http://www.javatpoint.com/

Java Tutorial

1. Java - What, Where and Why?
2. What is Java
3. Where Java is used
4. Java Applications

Java technology is widely used currently. Let's start learning of java from basic questions like what is java, where it is used, what type of applications are created in java and why use java?

What is Java?

Java is a **programming language** and a **platform**.

Platform Any hardware or software environment in which a program runs, known as a platform. Since Java has its own Runtime Environment (JRE) and API, it is called platform.

Where it is used?

According to Sun, 3 billion devices run java. There are many devices where java is currently used. Some of them are as follows:

1. Desktop Applications such as acrobat reader, media player, antivirus etc.
2. Web Applications such as irctc.co.in, javatpoint.com etc.
3. Enterprise Applications such as banking applications.
4. Mobile
5. Embedded System
6. Smart Card
7. Robotics
8. Games etc.

Types of Java Applications

There are mainly 4 type of applications that can be created using java:

1) Standalone Application

It is also known as desktop application or window-based application. An application that we need to install on every machine such as media player, antivirus etc. AWT and Swing are used in java for creating standalone applications.

2) Web Application

An application that runs on the server side and creates dynamic page, is called web application. Currently, servlet, jsp, struts, jsf etc. technologies are used for creating web applications in java.

3) Enterprise Application

An application that is distributed in nature, such as banking applications etc. It has the advantage of high level security, load balancing and clustering. In java, EJB is used for creating enterprise applications.

4) Mobile Application

An application that is created for mobile devices. Currently Android and Java ME are used for creating mobile applications.

Java Version History

There are many java versions that has been released.

1. JDK Alpha and Beta (1995)
2. JDK 1.0 (23rd Jan, 1996)
3. JDK 1.1 (19th Feb, 1997)
4. J2SE 1.2 (8th Dec, 1998)
5. J2SE 1.3 (8th May, 2000)
6. J2SE 1.4 (6th Feb, 2002)
7. J2SE 5.0 (30th Sep, 2004)
8. Java SE 6 (11th Dec, 2006)
9. Java SE 7 (28th July, 2011)

Features of Java

There is given many features of java. They are also known as java buzzwords.

1. Simple
2. Object-Oriented
3. Platform independent
4. Secured
5. Robust
6. Architecture neutral
7. Portable
8. Dynamic
9. Interpreted
10. High Performance
11. Multithreaded
12. Distributed

Simple

According to Sun, Java language is simple because:

syntax is based on C++ (so easier for programmers to learn it after C++).

Removed many confusing and/or rarely-used features e.g., explicit pointers, operator overloading etc.

No need to remove unreferenced objects because there is Automatic Garbage Collection in java.

Object-oriented

Object-oriented means we organize our software as a combination of different types of objects that incorporates both data and behaviour.

Object-oriented programming(OOPs) is a methodology that simplify software development and maintenance by providing some rules.

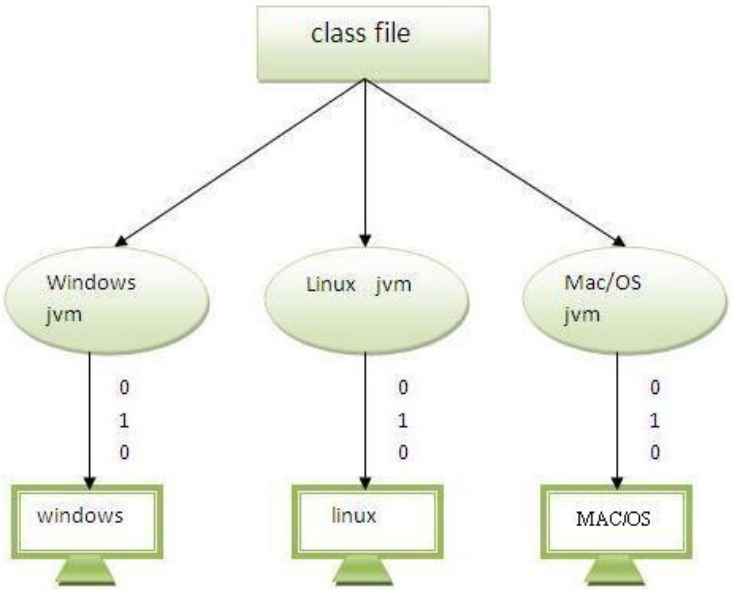
Basic concepts of OOPs are:

1. Object
2. Class
3. Inheritance
4. Polymorphism
5. Abstraction
6. Encapsulation

Platform Independent

A platform is the hardware or software environment in which a program runs. There are two types of platforms software-based and hardware-based. Java provides software-based platform. The Java platform differs from most other platforms in the sense that it's a software-based platform that runs on top of other hardware-based platforms. It has two components:

1. Runtime Environment
2. API(Application Programming Interface)

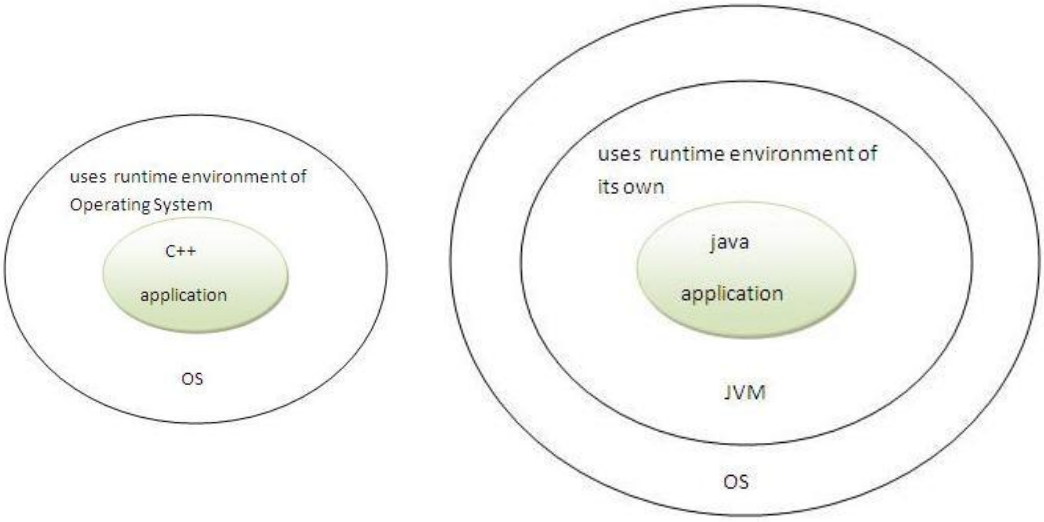


Java code can be run on multiple platforms e.g. Windows, Linux, Sun Solaris, Mac/OS etc. Java code is compiled by the compiler and converted into bytecode. This bytecode is a platform independent code because it can be run on multiple platforms i.e. Write Once and Run Anywhere(WORA).

Secured

Java is secured because:

- No explicit pointer
- Programs run inside virtual machine sandbox.



- **Classloader-** adds security by separating the package for the classes of the local file system from those that are imported from network sources.
- **Bytecode Verifier-** checks the code fragments for illegal code that can violate access right to objects.
- **Security Manager-** determines what resources a class can access such as reading and writing to the local disk.

These security are provided by java language. Some security can also be provided by application developer through SSL,JAAS,cryptography etc.

Robust

Robust simply means strong. Java uses strong memory management. There are lack of pointers that avoids security problem. There is automatic garbage collection in java. There is exception handling and type checking mechanism in java. All these points make java robust.

Architecture-neutral

There is no implementation dependent feature e.g. size of primitive types is set.

Portable

We may carry the java bytecode to any platform.

High-performance

Java is faster than traditional interpretation since byte code is "close" to native code still somewhat slower than a compiled language (e.g., C++)

Distributed

We can create distributed applications in java. RMI and EJB are used for creating distributed applications. We may access files by calling the methods from any machine on the internet.

Multi-threaded

A thread is like a separate program, executing concurrently. We can write Java programs that deal with many tasks at once by defining multiple threads. The main advantage of multi-threading is that it shares the same memory. Threads are important for multi-media, Web applications etc.

Creating hello java example

Let's create the hello java program:

```
1. class Simple{
2.     public static void main(String args[]){
3.      System.out.println("Hello Java");
4.     }
5. }
```

save this file as Simple.java

To compile: javac Simple.java

To execute: java Simple

```
Output:Hello Java
```

Understanding first java program

Let's see what is the meaning of class, public, static, void, main, String[], System.out.println().

- **class** keyword is used to declare a class in java.
- **public** keyword is an access modifier which represents visibility, it means it is visible to all.
- ***static*** *is a keyword, if we declare any method as static, it is known as static method. The core advantage of static method is that there is no need to create object to invoke the static method. The main method is executed by the JVM, so it doesn't require to create object to invoke the main method. So it saves memory.*
- **void** is the return type of the method, it means it doesn't return any value.
- **main** represents startup of the program.
- **String[] args** is used for command line argument. We will learn it later.
- **System.out.println()** is used print statement. We will learn about the internal working of System.out.println statement later.

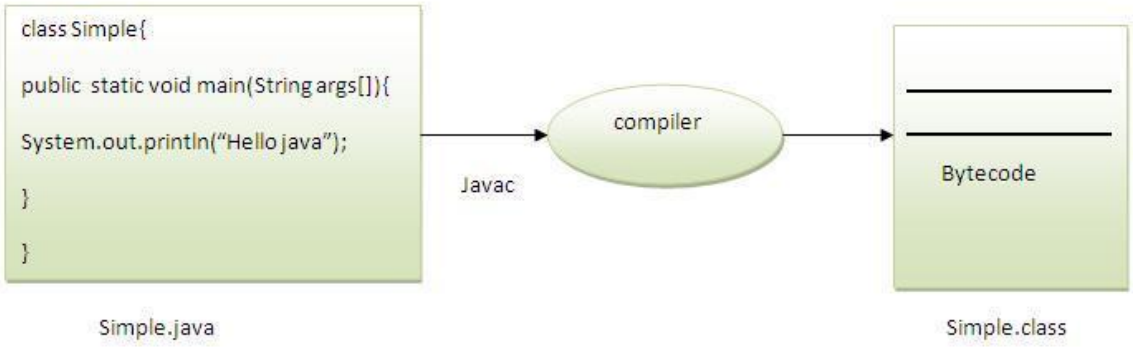
Internal Details of Hello Java Program

1. Internal Details of Hello Java

In the previous page, we have learned about the first program, how to compile and how to run the first java program. Here, we are going to learn, what happens while compiling and running the java program. Moreover, we will see some question based on the first program.

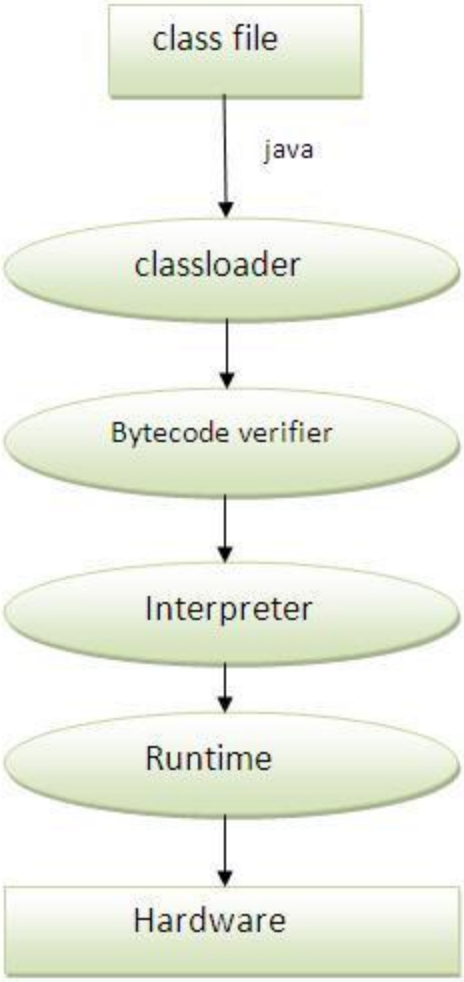
What happens at compile time?

At compile time, java file is compiled by Java Compiler (It does not interact with OS) and converts the java code into bytecode.



What happens at runtime?

At runtime, following steps are performed:



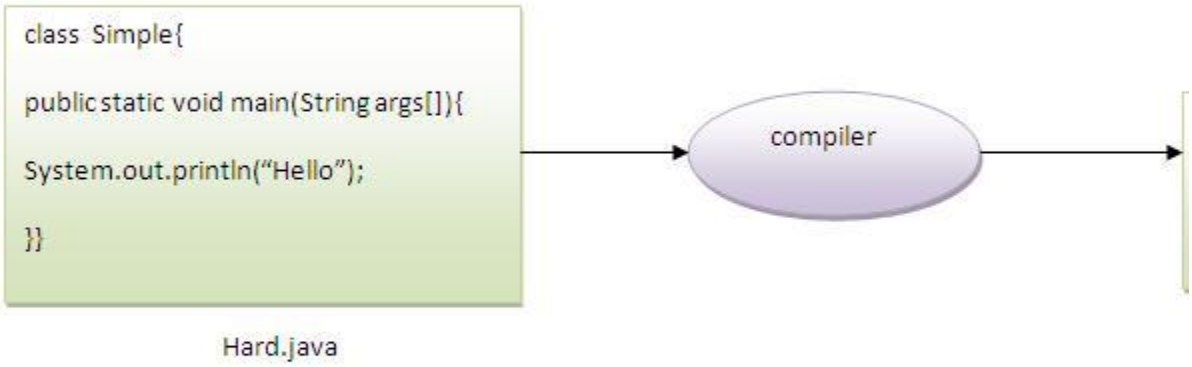
Classloader: is the subsystem of JVM that is used to load class files.

Bytecode Verifier: checks the code fragments for illegal code that can violate access right to objects.

Interpreter: read bytecode stream then execute the instructions.

Q)Can you save a java source file by other name than the class name?

Yes, like the figure given below illustrates:

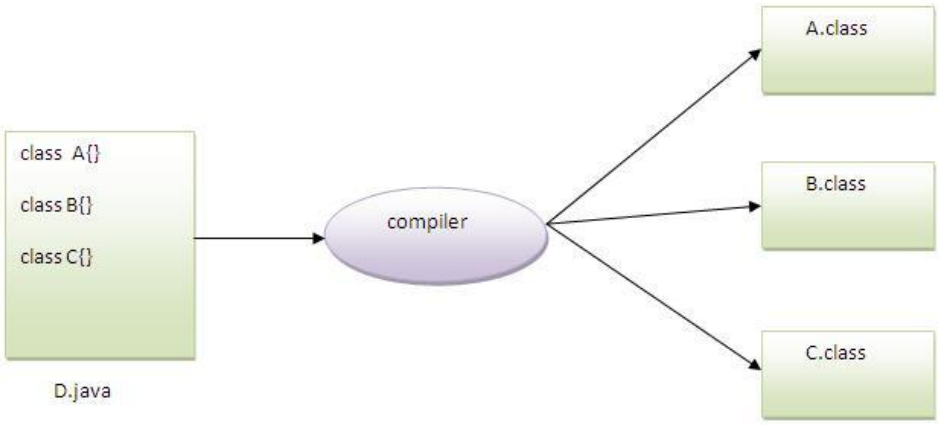


To compile: javac Hard.java

To execute: java Simple

Q)Can you have multiple classes in a java source file?

Yes, like the figure given below illustrates:



Difference between JDK, JRE and JVM

1. Brief summary of JVM
2. Java Runtime Environment (JRE)
3. Java Development Kit (JDK)

Understanding the difference between JDK, JRE and JVM is important in Java. We are having brief overview of JVM here.

If you want to get the detailed knowledge of Java Virtural Machine, move to the next page. Firstly, let's see the basic differences between the 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.

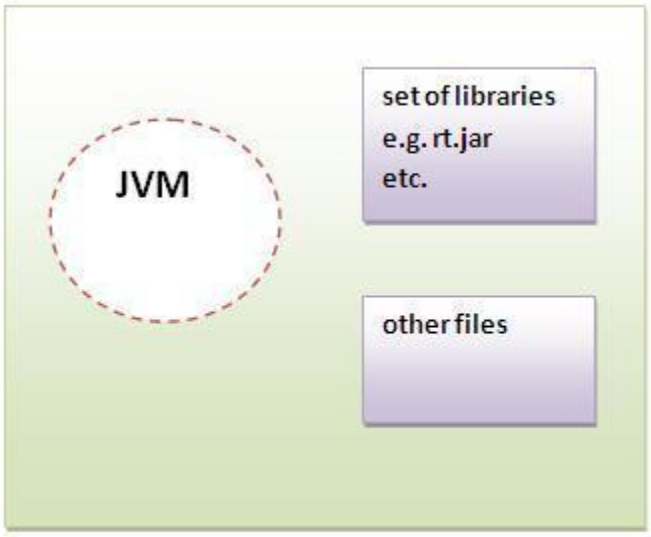
The JVM performs following main tasks:

- 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.

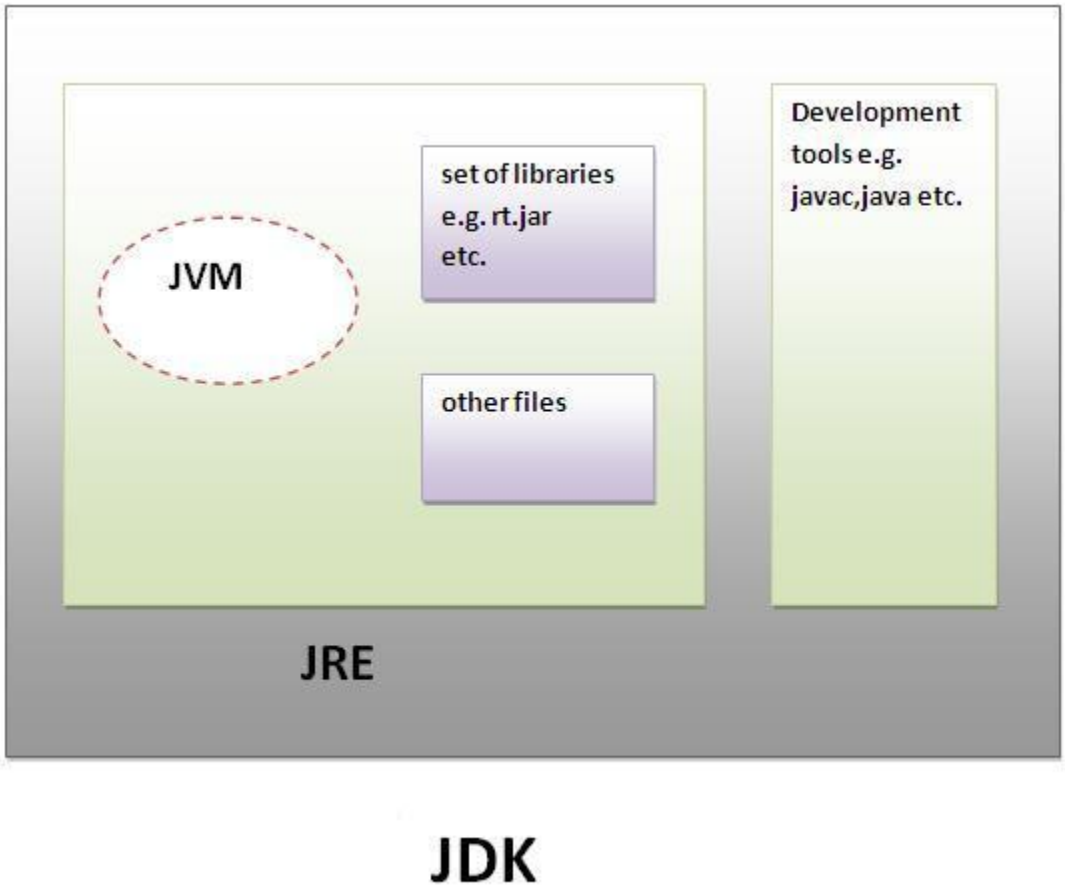
Implementation of JVMs are also actively released by other companies besides Sun Micro Systems.



JRE

JDK

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



JVM (Java Virtual Machine)

1. Java Virtual Machine
2. Internal Architecture of 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 (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:

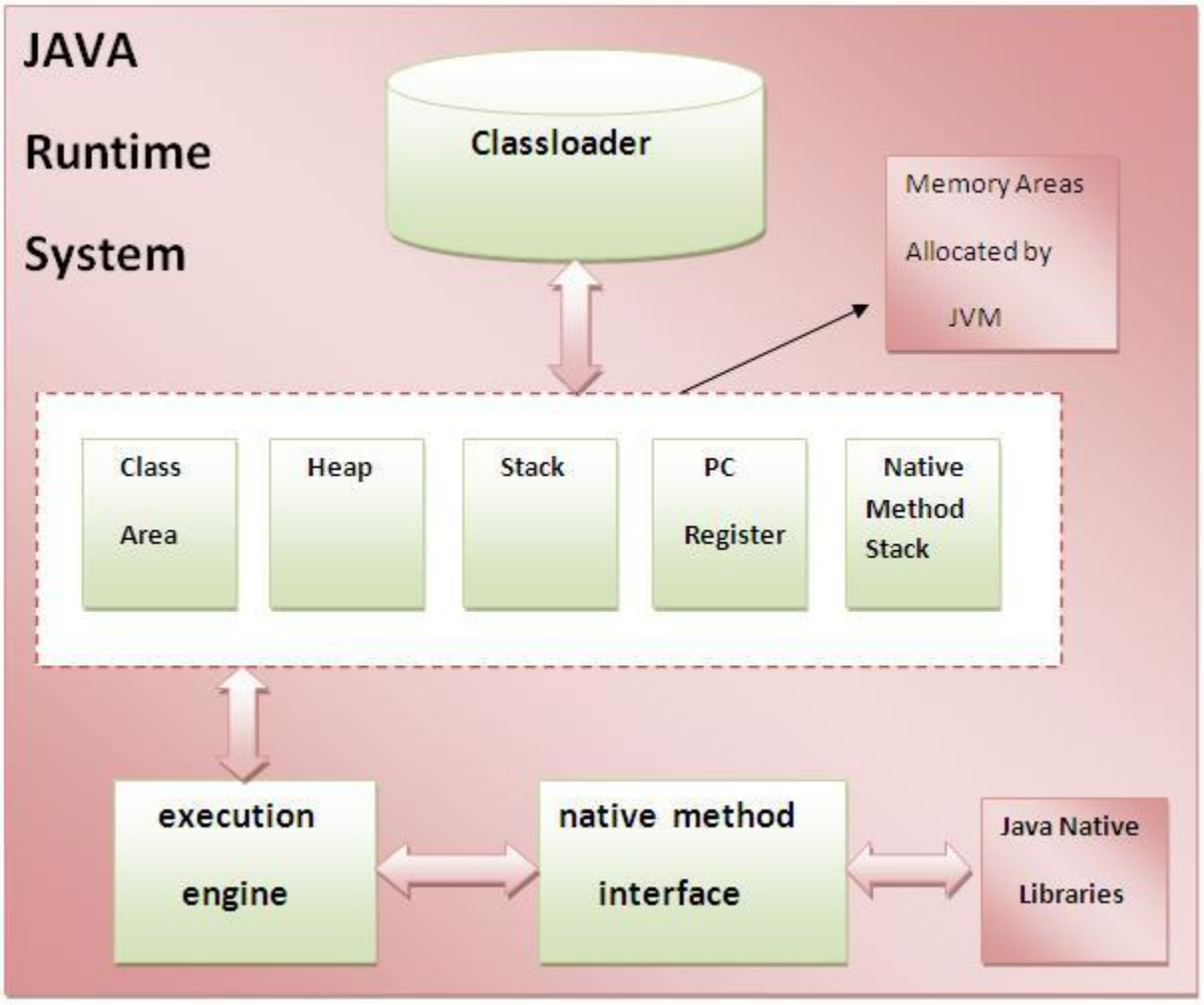
- Loads code
- Verifies code
- Executes code
- Provides runtime environment

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 Regiser:

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: Read bytecode stream then execute the instructions.

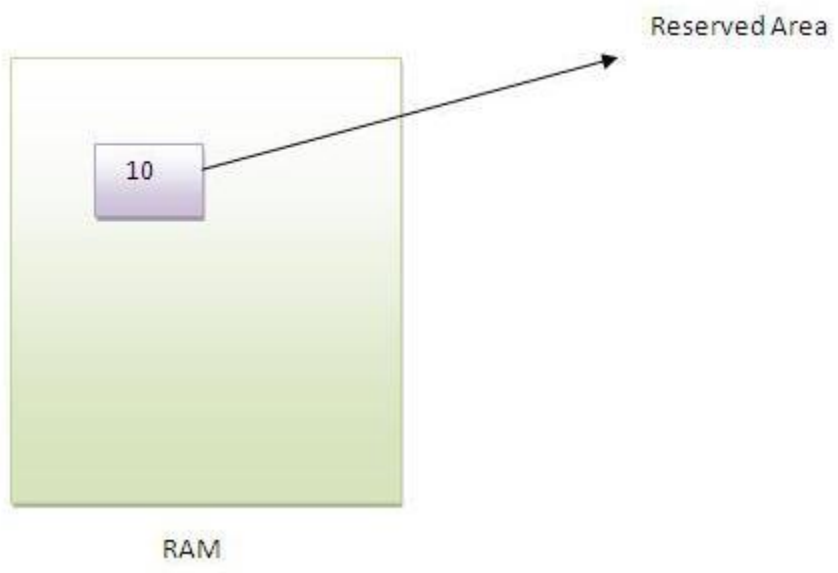
3) Just-In-Time (JIT) compiler: It is used to improve the performance. JIT compiles parts of the byte code that have similar functionality at the same time, and hence reduces the amount of time needed for compilation. Here the term compiler refers to a translator from the instruction set of a Java virtual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

Variable and Datatype in Java

In this page, we will learn about the variable and java data types. Variable is a name of memory location. There are three types of variables: local, instance and static. There are two types of data types in java, primitive and non-primitive.

Variable

Variable is name of reserved area allocated in memory.

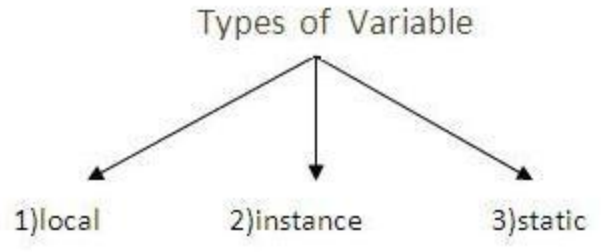



```
int data=50;//Here data is variable
```

Types of Variable

There are three types of variables in java

- local variable
- instance variable
- static variable



-

Local Variable

A variable that is declared inside the method is called local variable.

Instance Variable

A variable that is declared inside the class but outside the method is called instance variable. It is not declared as static.

Static variable

A variable that is declared as static is called static variable. It cannot be local.

We will have detailed learning of these variables in next chapters.

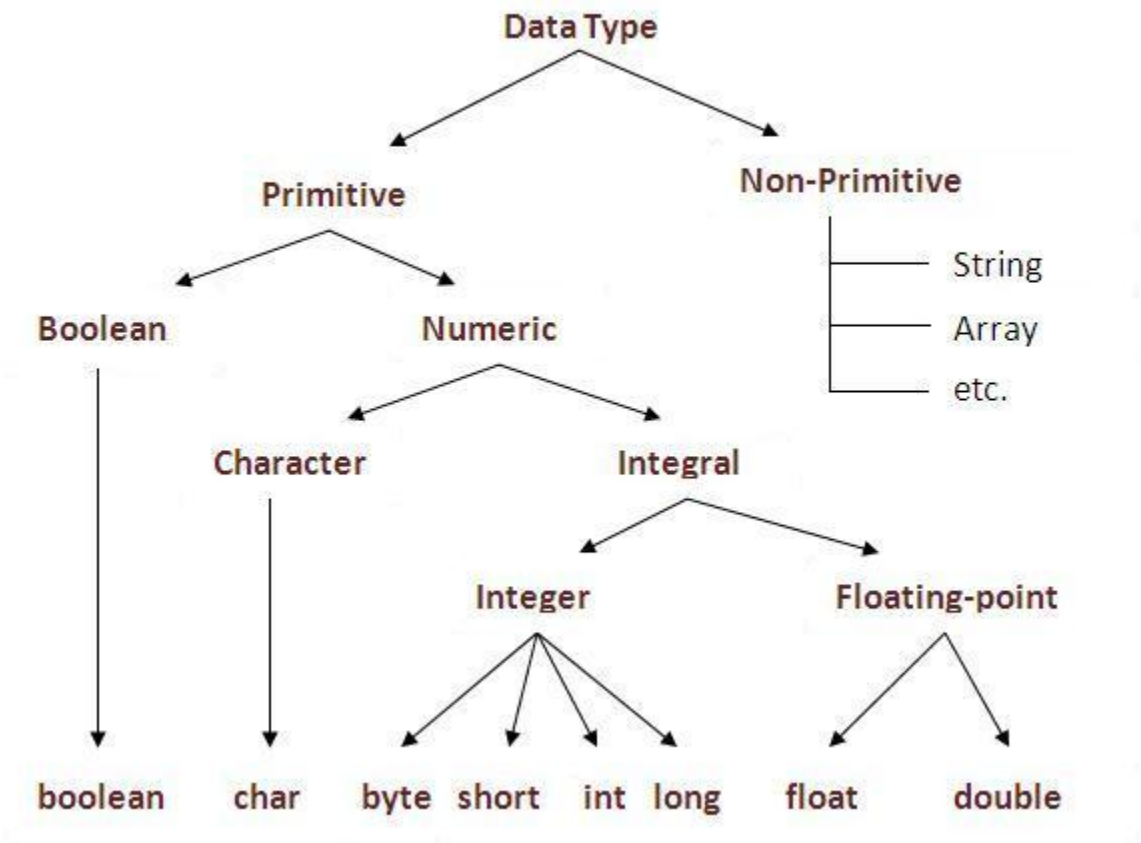
Example to understand the types of variables

```
class A{
	int data=50;//instance variable
	static int m=100;//static variable
	void method(){
		int n=90;//local variable
	}
}//end of class
```

Data Types in Java

In java, there are two types of data types

- primitive data types
- non-primitive data types



Data Type Default Value Default size

boolean	false	1 bit
char	'\u0000'	2 byte
byte	0	1 byte
short	0	2 byte
int	0	4 byte
long	0L	8 byte
float	0.0f	4 byte
double	0.0d	8 byte

Why char uses 2 byte in java and what is \u0000 ?

because java uses Unicode system rather than ASCII code system. \u0000 is the lowest range of unicode system.To get detail about Unicode see below.

Unicode System

Unicode is a universal international standard character encoding that is capable of representing most of the world's written languages.

Why java uses Unicode System?

Before Unicode, there were many language standards:

- **ASCII** (American Standard Code for Information Interchange) for the United States.
- **ISO 8859-1** for Western European Language.
- **KOI-8** for Russian.
- **GB18030 and BIG-5** for chinese, and so on.

This caused two problems:

1. A particular code value corresponds to different letters in the various language standards.
2. The encodings for languages with large character sets have variable length.Some common characters are encoded as single bytes, other require two or more byte.

To solve these problems, a new language standard was developed i.e. Unicode System.

In unicode, character holds 2 byte, so java also uses 2 byte for characters.

lowest value:\u0000

highest value:\uFFFF

Operators in java

Operator is a special symbol that is used to perform operations. There are many types of operators in java such as unary operator, arithmetic operator, relational operator, shift operator, bitwise operator, ternary operator and assignment operator.

Precedence of Operators	
Operators	Precedence
postfix	*expr*++ *expr*--
unary	++*expr* --*expr* +*expr* -*expr* ~ !
multiplicative	* / %
additive	+ -
shift	<< >> >>>
relational	< > <= >= instanceof
equality	== !=
bitwise AND	&
bitwise exclusive OR	^
bitwise inclusive OR	|
logical AND	&&
logical OR	||
Ternary	? :
assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

Useful Programs:

There is given some useful programs such as factorial number, prime number, fibonacci series etc.

It is better for the fresher's to skip this topic and come to it after OOPs concepts.

1) Program of factorial number.

```
class Operation{
	static int fact(int number){
		int f=1;
		for(int i=1;i<=number;i++){
			f=f*i;
		}
		return f;
	}

	public static void main(String args[]){
		int result=fact(5);
		System.out.println("Factorial of 5="+result);
	}
}
```

---

2) Program of fibonacci series.

```
class Fabnoci{
	public static void main(String...args)
	{
		int n=10,i,f0=1,f1=1,f2=0;
		for(i=1;i<=n;i++)
		{
			f2=f0+f1;
			f0=f1;
			f1=f2;
			f2=f0;
			System.out.println(f2);
		}
	}
}
```

---

3) Program of armstrong number.

```
class ArmStrong{
        public static void main(String...args)
        {
                int n=153,c=0,a,d;
                d=n;
                while(n>0)
                {
                        a=n%10;
                        n=n/10;
                        c=c+(a*a*a);
                }
                if(d==c)
                        System.out.println("armstrong number");
                else
                        System.out.println("it is not an armstrong number");
        }
}
```

---

4) Program of checking palindrome number.

```
class Palindrome
{
   public static void main( String...args)
  {
   int a=242;
   int  n=a,b=a,rev=0;
   while(n>0)
   {
     a=n%10;
     rev=rev*10+a;
     n=n/10;
   }
   if(rev==b)
   System.out.println("it is Palindrome");
   else
   System.out.println("it is not palinedrome");

  }
}
```

5) Program of swapping two numbers without using third variable.

```
class SwapTwoNumbers{
	public static void main(String args[]){
		int a=40,b=5;
		a=a*b;
		b=a/b;
		a=a/b;

		System.out.println("a= "+a);
		System.out.println("b= "+b);
	}
}
```

6) Program of factorial number by recursion

```
class FactRecursion{

	static int fact(int n){
		if(n==1)
			return 1;

		return n*=fact(n-1);
	}

	public static void main(String args[]){

		int f=fact(5);
		System.out.println(f);
	}
}
```

Java OOPs Concepts

1. Object Oriented Programming
2. Advantage of OOPs over Procedure-oriented programming language
3. Difference between Objcet-oriented and Objcet-based programming language.

In this page, we will learn about basics 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.

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 behavior 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 behaviors 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 convense 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.

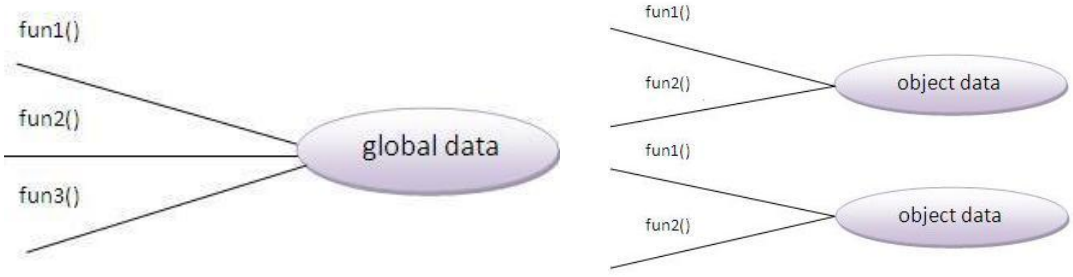
A java class is the example of encapsulation. Java bean is the fully encapsulated class because all the data members are private here.

Advantage of OOPs over Procedure-oriented programming language

1)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.

2)OOPs provides data hiding whereas in Procedure-oriented prgramming language a global data can be accessed from anywhere.

3)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.



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.

Do You Know ?

- Can we overload main method ?
- Constructor returns a value but, what ?
- Can we create a program without main method ?
- What are the 6 ways to use this keyword ?
- Why multiple inheritance is not supported in java ?
- Why use aggregation ?
- Can we override the static method ?
- What is covariant return type ?
- What are the three usage of super keyword?
- Why use instance initializer block?
- What is the usage of blank final variable ?
- What is marker or tagged interface ?
- What is runtime polymorphism or dynamic method dispatch ?
- What is the difference between static and dynamic binding ?
- How downcasting is possible in java ?
- What is the purpose of private constructor?
- What is object cloning ?

What we will learn in OOPs Concepts ?

- Advantage of OOPs
- Naming Convention
- Object and class
- Method overloading
- Constructor
- static keyword
- this keyword with 6 usage
- Inheritance
- Aggregation
- Method Overriding
- Covariant Return Type
- super keyword
- Instance Initializer block
- final keyword
- Abstract class
- Interface
- Runtime Polymorphism
- Static and Dynamic Binding
- Downcasting with instanceof operator
- Package

- Access Modifiers
- Encapsulation
- Object Cloning

Java Naming conventions

A **naming convention** is a rule to follow as you decide what to name your identifiers e.g. class, package, variable, constant, method etc.

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

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.

Understanding 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

1. Object in Java
2. Class in Java
3. Instace Variable in Java
4. Method in Java
5. Example of Object and class that maintains the records of student
6. Annonymous Object

In this page, we will learn about java objects and classes. 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 behavior is known as an object e.g. chair, bike, marker, pen, table, car etc. It can be physical or logical (tengible and intengible). The example of integible object is banking system.

An object has three characteristics:

- **state:** represents data (value) of an object.
- **behavior:** represents the behavior (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.

Object 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

A class 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 has 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 s1=new Student();   //creating an object of Student
            System.out.println(s1.id+" "+s1.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 behavior of an object.

Advantage of Method

- Code Reusability
- Code Optimization

new keyword

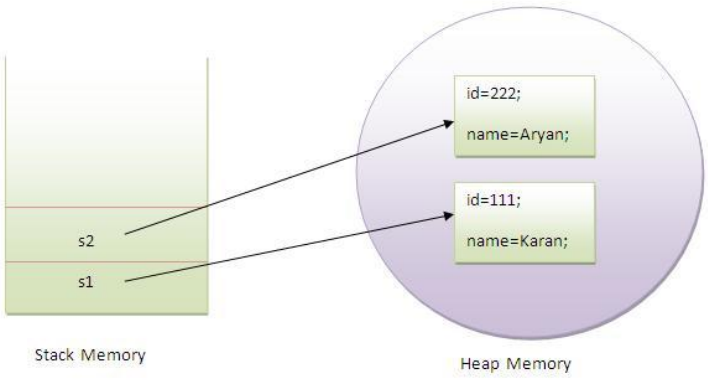
The new keyword is used to allocate memory at runtime.

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 insertRecord method on it. Here, we are displaying the state (data) of the objects by invoking the displayInformation method.

```
class Student{
        int rollno;
        String name;

        void insertRecord(int r, String n) {  //method
                rollno=r;
                name=n;
        }
        void displayInformation ( )  {
                System.out.println (rollno+" "+name);
        }//method
        public static void main ( String args[] ) {
                Student s1 = new Student ();
                Student s2 = new Student ();

                s1.insertRecord (111,"Karan");
                s2.insertRecord (222,"Aryan");

                s1.displayInformation ();
                s2.displayInformation ();

        }
}
```

```
Output: 111 Karan
        222 Aryan
```



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.

Another Example of Object and Class

There is given another example that maintains the records of Rectangle class. Its exaplanation is same as in the above Student class example.

```
class Rectangle{
    int length;
    int width;

    void insert(int l,int w){
            length=l;
            width=w;
    }

    void calculateArea () {
            System.out.println(length*width);
    }

    public static void main(String args[]){
            Rectangle r1=new Rectangle();
            Rectangle r2=new Rectangle();

            r1.insert(11,5);
            r2.insert(3,15);

            r1.calculateArea();
            r2.calculateArea();
    }
}
    Output:55
           45
```

What are the different ways to create an object in Java?

There are many ways to create an object in java. They are:

- By new keyword
- By newInstance() method
- By clone() method
- By factory method etc.

We will learn, these ways to create the object later.

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 Calculation{

    void fact(int  n){
            int fact=1;
            for(int i=1;i<=n;i++){
                    fact=fact*i;
            }
            System.out.println("factorial is "+fact);
    }

    public static void main(String args[]){
            new Calculation().fact(5);//calling method with annonymous object
    }
}
```

```
Output: Factorial is 120
```

Creating multiple objects by one type only

We can create multiple objects by one type only as we do in case of primitives.

1. Rectangle r1=new Rectangle(),r2=new Rectangle(); //creating two objects

Let's see the example:

```
class Rectangle{
    int length;
    int width;

    void insert(int l,int w){
            length=l;
            width=w;
    }

    void calculateArea(){
            System.out.println(length*width);
    }

    public static void main(String args[]){
            Rectangle r1=new Rectangle(),r2=new Rectangle();//creating two objects

            r1.insert(11,5);
            r2.insert(3,15);
```

```
        r1.calculateArea();
        r2.calculateArea();
    }
}
```

```
Output:55
        45
```

Method Overloading in Java

1. Different ways to overload the method
2. By changing the no. of arguments
3. By changing the data type
4. Why method overloading is not possible by changing the return type
5. Can we overload the main method
6. method overloading with Type Promotion

If a class has 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 a (int , int) for two parameters, and b(int,int,int) for three parameters then it may be difficult for you as well as other programmers to understand the behavior 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(int a,int b){
                System.out.println(a+b);
        }
        void sum(double a,double b){
                System.out.println(a+b);
        }

        public static void main(String args[]){
                Calculation obj=new Calculation();
                obj.sum(10.5,10.5);
                 obj.sum(20,20);

         }
}
```

```
Output:21.0
        40
```

Q) Why Method Overloaing 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);
        }

        public static void main(String args[]){
                Calculation obj=new Calculation();
                int result=obj.sum(20,20); //Compile Time Error
```

```
 }
}
```

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

---

Can we overload main() method?

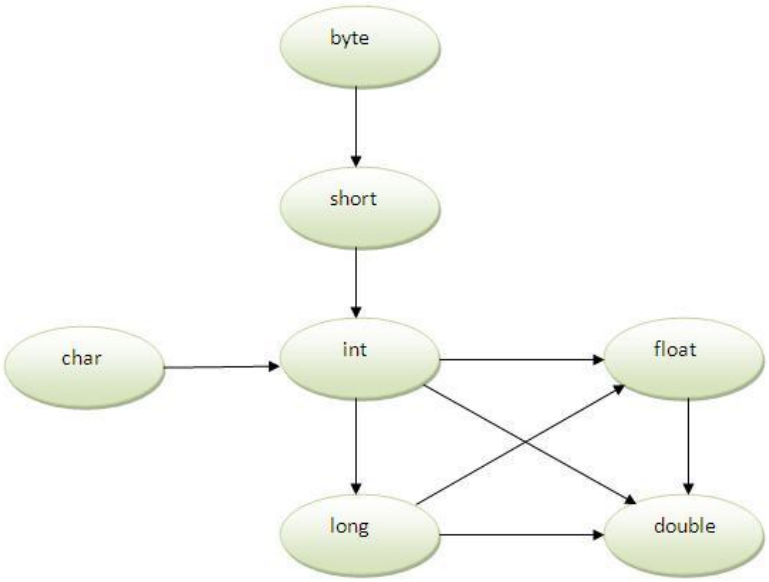
Yes, by method overloading. You can have any number of main methods in a class by method overloading. Let's see the simple example:

```
class Simple{
        public static void main(int a){
                System.out.println(a);
        }

        public static void main(String args[]){
                System.out.println("main() method invoked");
                main(10);
        }
}
```

```
Output:main() method invoked
        10
```

Method Overloading and TypePromotion

One type is promoted to another implicitly if no matching data type is found. Let's understand the concept by the figure given below:



As displayed in the above diagram, byte can be promoted to short, int, long, float or double. The short datatype can be promoted to int,long,float or double. The char datatype can be promoted to int,long,float or double and so on.

Example of Method Overloading with TypePromotion

```
class Calculation{
        void sum(int a,long 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(20,20);//now second int literal will be promoted to long
                obj.sum(20,20,20);

         }
}
```

```
Output:40
        60
```

Example of Method Overloading with TypePromotion if matching found

If there are matching type arguments in the method, type promotion is not performed.

```
class Calculation{
        void sum(int a,int b){
                System.out.println("int arg method invoked");
        }
        void sum(long a,long b){
                System.out.println("long arg method invoked");
        }

        public static void main(String args[]){
                Calculation obj=new Calculation();
                obj.sum(20,20);//now int arg sum() method gets invoked
        }
}
```

```
Output:int arg method invoked
```

Example of Method Overloading with TypePromotion in case ambiguity

If there are no matching type arguments in the method, and each method promotes similar number of arguments, there will be ambiguity.

```
class Calculation{
        void sum(int a,long b){
                System.out.println("a method invoked");
        }
        void sum(long a,int b){
                System.out.println("b method invoked");
        }

        public static void main(String args[]){
                Calculation obj=new Calculation();
                obj.sum(20,20);//now ambiguity
        }
}
```

```
Output:Compile Time Error
```

Constructor in Java

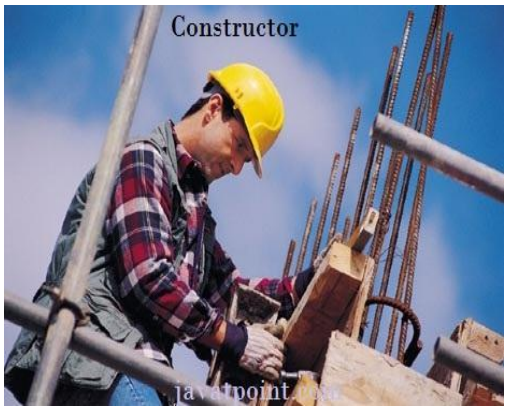
Constructor is a **special type of method** that is used to initialize the object.

Constructor is **invoked 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.

Rules for creating constructor

There are basically two rules defined for the constructor.

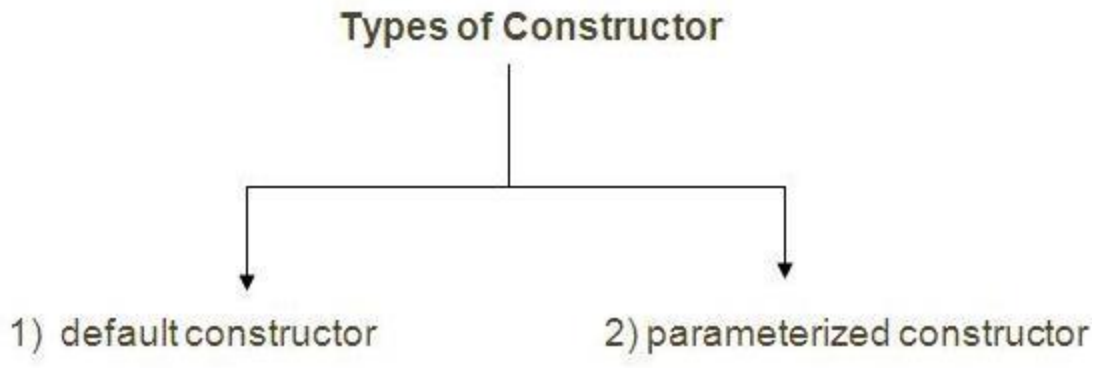
1. Constructor name must be same as its class name
2. Constructor must have no explicit return type



Types of constructors

There are two types of constructors:

1. default constructor (no-arg constructor)
2. parameterized constructor



---

1) Default Constructor

A constructor that has no parameter is known as default constructor.

Syntax of default constructor:

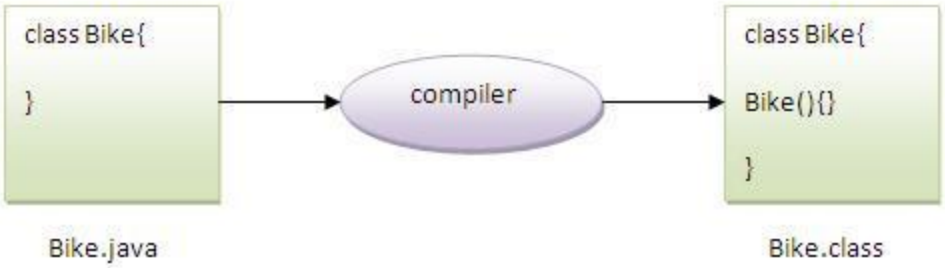
```
1. <class_name>(){}
```

Example of default constructor

In this example, we are creating the no-arg 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
```

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



Q) 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 id;
        String name;

        void display(){
                System.out.println(id+" "+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,you are not creating any constructor so compiler provides you a default constructor.Here 0 and null values are provided by default constructor.

Parameterized constructor

A constructor that have parameters is known as parameterized constructor.

Why use parameterized constructor?

Parameterized constructor is used to provide different values to the distinct objects.

Example of parameterized constructor

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

```
class Student{
        int id;
        String name;

        Student(int i,String n){
                id = i;
                name = n;
        }
        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:111 Karan
       222 Aryan
```

Constructor Overloading

Constructor overloading is a technique in Java in which a class can have any number of constructors that differ in parameter lists. The compiler differentiates these constructors by taking into account the number of parameters in the list and their type.

Example of Constructor Overloading

```
class Student{
        int id;
        String name;
        int age;
        Student(int i,String n){
                id = i;
                name = n;
```

```
    }
    Student(int i,String n,int a){
        id = i;
        name = n;
        age=a;
    }
    void display(){
        System.out.println(id+" "+name+" "+age);
    }

    public static void main(String args[]){
        Student s1 = new Student(111,"Karan");
        Student s2 = new Student(222,"Aryan",25);
        s1.display();
        s2.display();
    }
}
```

```
Output:111 Karan 0
       222 Aryan 25
```

What is the difference between constructor and method?

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

Constructor	Method
Constructor is used to initialize the state of an object.	Method is used to expose behavior 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.

Copying the values of one object to another like copy constructor in C++

There are many ways to copy the values of one object into another. They are:

- By constructor
- By assigning the values of one object into another
- By clone() method of Object class

In this example, we are going to copy the values of one object into another using constructor.

```
class Student{
        int id;
        String name;
        Student(int i,String n){
                id = i;
                name = n;
        }

        Student(Student s){
                id = s.id;
                name =s.name;
        }
        void display(){
                System.out.println(id+" "+name);
        }

        public static void main(String args[]){
                Student s1 = new Student(111,"Karan");
                Student s2 = new Student(s1);
                s1.display();
                s2.display();
        }
}
```

```
Output:111 Karan
       111 Karan
```

Copying the values of one object to another without constructor

We can copy the values of one object into another by assigning the objects values to another object. In this case, there is no need to create the constructor.

```
class Student{
        int id;
```

```
        String name;
        Student(int i,String n){
                id = i;
                name = n;
        }
        Student(){
                // Empty constructor
        }
        void display(){
                System.out.println(id+" "+name);
        }

        public static void main(String args[]){
                Student s1 = new Student(111,"Karan");
                Student s2 = new Student();
                s2.id=s1.id;
                s2.name=s1.name;
                s1.display();
                s2.display();
        }
}
```

```
Output:111 Karan
       111 Karan
```

Q) Does constructor return any value?

Ans: Yes, that is current class instance (You cannot use return type yet it returns a value).

Can constructor perform other tasks instead of initialization?

Yes, like object creation, starting a thread, calling method etc. You can perform any operation in the constructor as you perform in the method.

static keyword

The **static keyword** is used in java mainly for memory management. We may apply static keyword with variables, methods, blocks and nested class. The static keyword belongs to the class instead of 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) static variable

If you 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 your program **memory efficient** (i.e. it saves memory).

Understanding problem without static variable

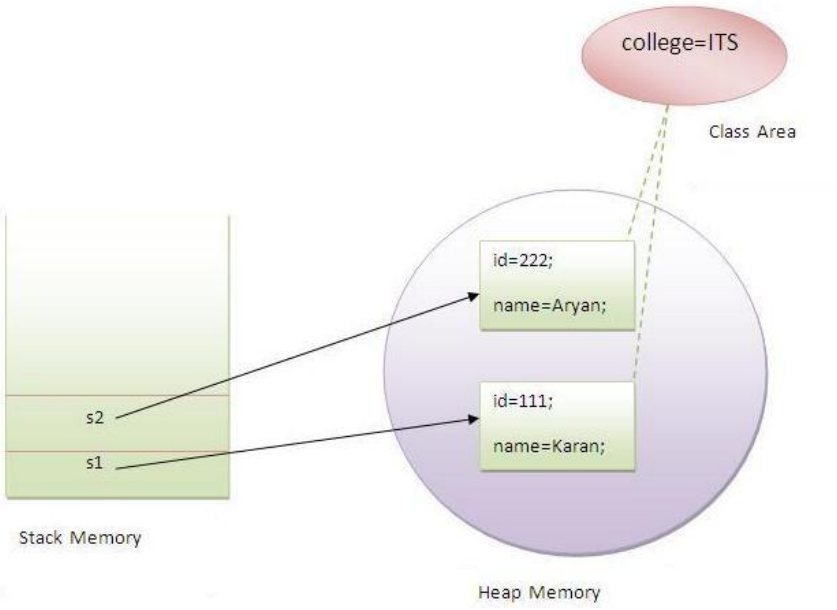
```
class Student{
	int rollno;
	String name;
	String college="ITS";
}
```

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 roll no 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.

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

                s1.display();
                s2.display();
        }
}
Output:111 Karan ITS
       222 Aryan ITS
```



Program of counter without static variable

In this example, we have created an instance variable named count which is incremented in the constructor. Since instance variable gets the memory at the time of object creation, each object will have the copy of the instance variable, if it is incremented, it won't reflect to other objects. So each object will have the value 1 in the count variable.

```
class Counter{
	int count=0;//will get memory when instance is created

	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
       1
       1
```

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) static method

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

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

```
class A{
        int a=40;//non static

        public static void main(String args[]){
                System.out.println(a);
        }
}
Output:Compile Time Error
```

Q) why main method is static?

Ans) because object is not required to call static method if it were non-static method, jvm create object first then call main() method that will lead the problem of extra memory allocation.

---

3) static block

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

Example of static block

```
class A{

        static {
                System.out.println("static block is invoked");
        }

        public static void main(String args[]){
                System.out.println("Hello main");
        }
}
Output: static block is invoked
        Hello main
```

Que)Can we execute a program without main() method?

Ans)Yes, one of the way is static block but in previous version of JDK not in JDK 1.7.

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

```
                System.exit(0);
        }
}
Output:static block is invoked (if not JDK7)
```

this keyword

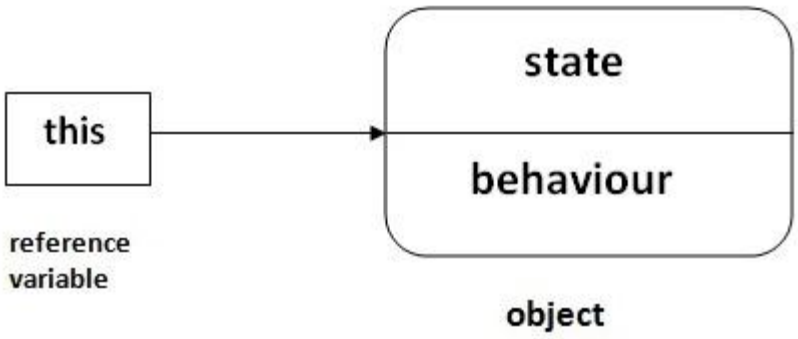
There can be a lot of usage of **this keyword**. In java, this is a **reference variable** that refers to the current object.

Usage of this keyword

Here is given the 6 usage of this keyword.

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

Suggestion: If you are beginner to java, lookup only two usage of this keyword.



---

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;

        student(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
```

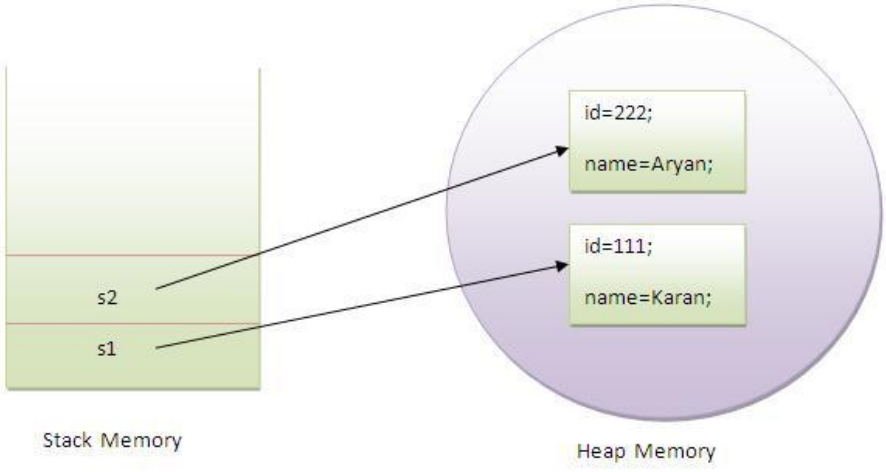
In the above example, parameter (formal arguments) and instance variables are same that is why we are using this keyword to distinguish between local variable and instance

variable.

Solution of the above problem by this keyword

```
//example of this keyword
class Student{
        int id;
        String name;

        student(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(222,"Aryan");
                s1.display();
                s2.display();
        }
}
```

```
Output111 Karan
        222 Aryan
```



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

Program where this keyword is not required

```
class Student{
	int id;
	String name;

	student(int i,String n){
		id = i;
		name = n;
	}
	void display(){
		System.out.println(id+" "+name);
	}
	public static void main(String args[]){
		Student e1 = new Student(111,"karan");
		Student e2 = new Student(222,"Aryan");
		e1.display();
		e2.display();
	}
}
```

```
Output:111 Karan
       222 Aryan
```

---

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 e1 = new Student(111,"karan");
                Student e2 = new Student(222,"Aryan");
                e1.display();
                e2.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 e1 = new Student(111,"karan");
                Student e2 = new Student(222,"Aryan","delhi");
                e1.display();
                e2.display();
        }
}
```

```
Output:111 Karan null
        222 Aryan delhi
```

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

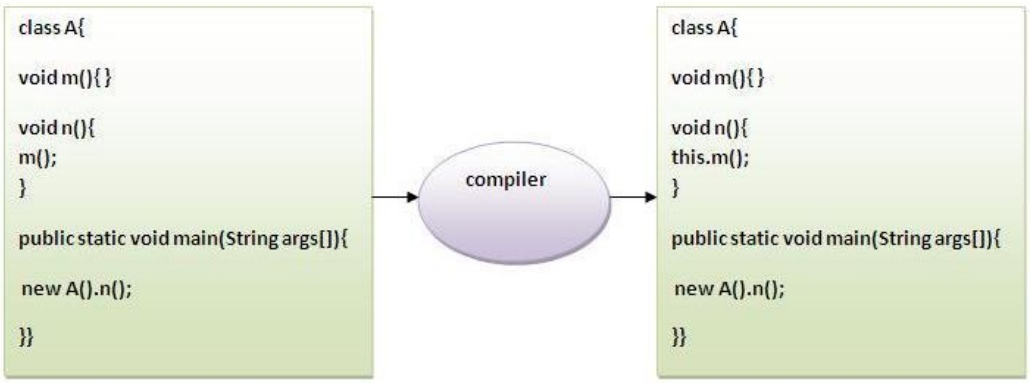
```
class Student{
        int id;
        String name;
        Student (){
                System.out.println("default constructor is invoked");
        }

        Student(int id,String name){
                id = id;
                name = name;
                this ();//must be the first statement
        }
        void display(){
                System.out.println(id+" "+name);
        }

        public static void main(String args[]){
                Student e1 = new Student(111,"karan");
                Student e2 = new Student(222,"Aryan");
                e1.display();
                e2.display();
        }
}
```

```
Output: Compile Time Error
```

---

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.

The 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 p(){
            m(this);
        }

        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 the “this” keyword in the constructor also. It is useful if we have to use one

object in multiple classes. Let's see the example:

```
1. class B{
2.   A obj;
3.   B(A obj){
4.     this.obj=obj;
5.   }
6.   void display(){
7.     System.out.println(obj.data);//using data member of A class
8.   }
9. }
10.
11. class A{
12.   int data=10;
13.   A(){
14.    B b=new B(this);
15.    b.display();
16.   }
17.   public static void main(String args[]){
18.    A a=new A();
19.   }
20. }
```

```
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

```
1. return_type method_name(){
2. return this;
3. }
```

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
```

Proving this keyword

Let's prove that this keyword refers to the current class instance variable. In this program, we are printing the reference variable and this, output of both variables are same.

```
class A{
        void m(){
                System.out.println(this);//prints same reference ID
        }

        public static void main(String args[]){
                A obj=new A();
                System.out.println(obj);//prints the reference ID

                obj.m();
        }
}
```

Inheritance in Java

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

The idea behind inheritance is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you reuse (or inherit) methods and fields, and you add new methods and fields to adapt your new class to new situations.

Inheritance represents the **IS-A relationship**.

Why use Inheritance?

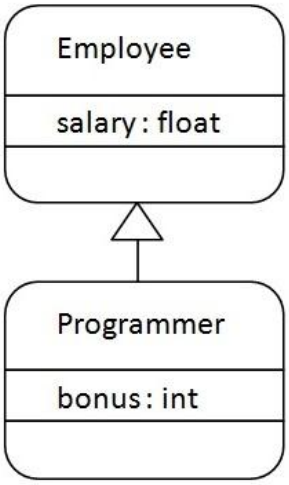
- For Method Overriding (So Runtime Polymorphism).
- For Code Reusability.

Syntax of Inheritance

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

The keyword extends indicates that you are making a new class that derives from an existing class. In the terminology of Java, a class that is inherited is called a superclass. 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);
        }
}
```

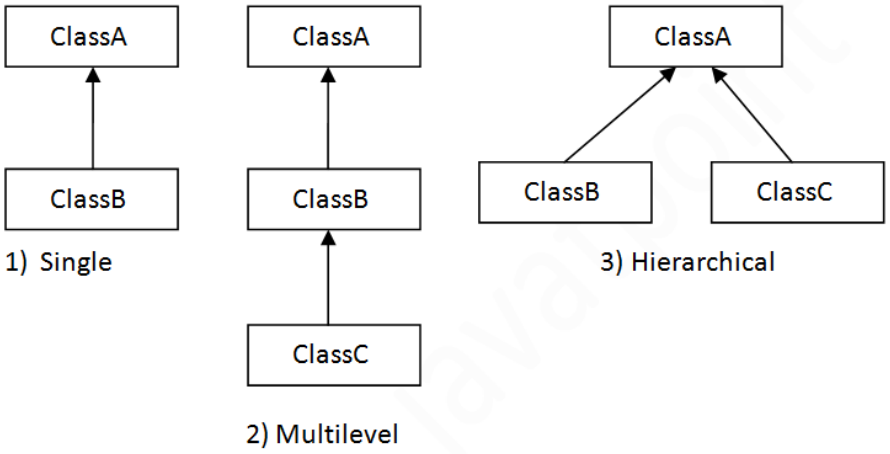
```
Output: Programmer salary is:40000.0
        Bonus of programmer is:10000
```

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

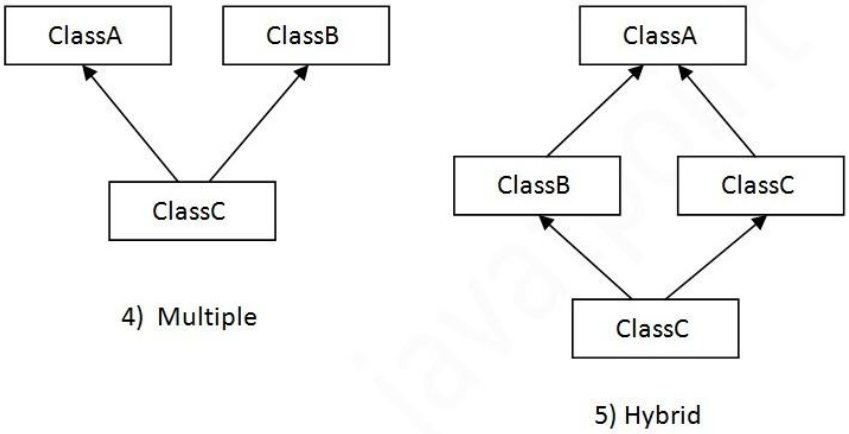
On the basis of class, there can be three types of inheritance: single, multilevel and hierarchical.

Multiple and Hybrid is supported through interface only. We will learn about interfaces later.



Multiple inheritances is not supported in java in case of class.

When a class extends multiple classes i.e. known as multiple inheritance. For Example:



Q) Why multiple inheritance is not supported in java?

- To reduce the complexity and simplify the language, multiple inheritance is not supported in java. For example:

```
class A{
        void msg(){
                System.out.println("Hello");
        }
}

class B{
```

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

class C extends A,B{//suppose if it were

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

Aggregation in Java

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

Consider a situation; Employee object contains much information such as id, name, email Id etc. It contains one more object named address, which contains its own information such as city, state, country, zip code etc. as given below.

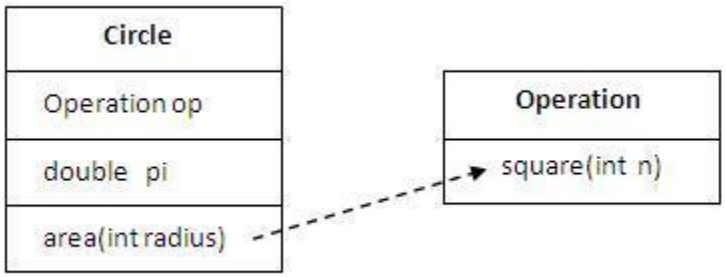
```
class Employee{
        int id;
        String name;
        Address address;//Address is a class
        ...
}
```

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();
                int rsquare=op.square(radius);//code reusability
                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 information such 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;

	public Emp(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("gzb","UP","india");
```

```
        Address address2=new Address("gno","UP","india");

        Emp e=new Emp(111,"varun",address1);
        Emp e2=new Emp(112,"arun",address2);

        e.display();
        e2.display();

    }
}
```

```
Output:111 varun
       gzb UP india
       112 arun
       gno UP india
```

http://www.javatpoint.com/aggregation-in-java