The super keyword can also be used to invoke the parent class constructor as given below:

```
class Vehicle{
  Vehicle(){System.out.println("Vehicle is created");}
}


class Bike5 extends Vehicle{
  Bike5(){
   super();//will invoke parent class constructor
   System.out.println("Bike is created");
  }
  public static void main(String args[]){
   Bike5 b=new Bike5();
}   }
```

Output:Vehicle is created
Bike is created

3) super can be used to invoke parent class method

The super keyword can also be used to invoke parent class method. It should be used in case subclass contains the same method as parent class as in the example given below:

```
class Person{
void message(){System.out.println("welcome");}
}

class Student16 extends Person{
void message(){System.out.println("welcome to java");}

void display(){
message();//will invoke current class message() method
super.message();//will invoke parent class message() method
}
```

```
public static void main(String args[]){
Student16 s=new Student16();
s.display();
}
}
```

Output: welcome to java
welcome

Note : Super class reference pointing to Sub class object.

In context to above example where Class B extends class A.

```
A a=new B();
```

is legal syntax because of IS-A relationship is there between class A and Class B.

Q. Can you use both this() and super() in a Constructor?

NO, because both super() and this() must be first statement inside a constructor. Hence we cannot use them together.

Final Keyword In Java

The **final keyword** in java is used to restrict the user. The java final keyword can be used in many context. Final can be:

1. variable
2. method
3. class

The final keyword can be applied with the variables, a final variable that have no value it is called blank final variable or uninitialized final variable. It can be initialized in the constructor only. The blank final variable can be static also which will be initialized in the static block only. We will have detailed learning of these. Let's first learn the basics of final keyword.

1) Java final variable

If you make any variable as final, you cannot change the value of final variable(It will be constant).

Example of final variable

There is a final variable speedlimit, we are going to change the value of this variable, but It can't be changed because final variable once assigned a value can never be changed.

```
class Bike9{
 final int speedlimit=90;//final variable
 void run(){
  speedlimit=400;
 }
 public static void main(String args[]){
```

```
 Bike9 obj=new  Bike9();
 obj.run();
 }
}//end of class
```

Output:Compile Time Error

2) Java final method

If you make any method as final, you cannot override it.

Example of final method

```
class Bike{
  final void run(){System.out.println("running");}
}
class Honda extends Bike{
   void run(){System.out.println("running safely with 100kmph");}
   public static void main(String args[]){
   Honda honda= new Honda();
   honda.run();
   }  }
```

Output:Compile Time Error

3) Java final class

If you make any class as final, you cannot extend it.

Example of final class

```
final class Bike{}
class Honda1 extends Bike{
  void run(){System.out.println("running safely with 100kmph");}

  public static void main(String args[]){
  Honda1 honda= new Honda();
  honda.run();
  }
}
```

Output:Compile Time Error

Q) Is final method inherited?

Ans) Yes, final method is inherited but you cannot override it. For Example:

```
class Bike{
  final void run(){System.out.println("running...");}
```

```
}
class Honda2 extends Bike{
  public static void main(String args[]){
   new Honda2().run();
  }
}
```

Output:running...

Que) Can we initialize blank final variable?

Yes, but only in constructor. For example:

```
class Bike10{
  final int speedlimit;//blank final variable

  Bike10(){
  speedlimit=70;
  System.out.println(speedlimit);
  }
 public static void main(String args[]){
   new Bike10();
 }
}
```

Output:70

Q) Can we declare a constructor final?

No, because constructor is never inherited.

Abstract class in Java

A class that is declared with abstract keyword, is known as abstract class in java. It can have abstract and non-abstract methods (method with body).

Abstraction in Java

Abstraction is a process of hiding the implementation details and showing only functionality to the user.

Another way, it shows only important things to the user and hides the internal details for example sending sms, you just type the text and send the message. You don't know the internal processing about the message delivery.

Abstraction lets you focus on what the object does instead of how it does it.

Ways to achieve Abstaction

There are two ways to achieve abstraction in java

1. Abstract class (0 to 100%)

2. Interface (100%)

Abstract class in Java

A class that is declared as abstract is known as **abstract class**. It needs to be extended and its method implemented. It cannot be instantiated.

Example abstract class

```
1.  abstract class A{}
```

abstract method

A method that is declared as abstract and does not have implementation is known as abstract method.

Example abstract method

```
1.  abstract void printStatus();//no body and abstract
```

Example of abstract class that has abstract method

In this example, Bike the abstract class that contains only one abstract method run. It implementation is provided by the Honda class.

```
abstract class Bike{
  abstract void run();
}
class Honda4 extends Bike{
void run(){System.out.println("running safely..");}
public static void main(String args[]){
 Bike obj = new Honda4();
 obj.run();
}  }
```

Output: running safely..

Example:

```
abstract class Shape{
abstract void draw();
}
//In real scenario, implementation is provided by others i.e. unknown by end user
class Rectangle extends Shape{
void draw(){System.out.println("drawing rectangle");}
}
```

```
class Circle1 extends Shape{
void draw(){System.out.println("drawing circle");}
}
//In real scenario, method is called by programmer or user
class TestAbstraction1{
public static void main(String args[]){
Shape s=new Circle1();//In real scenario, object is provided through method e.g. getShape(
) method
s.draw();
 }  }
```

Output: drawing circle

Abstract class having constructor, data member, methods etc.

An abstract class can have data member, abstract method, method body, constructor and even main() method.

```
//example of abstract class that have method body
 abstract class Bike{
   Bike(){System.out.println("bike is created");}
   abstract void run();
   void changeGear(){System.out.println("gear changed");}
 }
 class Honda extends Bike{
 void run(){System.out.println("running safely..");}
 }
 class TestAbstraction2{
 public static void main(String args[]){
  Bike obj = new Honda();
  obj.run();
  obj.changeGear();
 }
}
```

Output: bike is created
running safely. gear changed

Rule: If there is any abstract method in a class, that class must be abstract.

```
1. class Bike12{
2. abstract void run();
3. }
```

Output: compile time error

Rule: If you are extending any abstract class that have abstract method, you must either provide the implementation of the method or make this class abstract.

Topic: Interface

An **interface in java** is a blueprint of a class. It has static constants and abstract methods only.

The interface in java is **a mechanism to achieve fully abstraction**. There can be only abstract methods in the java interface not method body. It is used to achieve fully abstraction and multiple inheritance in Java.

Java Interface also **represents IS-A relationship**.

It cannot be instantiated just like abstract class.

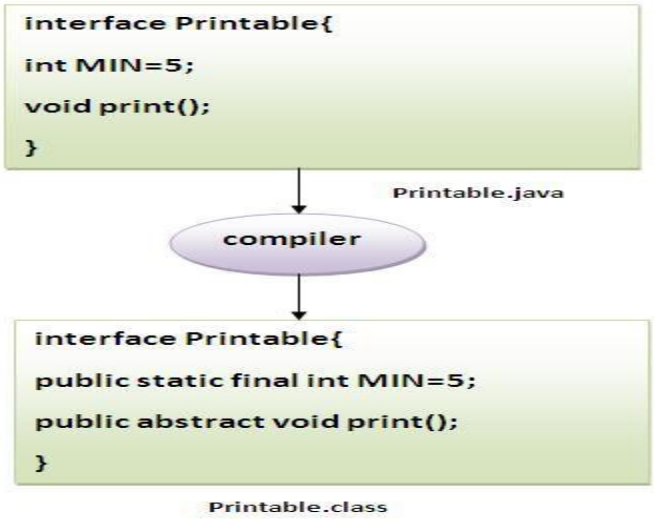
Why use Java interface?

There are mainly three reasons to use interface. They are given below.

- It is used to achieve fully abstraction.
- By interface, we can support the functionality of multiple inheritance.
- It can be used to achieve loose coupling.

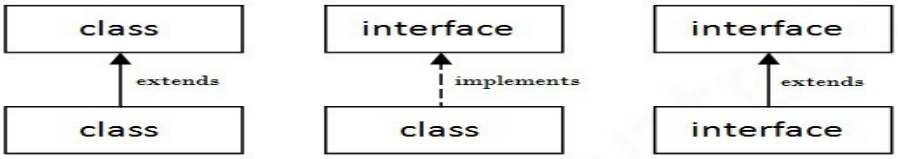
The java compiler adds public and abstract keywords before the interface method and public, static and final keywords before data members.

In other words, Interface fields are public, static and final by default, and methods are public and abstract.



Understanding relationship between classes and interfaces

As shown in the figure given below, a class extends another class, an interface extends another interface but a **class implements an interface**.



Simple example of Java interface

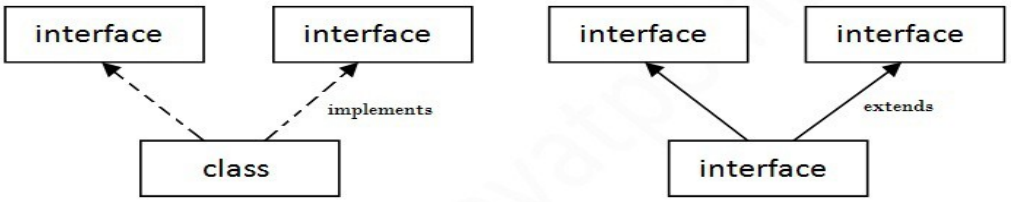
In this example, Printable interface have only one method, its implementation is provided in the A class.

```
interface printable{
void print();
}
 class A6 implements printable{
public void print(){System.out.println("Hello");}
 public static void main(String args[]){
A6 obj = new A6();
obj.print();
 }  }
```

Output:Hello

Multiple inheritance in Java by interface

If a class implements multiple interfaces, or an interface extends multiple interfaces i.e. known as multiple inheritance.



Multiple Inheritance in Java

```
interface Printable{
void print();
}
interface Showable{
void show();
}
```

```
class A7 implements Printable,Showable{
   public void print(){System.out.println("Hello");}
   public void show(){System.out.println("Welcome");}
public static void main(String args[]){
A7 obj = new A7();
obj.print();
obj.show();
 }   }
```

Output:Hello
Welcome

Q) Multiple inheritance is not supported through class in java but it is possible byinterface, why?

As we have explained in the inheritance chapter, multiple inheritance is not supported in case of class. But it is supported in case of interface because there is no ambiguity as implementation is provided by the implementation class. For example:

```
interface Printable{
void print();
}
interface Showable{
void print();
}

class TestTnterface1 implements Printable,Showable{
public void print(){System.out.println("Hello");}
public static void main(String args[]){
TestTnterface1 obj = new TestTnterface1();
obj.print();
 }
}
```

Output: Hello

As you can see in the above example, Printable and Showable interface have same methods but its implementation is provided by class TestTnterface1, so there is no ambiguity.

Interface inheritance

A class implements interface but one interface extends another interface .

```
interface Printable{
void print();
}
interface Showable extends Printable{
void show();
}
class Testinterface2 implements Showable{
```

```
public void print(){System.out.println("Hello");}
public void show(){System.out.println("Welcome");}

public static void main(String args[]){
Testinterface2 obj = new Testinterface2();
obj.print();
obj.show();
 }
}
```

Output: Hello
Welcome

Q) What is marker or tagged interface?

An interface that have no member is known as marker or tagged interface. For example: Serializable, Cloneable, Remote etc. They are used to provide some essential information to the JVM so that JVM may perform some useful operation.

```
//How Serializable interface is written?
public interface Serializable{
}
```

Nested Interface in Java

Note: An interface can have another interface i.e. known as nested interface. We will learn it in detail in the nested classes chapter. For example:

```
1. interface printable{
2.  void print();
3.  interface MessagePrintable{
4.    void msg();
5.  }
6. }
```

Difference between abstract class and interface

Abstract class and interface both are used to achieve abstraction where we can declare the abstract methods. Abstract class and interface both can't be instantiated.

But there are many differences between abstract class and interface that are given below.

Abstract class	Interface
1) Abstract class can **have abstract and non-abstract**methods.	Interface can have **only abstract** methods.
2) Abstract class **doesn't support multiple inheritance**.	Interface **supports multiple inheritance**.

3) Abstract class **can have final, non-final, static and non-static variables**.	Interface has **only static and final variables**.
4) Abstract class **can have static methods, main method and constructor**.	Interface **can't have static methods, main method or constructor**.
5) Abstract class **can provide the implementation of interface**.	Interface **can't provide the implementation of abstract class**.
6) The **abstract keyword** is used to declare abstract class.	The **interface keyword** is used to declare interface.

7) **Example:**
public abstract class Shape{
public abstract void draw();
}
	Example:
	public interface Drawable{
	void draw();
	}

Simply, abstract class achieves partial abstraction (0 to 100%) whereas interface achieves fully abstraction (100%).

Example of abstract class and interface in Java

```
//Creating interface that has 4 methods
interface A{
void a();//bydefault, public and abstract
void b();
void c();
void d();
}
//Creating abstract class that provides the implementation of one method of A interface
abstract class B implements A{
public void c(){System.out.println("I am C");}
}
//Creating subclass of abstract class, now we need to provide the implementation of rest of the methods
class M extends B{
public void a(){System.out.println("I am a");}
public void b(){System.out.println("I am b");}
public void d(){System.out.println("I am d");}
}
  //Creating a test class that calls the methods of A interface
class Test5{
public static void main(String args[]){
A a=new M();
a.a();
a.b();
a.c();
```

a.d(); }}

Output:

```
I am a
I am b
I am c
I am d
```