**1. What do you mean by OOPs Concept ? Describe the different features of OOPs.**

Object Oriented Programming is a paradigm that provides many concepts such as inheritance, data binding, polymorphism etc.

Simula is considered as the first object-oriented programming language. The programming paradigm where everything is represented as an object, is known as truly object-oriented programming language.

Smalltalk is considered as the first truly object-oriented programming language.

**Class/Object**

Enterire OOP methododlogy has been derived from a single root concept, called object. An object is anything that really exists in the world and can be distinguished from others. This definition specifies that everything in this world is an object. For example, a table, a ball, a car, a dog, a person etc, every thing will come under objects.

If something does not really exist, then it is not an object. For example, our thoughts, imagination, plans, ideas etc are not objects, because they do not physically exists.

Every has properties and can perform certain actions. For example, let us take a person whose name is “Raju”. Raju is an object because he exists physically. He has properties like name, age, sex etc. These properties can be represented by variables in our programming like as follows

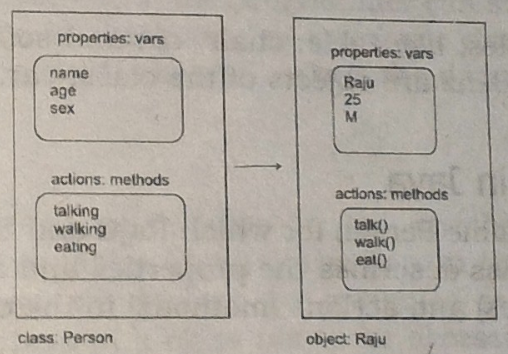
String name;

int age;

char sex;

Similarly, Raju can perform some actions like talking, walking, eating and sleeping. We may not write code for such actions in programming. But in programming, we can consider calculations and processing of data as actions. These actions are performed by methods (functions). So an object contains variables and methods.

It is possible that some objects may have similar properties and actions. Such objects belong to same category called a class. For example, not only Raju, but also Ravi, Sita, Vijay etc., person have same properties and actions. So they are all objects of same class, Person. Now observe that the Person will not physically but only Raju, Ravi, Sita etc. exist physically. This means, a class is group name and does not exist physically but objects exists physically.

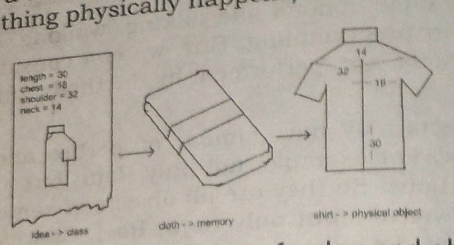


Person class and Raju object

We can use a class as a model for creating objects. To write a class, we can write all the characteristics of objects which should follow the class. These characteristics will guide us to create the objects. A class and its objects are almost the same with the difference that a class does not exist physically, while an object does. For example, if we say dog, it forms a picture into our mind with 4 legs, 2 ears, some length and height. This picture in our mind is class. If we tally this picture with physical thing around us, we can find Browny(name of dog) living in our house is satisfying these qualities. So Browny, which physically exists, is an object and not a class.

Let us take another example. Flower is a class but if we take Rose, Lily, Jasmine they are all objects flower does not exist physically but its objects, like Rose, Lily, Jasmine exist physically.

Let us consider another example. We want some shirts stitched by a tailor. First of all, the tailor takes the measurements and makes a plan for the shirt, according to the measurements. He may also draw a model shirt in his note book. This plan or model is called a class. Following this model, he stitches the shirts which we can wear. These shirts are called objects. To stitch the shirts, we need the material cloth. The cloth represents the memory allotted by the JVM for the objects. Remember, objects are created on heap memory by JVM at run time. This can be described with the following figure.



Creating of Class and Object

It is also possible to create several objects from the same class. An object cannot exist without a class. But a class can exist without any object. We can think that a class is a model and if it physically forms, then it becomes an object. So, an object is called instance (the thing physically happens) of a class.

Let use consider some more examples like table, chair, sofa are objects of the class, furniture. Similarly, Maruti, Santro and Benz are objects of the class car. Red, Blue, and Green are objects of the class Color.

Example of Class and Object in Java

class Student

{

//Properties of a Student – Variables

int roll;

String name;

//actions done by a student – Methods

void input()

{}

void output()

{}

}

Student Raju = new Student();

Here, Raju is an object of Student class. Object represents memory to store that actual data. Preceding new operator tells the JVM to allot necessary memory to the object of the Student class. Objects are created by JVM on heap memory.

**Encapsulation**

Encapsulation is a mechanism whee the data(variables) and the code(methods) that act on the data will bind together. For example, if we take a class, we write the variables and methods inside the class. Thus, class is binding them together. So, class is an example for encapsulation.

The variables and methods of a class are called members of the class. Generally, the variables in the class are declared by using a keyword private. This means the variables are not directly available to any other class. The methods of a class are declared as public. This means the methods can be called and sed from any where outside the class. To use the variables from outside, we should take the help of methods. There no other way of intereacting with the variables. This means outsiders do not know what variables are declared in a class, and what code is written in the method that is giving the result. Others can only use them and objtain the results.

Encapsulation thus protects the inner implementation of the members of the class from outside environment.

Encapsulation separates the members of a class from the members of another class. the reason is when objects are created, each object shares different memory and there will not be any overwriting of data. This gives an advantage to the programmer to use same names for the members of two different classes.

For example, a programmer can declare and use the following variables in two differenct classes i.e. Employee and Student.

int id;

String name;

Now in the variable id of Employee class, the programmer can store 1001 and in the id of Student class, he can store 12. Similarly, in the name of Employee class, he can store “Ram” and in the name of Student class he can store “Rohit”. Thus, the data of one class will not spoil the data of another class.

Example

class Student

{

//Properties of a Student – Variables

privateint roll;

private String name;

//actions done by a student – Methods

public void output()

{

System.out.println(“Roll Number = “ + roll);

System.out.println(“Name =” + name);

}

}

In this example, the variable roll and name are declared as private. There is no way to manipulate them. Only the public method output() can access them. By calling the method output() using Student class object, we can get the student name and roll name as student talk with us.

**Abstraction**

There may be a lot of data, a class may contains and the user does not need the entire data. The user requires only some part of the available data. In this case, we can hide the unnecessary data from the user and expose only that data that is required to the user. This is called abstraction.

A good example for abstraction is a car. Any car will have some parts like engine, radiator, mechanical and electrical equipment etc. The user of the car(driver) should know how to drive the car and does not require any knowledge of these parts. For example, driver is never bothered above how the engine is designed and the internal pars of the engine. This is why, the car manufacturers hide these pars from the driver in a separate panel, generally in the front.

The advantage of abstraction is that every user will get his own view of the data according to the requirements and will not get confused with unnecessary data. A bank clerk should see the customer details like account number, name and balance amount in the account. He should not be entitled to see the sensitive data like the staff salaries, profit or loss of the bank, interest amount paid by the bank and loans amount to be recovered etc. So such data can be abstracted from the clerk’s view. Whereas the bank manager is interested o know this data, it will be provided to the manager.

Example

class Bank

{

privateintaccno;

private String name;

private float balance;

private float loan;

private float profit;

public void displayToClerk()

{

System.out.println(“Account No =” + accno);

System.out.println(“Name =” +name);

System.out.println(“Balance =” + balance);

}

}

In this example, the displayToClerk() method is able to access and display only the accno, name and balance values. It cannot access profit and loan of the customer. This means the profit and loan data is hidden from the view of the bank clerk.

**Inheritance**

It creates new classes from existing classes, so that the new classes will acquire all the features of the existing classes is called inheritance. A good example for inheritance in nature is parents producing the children and children inheriting the qualities of the parents.

Let us create a class Address with some features (members i.e. variables and methods). If we feel another class Student wants almost the same features, then we can create the class Student from Address as follows.

class Address

{

protected String city;

protected String state;

protected long intpincode;

public void show()

{

}

}

class Student extends Address

{

int roll;

String name;

public void display()

{

}

}

Now, all the features of Address are available to Student. If an object to Student is created, it contains all the members of class Address and also its own members. Thu, the programmer can access and use all the members of both the classes Address and Student. Thus, class Student becomes more useful. This is called inheritance. The original class Address is called super class and the Student class is called the sub class.

There are three advantage of inheritance. First, we can create more useful classes needed by the application. Next, the process of creating the new classes is very easy, since they are built upon already existing classes. The last, but very important advantage is managing the code becomes easy, since the programmer creates several classes in a hierarchical manner, and segregates the code into several modules.

**Polymorphism**

The word Polymorphism cam from two Greek words Poly means many and morphos means forms. Thus, polymorphism represents the ability to assume several differernt forms. In programming, we can use a single variable to refer to objects of different types and thus, using that variable we can call the methods of the different objects. Thus a method call can perform different tasks depending on the type of the object.

Polymorphism provides flexibility in writing programs in such a way that the programmer uses same method call to perform different operations depending on the requirement.

Example

class One

{

public void calculate(int x)

{

System.out.println(“Square value =” + (x\*x));

}

}

class Two extends One

{

public void calculate(int x)

{

System.out.println(“Cube value =” + (x\*x\*x));

}

}

Let us take two obj1 and obj2 are objets of classes One and Two respectively. Let us create a reference variable for class One as

One ref;

If we use this ref to refer to the object of class One, as

ref=obj1;

and then call the calculate() method as follows

ref.calculate(2);

It will calculate the square value

If on the other hand, we use ref to the object of class Two, then ref will call the method of the class Two as

ref=obj2;

ref.calculate(2);

This will calculate the cube value. Now we have to observe that the same method call is performing two different tasks.

\* \* \*

**OR**

**Java OOPs Concepts**

Object Oriented Programming is a paradigm that provides many concepts such as inheritance, data binding, polymorphism etc.

Simula is considered as the first object-oriented programming language. The programming paradigm where everything is represented as an object, is known as truly object-oriented programming language.

Smalltalk is considered as the first truly object-oriented programming language.

**OOPs (Object Oriented Programming System)**



Object means a real word entity such as pen, chair, table etc. Object-Oriented Programming is a methodology or paradigm to design a program using classes and objects. It simplifies the software development and maintenance by providing some concepts:

* Object
* Class
* Inheritance
* Polymorphism
* Abstraction
* Encapsulation

**Object**

Any entity that has state and behaviour is known as an object. For example: chair, pen, table, keyboard, bike etc. It can be physical and logical.

**Class**

Collection of objects is called class. It is a logical entity.

**Inheritance**

When one object acquires all the properties and behaviours of parent object i.e. known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.

**Polymorphism**



When one task is performed by different ways i.e. known as polymorphism. For example: to convenes the customer differently, to draw something e.g. shape or rectangle etc.

In java, we use method overloading and method overriding to achieve polymorphism.

Another example can be to speak something e.g. cat speaks meaw, dog barks woof etc.

**Abstraction**

Hiding internal details and showing functionality is known as abstraction. For example: phone call, we don't know the internal processing.

In java, we use abstract class and interface to achieve abstraction.

**Encapsulation**



Binding (or wrapping) code and data together into a single unit is known as encapsulation. For example: capsule, it is wrapped with different medicines.

A java class is the example of encapsulation. Java bean is the fully encapsulated class because all the data members are private here.

**Advantage of OOPs over Procedure-oriented programming language**

* OOPs makes development and maintenance easier where as in Procedure-oriented programming language it is not easy to manage if code grows as project size grows.
* OOPs provides data hiding whereas in Procedure-oriented prgramming language a global data can be accessed from anywhere.
* OOPs provides ability to simulate real-world event much more effectively. We can provide the solution of real word problem if we are using the Object-Oriented Programming language.

|  |  |
| --- | --- |
| Global Data | Object Data |

**What is difference between object-oriented programming language and object-based programming language?**

|  |
| --- |
| Object based programming language follows all the features of OOPs except Inheritance. JavaScript and VBScript are examples of object based programming languages. |

**Difference between JDK, JRE and JVM**

### JVM

JVM (Java Virtual Machine) is an abstract machine. It is a specification that provides runtime environment in which java bytecode can be executed.

JVMs are available for many hardware and software platforms. JVM, JRE and JDK are platform dependent because configuration of each OS differs. But, Java is platform independent.

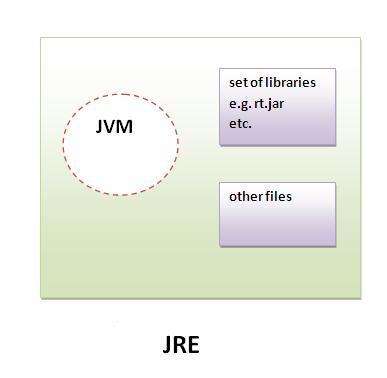
The JVM performs following main tasks:

* Loads code
* Verifies code
* Executes code
* Provides runtime environment

### JRE

JRE is an acronym for Java Runtime Environment.It is used to provide runtime environment.It is the implementation of JVM.It physically exists.It contains set of libraries + other files that JVM uses at runtime.

Implementation of JVMs are also actively released by other companies besides Sun Micro Systems.



**JDK**

JDK is an acronym for Java Development Kit.It physically exists.It contains JRE + development tools.

****

**JVM Architecture**

JVM (Java Virtual Machine) is an abstract machine. It is a specification that provides runtime environment in which java bytecode can be executed.

JVMs are available for many hardware and software platforms (i.e.JVM is plateform dependent).

### What is JVM?

It is:

1. **A specification** where working of Java Virtual Machine is specified. But implementation provider is independent to choose the algorithm. Its implementation has been provided by Sun and other companies.
2. **An implementation** Its implementation is known as JRE (Java Runtime Environment).
3. **Runtime Instance** Whenever you write java command on the command prompt to run the java class, and instance of JVM is created.

### What it does?

The JVM performs following operation:

* Loads code
* Verifies code
* Executes code
* Provides runtime environment

JVM provides definitions for the:

* Memory area
* Class file format
* Register set
* Garbage-collected heap
* Fatal error reporting etc

**Internal Architecture of JVM**

Let's understand the internal architecture of JVM. It contains classloader, memory area, execution engine etc.



### 1) Classloader:

Classloader is a subsystem of JVM that is used to load class files.

### 2) Class(Method) Area:

Class(Method) Area stores per-class structures such as the runtime constant pool, field and method data, the code for methods.

### 3) Heap:

It is the runtime data area in which objects are allocated.

### 4) Stack:

Java Stack stores frames.It holds local variables and partial results, and plays a part in method invocation and return.

Each thread has a private JVM stack, created at the same time as thread.

A new frame is created each time a method is invoked. A frame is destroyed when its method invocation completes.

### 5) Program Counter Register:

PC (program counter) register. It contains the address of the Java virtual machine instruction currently being executed.

### 6) Native Method Stack:

It contains all the native methods used in the application.

### 7) Execution Engine:

It contains:

**1) A virtual processor**

**2) Interpreter:**Readbytecode stream then execute the instructions.

**3) Just-In-Time(JIT) compiler:**It is used to improve the performance.JIT compiles parts of the byte code that have similar functionality at the same time, and hence reduces the amount of time needed for compilation.Here the term ?compiler? refers to a translator from the instruction set of a Java virtual machine (JVM) to the instruction set of a specific CPU.

|  |
| --- |
|  |
|  |

**2. Explain the naming convention of java.**

Java naming convention is a rule to follow as you decide what to name your identifiers such as class, package, variable, constant, method etc.

But, it is not forced to follow. So, it is known as convention not rule.

All the classes, interfaces, packages, methods and fields of java programming language are given according to java naming convention.

## Advantage of naming conventions in java

By using standard Java naming conventions, you make your code easier to read for yourself and for other programmers. Readability of Java program is very important. It indicates that less time is spent to figure out what the code does.

|  |  |
| --- | --- |
| **Name** | **Convention** |
| class name | should start with uppercase letter and be a noun e.g. String, Color, Button, System, Thread etc. |
| interface name | should start with uppercase letter and be an adjective e.g. Runnable, Remote, ActionListener etc. |
| method name | should start with lowercase letter and be a verb e.g. actionPerformed(), main(), print(), println() etc. |
| variable name | should start with lowercase letter e.g. firstName, orderNumber etc. |
| package name | should be in lowercase letter e.g. java, lang, sql, util etc. |
| constants name | should be in uppercase letter. e.g. RED, YELLOW, MAX\_PRIORITY etc. |

## CamelCase in java naming conventions

Java follows CamelCase syntax for naming the class, interface, method and variable.

If name is combined with two words, second word will start with uppercase letter always e.g. actionPerformed(), firstName, ActionEvent, ActionListener etc.

**Object and Class in Java**

In object-oriented programming technique, we design a program using objects and classes.

Object is the physical as well as logical entity whereas class is the logical entity only.

**Object in Java**



An entity that has state and behaviour is known as an object e.g. chair, bike, marker, pen, table, car etc. It can be physical or logical (tangible and intangible). The example of intangible object is banking system.

An object has three characteristics:

* state: represents data (value) of an object.
* behaviour: represents the behaviour (functionality) of an object such as deposit, withdraw etc.
* identity: Object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user. But,it is used internally by the JVM to identify each object uniquely.

|  |
| --- |
| For Example: Pen is an object. Its name is Reynolds, color is white etc. known as its state. It is used to write, so writing is its behavior. |

OObject is an instance of a class. Class is a template or blueprint from which objects are created. So object is the instance(result) of a class.

**Class in Java**

Aclass is a group of objects that has common properties. It is a template or blueprint from which objects are created.

A class in java can contain:

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

Syntax to declare a class:

class <class\_name>

{

 data member;

 method;

}

Simple Example of Object and Class

In this example, we have created a Student class that have two data members id and name. We are creating the object of the Student class by new keyword and printing the objects value.

class Student

{

 int id;//data member (also instance variable)

String name;//data member(also instance variable)

 public static void main(String args[])

{

Student s=new Student();//creating an object of Student

System.out.println(s.id);

System.out.println(s.name);

  }

}

**Output:**

0

null

**Instance variable in Java**

|  |
| --- |
| A variable that is created inside the class but outside the method, is known as instance variable.Instance variable doesn't get memory at compile time.It gets memory at runtime when object(instance) is created.That is why, it is known as instance variable. |

**Method in Java**

In java, a method is like function i.e. used to expose behaviour of an object.

Advantage of Method

* Code Reusability
* Code Optimization

**new keyword**

|  |
| --- |
| The new keyword is used to allocate memory at runtime. |

**Example of Object and class that maintains the records of students**

|  |
| --- |
| In this example, we are creating the two objects of Student class and initializing the value to these objects by invoking the assign() method on it. Here, we are displaying the state (data) of the objects by invoking the output() method. |

class Student

{

int roll;

String name;

void assign(int r, String n)

{

roll=r;

name=n;

}

 void output()

{

System.out.println(“Roll Number =” +roll);

System.out.println(“Name =” +name);

}

public static void main(String args[])

{

Student s1=new Student();

Student s2=new Student();

 s1.assign(101,"Ram");

s2.assign(102,"Shyam");

s1.output();

s2.output();

 }

}

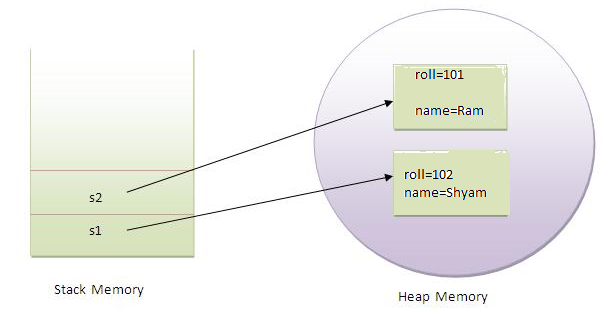
[**Output**](http://www.javatpoint.com/opr/test.jsp?filename=Student2)

Roll Number =101

Name=Ram

Roll Number=102

Name=Shyam



As you see in the above figure, object gets the memory in Heap area and reference variable refers to the object allocated in the Heap memory area. Here, s1 and s2 both are reference variables that refer to the objects allocated in memory.

**Anonymous object**

|  |
| --- |
| Anonymous simply means nameless. An object that has no reference is known as anonymous object. |
| If you have to use an object only once, anonymous object is a good approach. |

class Student

{

int roll;

String name;

 void output()

{

roll=101;

name=”Ram”;

System.out.println(“Roll Number =” +roll);

System.out.println(“Name =” +name);

}

public static void main(String args[])

{

new Student().output();

}

}

**Creating multiple objects by one type only**

|  |
| --- |
| We can create multiple objects by one type only as we do in case of primitives. |
| Student s1=new Student(),  s2=new Student();  **Example** |

class Student

{

int roll;

String name;

void assign(int r, String n)

{

roll=r;

name=n;

}

 void output()

{

System.out.println(“Roll Number =” +roll);

System.out.println(“Name =” +name);

}

public static void main(String args[])

{

Student s1=new Student(),  s2=new Student();

 s1.assign(101,"Ram");

s2.assign(102,"Shyam");

s1.output();

s2.output();

 }

}

**4. What do you mean by method overloading explain with an example.**

Method Overloading in Java

If a class have multiple methods by same name but different parameters, it is known as Method Overloading.

If we have to perform only one operation, having same name of the methods increases the readability of the program.

Suppose you have to perform addition of the given numbers but there can be any number of arguments, if you write the method such as add(int,int) for two parameters, and sum(int,int,int) for three parameters then it may be difficult for you as well as other programmers to understand the behaviour of the method because its name differs. So, we perform method overloading to figure out the program quickly.

**Advantage of method overloading?**

Method overloading increases the readability of the program.

**Different ways to overload the method**

|  |
| --- |
| There are two ways to overload the method in java |

1. By changing number of arguments
2. By changing the data type

In java, Method Overloading is not possible by changing the return type of the method.

1)Example of Method Overloading by changing the no. of arguments

In this example, we have created two overloaded methods, first sum method performs addition of two numbers and second sum method performs addition of three numbers.

class Calculation

{

void sum(int a,int b)

{

System.out.println(a+b);

}

void sum(int a,int b,int c)

{

System.out.println(a+b+c);

}

public static void main(String args[])

{

Calculation obj=new Calculation();

obj.sum(10,10,10);

obj.sum(20,20);

}

}

**Output:**

30

40

2)Example of Method Overloading by changing data type of argument

In this example, we have created two overloaded methods that differs in data type. The first sum method receives two integer arguments and second sum method receives two double arguments.

class Calculation

{

void sum(double a,double b)

{

System.out.println(a+b);

}

void sum(int a,int b,int c)

{

System.out.println(a+b+c);

}

public static void main(String args[])

{

Calculation obj=new Calculation();

obj.sum(10.5,10.5);

obj.sum(20,20,20);

}

}

**Output:**

21.0

60

**Why Method Overloading is not possible by changing the return type of method?**

In java, method overloading is not possible by changing the return type of the method because there may occur ambiguity. Let's see how ambiguity may occur:

because there was problem:

class Calculation

{

int sum(int a,int b)

{

System.out.println(a+b);

}

double sum(int a,int b)

{

System.out.println(a+b+c);

}

public static void main(String args[])

{

Calculation obj=new Calculation();

int result = obj.sum(20,20);

}

}

int result=obj.sum(20,20); //Here how can java determine which sum() method should be called

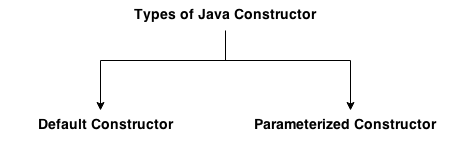
**5. What do you mean by constructor ? Describe different types of constructor available in java.**

**Constructor in java** is a *special type of method* having a name similar to class name without any return type. Constructor is normally used to initialize the object. Java constructor is *invoked automatically at the time of object creation*. It constructs the values i.e. provides data for the object that is why it is known as constructor.

**Types of java constructors**

There are two types of constructors:

1. Default constructor (no-argument constructor)
2. Parameterized constructor



**Default Constructor**

|  |
| --- |
| A constructor without parameter is known as default constructor. |

**Syntax of default constructor:**

<class\_name>()

{

}

**Example of default constructor**

|  |
| --- |
| In this example, we are creating the no-argument constructor in the Bike class. It will be invoked at the time of object creation. |

class Bike

{

Bike()

{

System.out.println("Bike is created");

}

public static void main(String args[])

{

Bike b=new Bike();

}

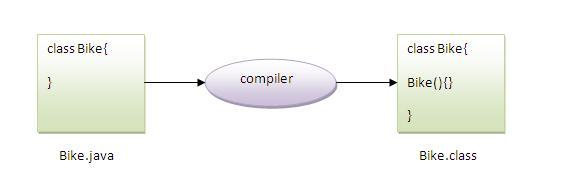
}

**Output:**

Bike is created

As soon as an object is created it automatically invokes the constructor and displays the above message but a constructor is normally used to initialise the instance variables and not to display message like Bike is created. Here, it is only example given to indicate the working methodology of constructor. But what happens when a class does not have any constructor and the answer is explained below.

**Rule: If there is no constructor in a class, compiler automatically creates a default constructor.**



**What is the purpose of default constructor?**

Default constructor provides the default values to the object like 0, null etc. depending on the type.

**Example of default constructor that displays the default values**

class Student

{

int roll;

String name;

   void display()

{

System.out.println(roll+" "+name);

}

public static void main(String args[])

{

Student s1=new Student();

Student s2=new Student();

s1.display();

s2.display();

}

}

**Output:**

0 null

0 null

**Explanation:**

In the above class, we are not creating any constructor, so compiler provides us a default constructor and 0 and null values are provided by default constructor to roll and name.

**Java parameterized constructor**

Aconstructor without any parameters is known as parameterized constructor. Parameterized constructor is used to provide different values to the distinct objects

In this example, we have created the constructor of Student class that have two parameters. We can have any number of parameters in the constructor.

class Student

{

int roll;

String name;

Student(intr,String nm)

{

roll = r;

name = nm;

}

void display()

{

System.out.println(roll+" "+name);

}

public static void main(String args[])

{

Student s1 = new Student(101,"Ram");

Student s2 = new Student4(102,"Shyam");

s1.display();

s2.display();

}

}

**Output**

101 Ram

102 Shyam

**Constructor Overloading in Java**

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 into account the number of parameters in the list and their type

Example of constructor overloading

class Student

{

int roll;

String name;

int age;

Student(intr,String nm)

{

roll = r;

name = nm;

}

Student(intr,String nm, intag)

{

roll = r;

name = nm;

age=ag;

}

void display()

{

System.out.println(roll+" "+name+ “ “ + age);

}

public static void main(String args[])

{

Student s1 = new Student(101,"Ram");

Student s2 = new Student4(102,"Shyam", 18);

s1.display();

s2.display();

}

}

**Output**

101 Ram 0

102 Shyam 18

**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. |

**5. Explain static key word.**

The static keyword in java is used for memory management mainly. We can apply java static keyword with variables, methods, blocks and nested class. The static keyword belongs to the class than instance of the class.

The static can be:

1. variable (also known as class variable)
2. method (also known as class method)
3. block
4. nested class

**1) Java static variable**

If we declare any variable as static, it is known static variable.

* The static variable can be used to refer the common property of all objects (that is not unique for each object) e.g. company name of employees,college name of students etc.
* The static variable gets memory only once in class area at the time of class loading.

**Advantage of static variable**

It makes program memory efficient i.e. it saves memory. In order to get a clear idea on static variable, we have to understand problem without static variable which is given below.

class Student

{

int rollno;

String name;

String college="ITS";

}

Suppose there are 500 students in my college, now all instance data members will get memory each time when object is created.All student have its unique rollno and name so instance data member is good.Here, college refers to the common property of all objects.If we make it static,this field will get memory only once i.e. Java static property is shared to all objects.

Example of static variable

//Program of static variable

class Student

{

int rollno;

String name;

static String college ="ITS";

Student(int r,String n)

{

rollno = r;

name = n;

}

void display ()

{

System.out.println(rollno+" "+name+" "+college);

}

public static void main(String args[])

{

Student s1 = new Student(101,"Ram");

Student s2 = new Student(102,"Shyam");

s1.display();

s2.display();

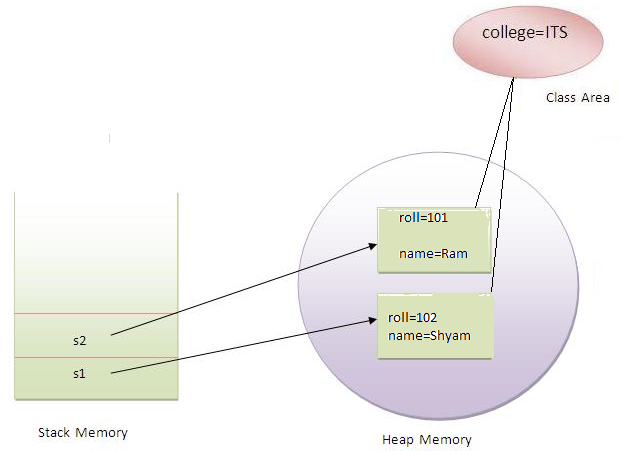
 }

}

Output:

101 Ram ITS

102 Shyam ITS



Program of counter by static variable

|  |
| --- |
| As we have mentioned above, static variable will get the memory only once, if any object changes the value of the static variable, it will retain its value. |

class Counter

{

static int count=0;//will get memory only once and retain its value

Counter()

{

count++;

System.out.println(count);

}

public static void main(String args[])

{

Counter c1=new Counter();

Counter c2=new Counter();

Counter c3=new Counter();

}

}

**Output:**

1

2

3

**2) Java static method**

If we 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 instance of a class.
* static method can access static data member and can change the value of it.

Example of static method

//Program of changing the common property of all objects(static field).

class Student

{

int rollno;

String name;

   static String college = "ITS";

        static void change()

{

college = "BBDIT";

}

   Student(int r, String n)

{

rollno = r;

name = n;

    }

   void display ()

{

System.out.println(rollno+" "+name+" "+college);

}

   public static void main(String args[])

{

Student.change();

   Student s1 = new Student (111,"Karan");

Student s2 = new Student (222,"Aryan");

Student s3 = new Student (333,"Sonoo");

   s1.display();

s2.display();

s3.display();

}

}

Output:

111 Karan BBDIT

222 Aryan BBDIT

1. noo BBDIT

Another example of static method that performs normal calculation

//Program to get cube of a given number by static method

class Calculate

{

static int cube(int x)

{

return x\*x\*x;

}

public static void main(String args[])

{

int result=Calculate.cube(5);

System.out.println(result);

}

}

Output:

125

**Restrictions for static method**

|  |
| --- |
| There are two main restrictions for the static method. They are:   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. |

1. class A{
2. int a=40;//non static
4. public static void main(String args[]){
5. System.out.println(a);
6. }
7. }

**Output:**

Compile Time Error

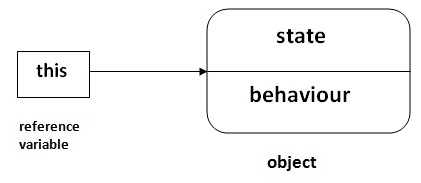
**this keyword in java**

In java, this is a **reference variable** that refers to the current object.

**Usage of java this keyword**

Here is given the 6 usage of java 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 keyword can be used to invoke current class method (implicitly)
4. this can be passed as an argument in the method call.
5. this can be passed as argument in the constructor call.
6. this keyword can also be used to return the current class instance.



**1) The this keyword can be used to refer current class instance variable.**

If there is ambiguity between the instance variable and parameter, this keyword resolves the problem of ambiguity.

|  |
| --- |
|  |

**Understanding the problem without this keyword**

Let's understand the problem if we don't use this keyword by the example given below:

|  |
| --- |
|  |

class Student

{

int id;

String name;

Student10(int id,String name)

{

id = id;

name = name;

}

void display()

{

System.out.println(id+" "+name);

}

public static void main(String args[])

{

Student s1 = new Student(111,"Karan");

Student s2 = new Student(321,"Aryan");

s1.display();

s2.display();

}

}

**Output:**

0 null

0 null

|  |
| --- |
| In the above example, parameter (formal arguments) and instance variables are same that is why we are using this keyword to distinguish between local variable and instance variable. |

**Solution of the above problem by this keyword**

class Student

{

int id;

String name;

Student10(int id,String name)

{

this.id = id;

this.name = name;

}

void display()

{

System.out.println(id+" "+name);

}

public static void main(String args[])

{

Student s1 = new Student(111,"Karan");

Student s2 = new Student(321,"Aryan");

s1.display();

s2.display();

}

}

**Output:**

111 Karan

321 Aryan

|  |
| --- |
| If local variables(formal arguments) and instance variables are different, there is no need to use this keyword like in the following program: |

**2) this() can be used to invoked current class constructor.**

The this() constructor call can be used to invoke the current class constructor (constructor chaining). This approach is better if you have many constructors in the class and want to reuse that constructor.

//Program of this() constructor call (constructor chaining)

class Student

{

int id;

String name;

Student()

{

System.out.println("default constructor is invoked");

}

Student(int id,String name)

{

this ();//it is used to invoked current class constructor.

this.id = id;

this.name = name;

}

void display()

{

System.out.println(id+" "+name);

}

public static void main(String args[])

{

Student s1 = new Student(111,"karan");

Student s2 = new Student(222,"Aryan");

s1.display();

s2.display();

}

}

**Output:**

default constructor is invoked

default constructor is invoked

111 Karan

222 Aryan

**Where to use this() constructor call?**

The this() constructor call should be used to reuse the constructor in the constructor. It maintains the chain between the constructors i.e. it is used for constructor chaining. Let's see the example given below that displays the actual use of this keyword

class Student

{

int id;

String name;

String city;

Student(int id,String name)

{

this.id = id;

this.name = name;

}

Student(int id,String name,String city)

{

this(id,name);//now no need to initialize id and name

this.city=city;

}

void display()

{

System.out.println(id+" "+name+" "+city);

}

public static void main(String args[])

{

Student s1 = new Student(111,"karan");

Student s2 = new Student(222,"Aryan","delhi");

s1.display();

s2.display();

}

}

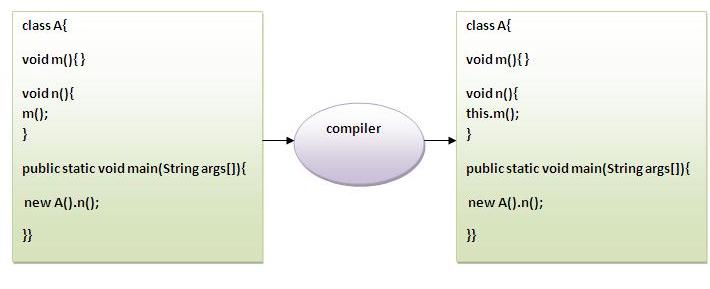
**Output:**

111 Karan null

222 Aryan delhi

**Rule: Call to this() must be the first statement in constructor.**

**3)The this keyword can be used to invoke current class method (implicitly).**

You may invoke the method of the current class by using the this keyword. If you don't use the this keyword, compiler automatically adds this keyword while invoking the method. Let's see the example

class S

{

void m()

{

System.out.println("method is invoked");

}

void n()

{

this.m();//no need because compiler does it for you.

}

void p()

{

n();//complier will add this to invoke n() method as this.n()

}

public static void main(String args[])

{

S s1 = new S();

s1.p();

}

}

**Output:**

method is invoked

**4) this keyword can be passed as an argument in the method.**

this keyword can also be passed as an argument in the method. It is mainly used in the event handling. Let's see the example

class S

{

void m(S obj)

{

System.out.println("method is invoked");

}

void n()

{

m(this);//no need because compiler does it for you.

}

void p()

{

n();//complier will add this to invoke n() method as this.n()

}

public static void main(String args[])

{

S s1 = new S();

s1.p();

}

}

**Output:**

method is invoked

**Application of this that can be passed as an argument:**

In event handling (or) in a situation where we have to provide reference of a class to another one

|  |
| --- |
|  |

**5) The this keyword can be passed as argument in the constructor call.**

We can pass this keyword as parameter in the constructor also. It is useful if we have to use one object in multiple classes.

**Example**

class B

{

A obj;

B(A obj)

{

this.obj=obj;

}

void display()

{

System.out.println(obj.data);//using data member of A class

}

}

class A

{

int data=10;

A()

{

B b=new B(this);

b.display();

}

public static void main(String args[])

{

A a=new A();

}

}

**Output:**

10

**6) The this keyword can be used to return current class instance.**

We can return the this keyword as an statement from the method. In such case, return type of the method must be the class type (non-primitive). Let's see the example.

**Syntax of this that can be returned as a statement**

return\_type method\_name()

{

return this;

}

**Example of this keyword that you return as a statement from the method**

class A

{

A getA()

{

return this;

}

void msg()

{

System.out.println("Hello java");

}

}

class Test

{

public static void main(String args[])

{

new A().getA().msg();

}

}

**Output:**

Hello java

**What do you mean by inheritance ? Describe the different types of inheritance supported by java.**

**Inheritance in java** is a mechanism in which one object acquires all the properties and behaviours of parent object.The idea behind inheritance in java is that we can create new classes that are built upon existing classes. When we inherit from an existing class, we can reuse methods and fields of parent class, and we can add new methods and fields also. So, deriving new classes from existing classes such that new classes acquire all the features of existing classes is called inheritance.

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

{

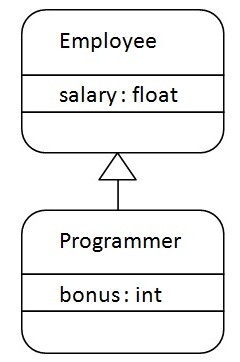
   //methods and fields

}

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

In the terminology of Java, a class that is inherited is called a super class. The new class is called a subclass.

Understanding the simple example of inheritance



As displayed in the above figure, Programmer is the subclass and Employee is the superclass. Relationship between two classes is **Programmer IS-A Employee**.It means that Programmer is a type of Employee.

**class** Employee

{

**float** salary=40000;

}

**class** Programmer **extends** Employee

{

**int** bonus=10000;

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

{

Programmer p=**new** Programmer();

System.out.println("Programmer salary is:"+p.salary);

System.out.println("Bonus of Programmer is:"+p.bonus);

}

}

**Output**

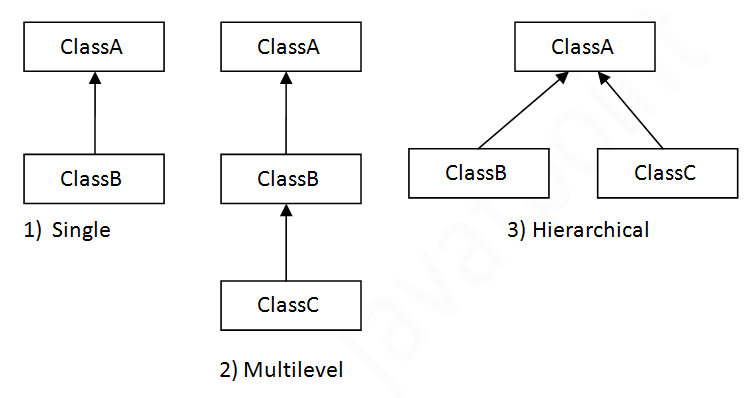
Programmer salary is:40000.0

Bonus of programmer is:10000

In the above example, Programmer object can access the field of own class as well as of Employee class i.e. code reusability.

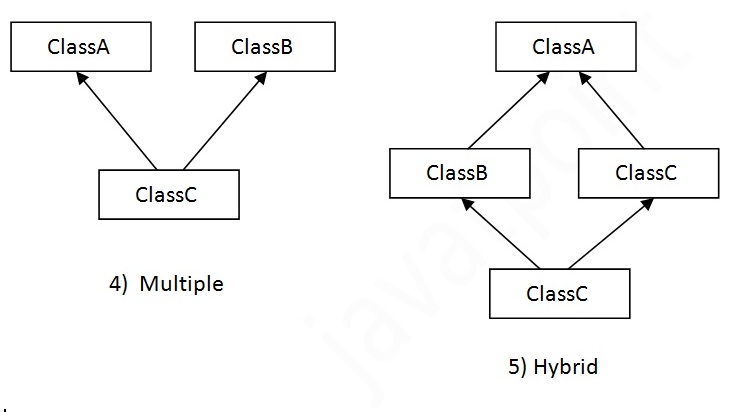
**Types of inheritance in java**

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 not supported through class. They are supported through interface only.

When a class extends multiple classes then it is known as multiple inheritance. For Example:



**Why multiple inheritanceis 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 we 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 we inherit 2 classes. So whether we have same method or different, there will be compile time error now.

**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?

}

}

**Output**

Compile Time Error

**What do you mean by Aggregation in Java, explain with suitable example.**

If a class have an entity reference, it is known as Aggregation. Aggregation represents HAS-A relationship.

Consider a situation, Employee object contains many information such as id, name, emailId etc. It contains one more object named address, which contains its own informations such as city, state, country, zipcode etc. as given below.

class Employee

{

int id;

String name;

Address address;//Address is a class

...

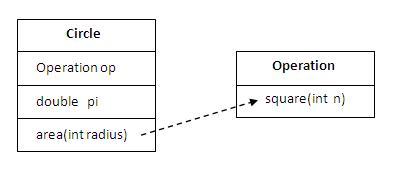
}

In such case, Employee has an entity reference address, so relationship is Employee HAS-A address.

**Why use Aggregation?**

* For Code Reusability.

Simple Example of Aggregation



In this example, we have created the reference of Operation class in the Circle class.

class Operation

{

int square(int n)

{

return n\*n;

}

}

class Circle

{

Operation op;//aggregation

double pi=3.14;

double area(int radius)

{

op=new Operation();

intrsquare=op.square(radius);//code reusability (i.e. delegates the method call).

return pi\*rsquare;

}

public static void main(String args[])

{

Circle c=new Circle();

double result=c.area(5);

System.out.println(result);

}

}

**Output:**

78.5

**When use Aggregation?**

* Code reuse is also best achieved by aggregation when there is no is-a relationship.
* Inheritance should be used only if the relationship is-a is maintained throughout the lifetime of the objects involved; otherwise, aggregation is the best choice.

Understanding meaningful example of Aggregation

In this example, Employee has an object of Address, address object contains its own informationsuch as city, state, country etc. In such case relationship is Employee HAS-A address.

**Address.java**

public class Address

{

String city, state,country;

public Address(String city, String state, String country)

{

this.city = city;

this.state = state;

this.country = country;

}

}

**Emp.java**

public class Emp

{

int id;

String name;

Address address;

publicEmp(int id, String name,Address address)

{

this.id = id;

this.name = name;

this.address=address;

}

void display()

{

System.out.println(id+" "+name);

System.out.println(address.city+" "+address.state+" "+address.country);

}

public static void main(String[] args)

{

Address address1=new Address("Sambalpur","Odisha","India");

Address address2=new Address("Bhubaneswar","Odisha","India");

Emp e=new Emp(111,"varun",address1);

Emp e2=new Emp(112,"arun",address2);

e.display();

e2.display();

}

}

Output:

111 varun

SambalpurOdishaIndia

112 arun

Bhubaneswar Odisha India

**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).

**Example of method overriding**

In this example, we have defined the run method in the subclass as defined in the parent class but it has some specific implementation. The name and parameter of the method is same and there is IS-A relationship between the classes, so there is 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();

}

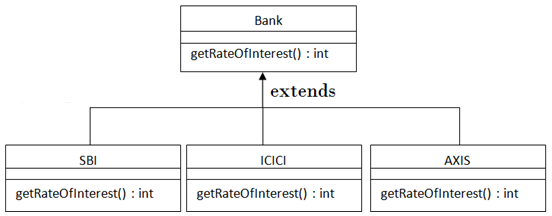
}

**Output:**

Bike is running safely

**Real example of Java Method Overriding**

Consider a scenario, Bank is a class that provides functionality to get rate of interest. But, rate of interest varies according to banks. For example, SBI, ICICI and AXIS banks could provide 8%, 7% and 9% rate of interest.



class Bank

{

int getRateOfInterest()

{

return 0;

}

}

class SBI extends Bank

{

int getRateOfInterest()

{

return 8;

}

}

class ICICI extends Bank

{

int getRateOfInterest()

{

return 7;

}

}

class AXIS extends Bank

{

int getRateOfInterest()

{

return 9;

}

}

class Test

{

public static void main(String args[])

{

SBI s=new SBI();

ICICI i=new ICICI();

AXIS a=new AXIS();

System.out.println("SBI Rate of Interest: "+s.getRateOfInterest());

System.out.println("ICICI Rate of Interest: "+i.getRateOfInterest());

System.out.println("AXIS Rate of Interest: "+a.getRateOfInterest());

}

}

**Output:**

SBI Rate of Interest: 8

ICICI Rate of Interest: 7

AXIS Rate of Interest: 9

**Can we override static method?**

No, static method cannot be overridden. It can be proved by runtime polymorphism, so we will learn it later.

**Why we cannot override static method?**

because static method is bound with class whereas instance method is bound with object. Static belongs to class area and instance belongs to heap area.

**Can we override java main method?**

No, because main is a static method.

**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. |

**Method Overloading example**

class OverloadingExample

{

static int add(int a,int b)

{

return a+b;

}

static int add(int a,int b,int c)

{

return a+b+c;

}

}

**Method Overriding example**

class Animal

{

void eat()

{

System.out.println("eating...");

}

}

class Dog extends Animal

{

void eat()

{

System.out.println("eating bread...");

}

}

**super keyword in java**

The **super** keyword in java is a reference variable that is used to refer immediate parent class object. Whenever we create the instance of subclass, an instance of parent class is created implicitly i.e. referred by super reference 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** Bike **extends** Vehicle

{

**int** speed=100;

**void** display()

{

System.out.println(speed);//will print speed of Bike

}

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

{

Bike b=**new** Bike();

b.display();

}

}

Output:100

In the above example Vehicle and Bike both class have a common property speed. Instance variable of current class is referred by instance by default, but I have to refer parent class instance variable that is why we use super keyword to distinguish between parent class instance variable and current class instance variable.

***Solution by super keyword***

//example of super keyword

**class** Vehicle

{

**int** speed=50;

}

**class** Bike **extends** Vehicle

{

**int** speed=100;

**void** display()

{

 System.out.println(**super**.speed);//will print speed of Vehicle now

 }

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

{

Bike b=**new** Bike();

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** Bike **extends** Vehicle

{

Bike()

{

**super**(); //will invoke parent class constructor

 System.out.println("Bike is created");

 }

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

{

Bike b=**new** Bike();

         }

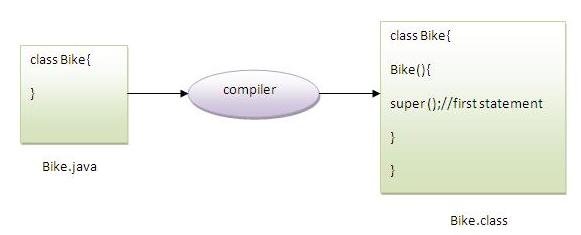
}

**Output:**

Vehicle is created

Bike is created

**Note: super() is added in *each class constructor automatically by compiler.***



As we know well that default constructor is provided by compiler automatically but it also adds super() for the first statement.If you are creating your own constructor and you don't have either this() or super() as the first statement, compiler will provide super() as the first statement of the constructor.

**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** Student **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[])

{

Student s=**new** Student();

s.display();

}

}

Output:

welcome to java

welcome

In the above example Student and Person both classes have message() method if we call message() method from Student class, it will call the message() method of Student class not of Person class because priority is given to local.In case there is no method in subclass as parent, there is no need to use super.

**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.

Java Final keyword

* Stop value change
* Stop method overriding
* Stop inheritance

**1) Java final variable**

If we make any variable as final, we 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** Bike

{

**final** **int** speedlimit=90; //final variable

**void** run()

{

speedlimit=400;

}

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

{

Bike obj=**new**  Bike();

obj.run();

 }

}

Output:

Compile Time Error

**2) Java final method**

If we make any method as final, we 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 we make any class as final, we cannot extend it.

Example of final class

**final** **class** Bike

{

}

**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

**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** Honda **extends** Bike

{

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

{

**new** Honda().run();

}

}

**Output:**

running...

Q) What is blank or uninitialized final variable?

A final variable that is not initialized at the time of declaration is known as blank final variable.

If you want to create a variable that is initialized at the time of creating object and once initialized may not be changed, it is useful. For example PAN CARD number of an employee.

It can be initialized only in constructor.

Example of blank final variable

**class** Student

{

**int** id;

String name;

**final** String PAN\_CARD\_NUMBER;

...

}

**Can we initialize blank final variable?**

Yes, but only in constructor. For example:

**class** Bike

{

**final** **int** speedlimit;//blank final variable

 Bike()

{

   speedlimit=70;

 System.out.println(speedlimit);

 }

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

{

**new** Bike();

 }

}

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=Bike10)

**Output:70**

**static blank final variable**

A static final variable that is not initialized at the time of declaration is known as static blank final variable. It can be initialized only in static block.

Example of static blank final variable

**class** A

{

**static** **final** **int** data;//static blank final variable

**static**

{

data=50;

}

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

{

     System.out.println(A.data);

  }

}

**What is final parameter?**

If you declare any parameter as final, you cannot change the value of it.

**class** Bike

{

**int** cube(**final** **int** n)

{

 n=n+2;//can't be changed as n is final

 n\*n\*n;

}

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

{

 Bike11 b=**new** Bike11();

 b.cube(5);

 }

}

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=Bike11)

Output:Compile Time Error

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).

Before explaining java abstract class, let's understand the abstraction in java first.

**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 the user focus on what the object does instead of how it does it.

**Ways to achieve Abstraction**

There are two ways to achieve abstraction in java

* Abstract class (0 to 100%)
* 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 One

{

}

**abstract method**

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

**Example**

abstract void display();//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 One

{

abstract void display();

}

class Two extends One

{

void display()

{

System.out.println("Hello..");

}

public static void main(String args[])

{

Two obj = new Two();

obj.display();

}

}

Understanding the real scenario of abstract class

In this example, Shape is the abstract class, its implementation is provided by the Rectangle and Circle classes. Mostly, we don't know about the implementation class (i.e. hidden to the end user) and object of the implementation class is provided by the factory method.

In this example, if you create the instance of Rectangle class, draw() method of Rectangle class will be invoked.

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 ShapeTest

{

public static void main(String args[])

{

Rectangle r = new Rectangle();

Circle r= new Circle();

r.draw();

c.draw();

}

}

Another example of abstract class in java

abstract class Bank

{

abstract int getRateOfInterest();

}

class SBI extends Bank

{

int getRateOfInterest()

{

return 7;

}

}

class PNB extends Bank

{

int getRateOfInterest()

{

return 7;

}

}

class TestBank

{

public static void main(String args[])

{

Bank b=new SBI();//if object is PNB, method of PNB will be invoked

int interest=b.getRateOfInterest();

System.out.println("Rate of Interest is: "+interest+" %");

}

}

**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 TestBike

{

public static void main(String args[])

{

Bike obj = new Honda();

obj.run();

obj.changeGear();

}

}

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

class Bike

{

abstract void run();

}

**output**

compile time error

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

The abstract class can also be used to provide some implementation of the interface. In such case, the end user may not be forced to override all the methods of the interface.

interface A

{

void a();

void b();

void c();

void d();

}

abstract class B implements A

{

public void c()

{

System.out.println("I am C");

}

}

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");

}

}

class Test

{

public static void main(String args[])

{

A obj=new M();

obj.a();

obj.b();

obj.c();

obj.d();

}

}

**Interface in Java**

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.

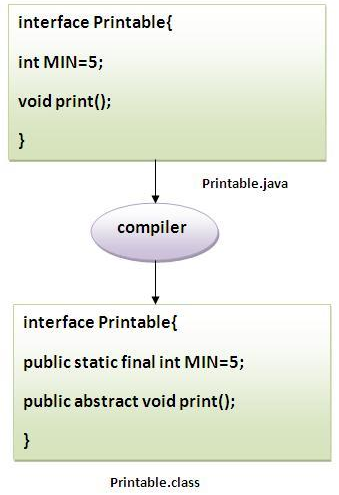
**Why use Java interface?**

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.

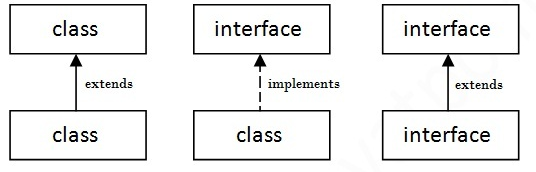
The java compiler adds public and abstract keywords before the interface method and public, static and final keywords before data members.

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.



interface printable

{

void print();

}

class One implements printable

{

public void print()

{

System.out.println("Hello");

}

public static void main(String args[])

{

One obj = new One();

obj.print();

}

}

**Multiple inheritance in Java by interface**

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

interface Printable

{

void print();

}

interface Showable

{

void show();

}

class One implements Printable, Showable

{

public void print()

{

System.out.println("Hello");

}

public void show()

{

System.out.println("Welcome");

}

public static void main(String args[])

{

One obj = new One();

obj.print();

obj.show();

}

}

**Multiple inheritance is not supported through class in java but it is possible by interface, why?**

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 TestTnterface implements Printable, Showable

{

public void print()

{

System.out.println("Hello");

}

public static void main(String args[])

{

TestTnterface obj = new TestTnterface();

obj.print();

}

}

As you can see in the above example, Printable and Showable interface have same methods but its implementation is provided by class TestTnterface, 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 Testinterface implements Showable

{

public void print()

{

System.out.println("Hello");

}

public void show()

{

System.out.println("Welcome");

}

public static void main(String args[])

{

Testinterface obj = new Testinterface();

obj.print();

obj.show();

}

}

Exception Handling in Java

Exception Handling

Advantage of Exception Handling

Hierarchy of Exception classes

Types of Exception

Scenarios where exception may occur

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.

In this page, we will learn about java exception, its type and the difference between checked and unchecked exceptions.

What is exception

Dictionary Meaning: Exception is an abnormal condition.

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

What is exception handling

Exception Handling is a mechanism to handle runtime errors such as ClassNotFound, IO, SQL, Remote etc.

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. Let's take a scenario:

statement 1;

statement 2;

statement 3;

statement 4;

statement 5;//exception occurs

statement 6;

statement 7;

statement 8;

statement 9;

statement 10;

Suppose there is 10 statements in your program and there occurs an exception at statement 5, rest of the code will not be executed i.e. statement 6 to 10 will not run. If we perform exception handling, rest of the statement will be executed. That is why we use exception handling in java.

Do You Know ?

What is the difference between checked and unchecked exceptions ?

What happens behind the code int data=50/0; ?

Why use multiple catch block ?

Is there any possibility when finally block is not executed ?

What is exception propagation ?

What is the difference between throw and throws keyword ?

What are the 4 rules for using exception handling with method overriding ?

Hierarchy of Java Exception classes

hierarchy of 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:

Checked Exception

Unchecked Exception

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.

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.

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.

String s=null;

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.

String s="abc";

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:

int a[]=new int[5];

a[10]=50; //ArrayIndexOutOfBoundsException

Java Exception Handling Keywords

There are 5 keywords used in java exception handling.

try

catch

finally

throw

throws

Java try-catch

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

try{

//code that may throw exception

}catch(Exception\_class\_Name ref){}

Syntax of try-finally block

try{

//code that may throw exception

}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...");

}

}

Test it Now

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...");

}

}

Test it Now

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.

Internal working of java try-catch block

internal working of try-catch block

The JVM firstly checks whether the exception is handled or not. If exception is not handled, JVM provides a default exception handler that performs the following tasks:

Prints out exception description.

Prints the stack trace (Hierarchy of methods where the exception occurred).

Causes the program to terminate.

But if exception is handled by the application programmer, normal flow of the application is maintained i.e. rest of the code is executed.

Java catch multiple exceptions

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.

public class TestMultipleCatchBlock{

public static void main(String args[]){

try{

int a[]=new int[5];

a[5]=30/0;

}

catch(ArithmeticException e){System.out.println("task1 is completed");}

catch(ArrayIndexOutOfBoundsException e){System.out.println("task 2 completed");}

catch(Exception e){System.out.println("common task completed");}

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

}

}

Test it Now

Output:task1 completed

rest of the code...

Rule: At a time only one Exception is occured and at a time only one catch block is executed.

Rule: All catch blocks must be ordered from most specific to most general i.e. catch for ArithmeticException must come before catch for Exception .

class TestMultipleCatchBlock1{

public static void main(String args[]){

try{

int a[]=new int[5];

a[5]=30/0;

}

catch(Exception e){System.out.println("common task completed");}

catch(ArithmeticException e){System.out.println("task1 is completed");}

catch(ArrayIndexOutOfBoundsException e){System.out.println("task 2 completed");}

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

}

}

Test it Now

Output:

Compile-time error

Java Nested try block

The try block within a try block is known as nested try block in java.

Why use nested try block

Sometimes a situation may arise where a part of a block may cause one error and the entire block itself may cause another error. In such cases, exception handlers have to be nested.

Syntax:

....

try

{

statement 1;

statement 2;

try

{

statement 1;

statement 2;

}

catch(Exception e)

{

}

}

catch(Exception e)

{

}

....

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..");

}

}

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 must be followed by try or catch block.

java finally

Note: If you don't handle exception, before terminating the program, JVM executes finally block(if any).

Why use java finally

Finally block in java can be used to put "cleanup" code such as closing a file, closing connection etc.

Usage of Java finally

Let's see the different cases where java finally block can be used.

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...");

}

}

Test it Now

Output:5

finally block is always executed

rest of the code...

Case 2

Let's see the java finally example where exception occurs and not handled.

class TestFinallyBlock1{

public static void main(String args[]){

try{

int data=25/0;

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...");

}

}

Test it Now

Output:finally block is always executed

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

Case 3

Let's see the java finally example where exception occurs and handled.

public class TestFinallyBlock2{

public static void main(String args[]){

try{

int data=25/0;

System.out.println(data);

}

catch(ArithmeticException e){System.out.println(e);}

finally{System.out.println("finally block is always executed");}

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

}

}

Test it Now

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

finally block is always executed

rest of the code...

Rule: For each try block there can be zero or more catch blocks, but only one finally block.

Note: The finally block will not be executed if program exits(either by calling System.exit() or by causing a fatal error that causes the process to abort).

Java throw exception

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.

throw exception;

Let's see the example of throw IOException.

throw new IOException("sorry device error);

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.

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...");

}

}

Test it Now

Output:

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

Java Exception propagation

An exception is first thrown from the top of the stack and if it is not caught, it drops down the call stack to the previous method,If not caught there, the exception again drops down to the previous method, and so on until they are caught or until they reach the very bottom of the call stack.This is called exception propagation.

Rule: By default Unchecked Exceptions are forwarded in calling chain (propagated).

Program of Exception Propagation

class TestExceptionPropagation1{

void m(){

int data=50/0;

}

void n(){

m();

}

void p(){

try{

n();

}catch(Exception e){System.out.println("exception handled");}

}

public static void main(String args[]){

TestExceptionPropagation1 obj=new TestExceptionPropagation1();

obj.p();

System.out.println("normal flow...");

}

}

Test it Now

Output:exception handled

normal flow...

exception propagation

In the above example exception occurs in m() method where it is not handled,so it is propagated to previous n() method where it is not handled, again it is propagated to p() method where exception is handled.

Exception can be handled in any method in call stack either in main() method,p() method,n() method or m() method.

Rule: By default, Checked Exceptions are not forwarded in calling chain (propagated).

Program which describes that checked exceptions are not propagated

class TestExceptionPropagation2{

void m(){

throw new java.io.IOException("device error");//checked exception

}

void n(){

m();

}

void p(){

try{

n();

}catch(Exception e){System.out.println("exception handeled");}

}

public static void main(String args[]){

TestExceptionPropagation2 obj=new TestExceptionPropagation2();

obj.p();

System.out.println("normal flow");

}

}

Test it Now

Output:Compile Time Error

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

return\_type method\_name() throws exception\_class\_name{

//method code

}

Which exception should be declared

Ans) checked exception only, because:

unchecked Exception: under your control so correct your code.

error: beyond your control e.g. you are unable to do anything if there occurs VirtualMachineError or StackOverflowError.

Advantage of Java throws keyword

Now Checked Exception can be propagated (forwarded in call stack).

It provides information to the caller of the method about the exception.

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...");

}

}

Test it Now

Output:

exception handled

normal flow...

Rule: If you are calling a method that declares an exception, you must either caught or declare the exception.

There are two cases:

Case1:You caught the exception i.e. handle the exception using try/catch.

Case2:You declare the exception i.e. specifying throws with the method.

Case1: You handle the exception

In case you handle the exception, the code will be executed fine whether exception occurs during the program or not.

import java.io.\*;

class M{

void method()throws IOException{

throw new IOException("device error");

}

}

public class Testthrows2{

public static void main(String args[]){

try{

M m=new M();

m.method();

}catch(Exception e){System.out.println("exception handled");}

System.out.println("normal flow...");

}

}

Test it Now

Output:exception handled

normal flow...

Case2: You declare the exception

A)In case you declare the exception, if exception does not occur, the code will be executed fine.

B)In case you declare the exception if exception occures, an exception will be thrown at runtime because throws does not handle the exception.

A)Program if exception does not occur

import java.io.\*;

class M{

void method()throws IOException{

System.out.println("device operation performed");

}

}

class Testthrows3{

public static void main(String args[])throws IOException{//declare exception

M m=new M();

m.method();

System.out.println("normal flow...");

}

}

Test it Now

Output:device operation performed

normal flow...

B)Program if exception occurs

import java.io.\*;

class M{

void method()throws IOException{

throw new IOException("device error");

}

}

class Testthrows4{

public static void main(String args[])throws IOException{//declare exception

M m=new M();

m.method();

System.out.println("normal flow...");

}

}

Test it Now

Output:Runtime Exception

Difference between throw and throws

Click me for details

Que) Can we rethrow an exception?

Yes, by throwing same exception in catch block.

Difference between throw and throws in Java

There are many differences between throw and throws keywords. A list of differences between throw and throws are given below:

No. throw throws

1) Java throw keyword is used to explicitly throw an exception. Java throws keyword is used to declare an exception.

2) Checked exception cannot be propagated using throw only. Checked exception can be propagated with throws.

3) Throw is followed by an instance. Throws is followed by class.

4) Throw is used within the method. Throws is used with the method signature.

5) You cannot throw multiple exceptions. You can declare multiple exceptions e.g.

public void method()throws IOException,SQLException.

Java throw example

void m(){

throw new ArithmeticException("sorry");

}

Java throws example

void m()throws ArithmeticException{

//method code

}

Java throw and throws example

void m()throws ArithmeticException{

throw new ArithmeticException("sorry");

}

Difference between final, finally and finalize

There are many differences between final, finally and finalize. A list of differences between final, finally and finalize are given below:

No. final finally finalize

1) Final is used to apply restrictions on class, method and variable. Final class can't be inherited, final method can't be overridden and final variable value can't be changed. Finally is used to place important code, it will be executed whether exception is handled or not. Finalize is used to perform clean up processing just before object is garbage collected.

2) Final is a keyword. Finally is a block. Finalize is a method.

Java final example

class FinalExample{

public static void main(String[] args){

final int x=100;

x=200;//Compile Time Error

}}

Java finally example

class FinallyExample{

public static void main(String[] args){

try{

int x=300;

}catch(Exception e){System.out.println(e);}

finally{System.out.println("finally block is executed");}

}}

Java finalize example

class FinalizeExample{

public void finalize(){System.out.println("finalize called");}

public static void main(String[] args){

FinalizeExample f1=new FinalizeExample();

FinalizeExample f2=new FinalizeExample();

f1=null;

f2=null;

System.gc();

}}

ExceptionHandling with MethodOverriding in Java

There are many rules if we talk about methodoverriding with exception handling. The Rules are as follows:

If the superclass method does not declare an exception

If the superclass method does not declare an exception, subclass overridden method cannot declare the checked exception but it can declare unchecked exception.

If the superclass method declares an exception

If the superclass method declares an exception, subclass overridden method can declare same, subclass exception or no exception but cannot declare parent exception.

If the superclass method does not declare an exception

1) Rule: If the superclass method does not declare an exception, subclass overridden method cannot declare the checked exception.

import java.io.\*;

class Parent{

void msg(){System.out.println("parent");}

}

class TestExceptionChild extends Parent{

void msg()throws IOException{

System.out.println("TestExceptionChild");

}

public static void main(String args[]){

Parent p=new TestExceptionChild();

p.msg();

}

}

Test it Now

Output:Compile Time Error

2) Rule: If the superclass method does not declare an exception, subclass overridden method cannot declare the checked exception but can declare unchecked exception.

import java.io.\*;

class Parent{

void msg(){System.out.println("parent");}

}

class TestExceptionChild1 extends Parent{

void msg()throws ArithmeticException{

System.out.println("child");

}

public static void main(String args[]){

Parent p=new TestExceptionChild1();

p.msg();

}

}

Test it Now

Output:child

If the superclass method declares an exception

1) Rule: If the superclass method declares an exception, subclass overridden method can declare same, subclass exception or no exception but cannot declare parent exception.

Example in case subclass overridden method declares parent exception

import java.io.\*;

class Parent{

void msg()throws ArithmeticException{System.out.println("parent");}

}

class TestExceptionChild2 extends Parent{

void msg()throws Exception{System.out.println("child");}

public static void main(String args[]){

Parent p=new TestExceptionChild2();

try{

p.msg();

}catch(Exception e){}

}

}

Test it Now

Output:Compile Time Error

Example in case subclass overridden method declares same exception

import java.io.\*;

class Parent{

void msg()throws Exception{System.out.println("parent");}

}

class TestExceptionChild3 extends Parent{

void msg()throws Exception{System.out.println("child");}

public static void main(String args[]){

Parent p=new TestExceptionChild3();

try{

p.msg();

}catch(Exception e){}

}

}

Test it Now

Output:child

Example in case subclass overridden method declares subclass exception

import java.io.\*;

class Parent{

void msg()throws Exception{System.out.println("parent");}

}

class TestExceptionChild4 extends Parent{

void msg()throws ArithmeticException{System.out.println("child");}

public static void main(String args[]){

Parent p=new TestExceptionChild4();

try{

p.msg();

}catch(Exception e){}

}

}

Test it Now

Output:child

Example in case subclass overridden method declares no exception

import java.io.\*;

class Parent{

void msg()throws Exception{System.out.println("parent");}

}

class TestExceptionChild5 extends Parent{

void msg(){System.out.println("child");}

public static void main(String args[]){

Parent p=new TestExceptionChild5();

try{

p.msg();

}catch(Exception e){}

}

}

Test it Now

Output:child

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...");

}

}

Test it Now

Output:Exception occured: InvalidAgeException:not valid

rest of the code...