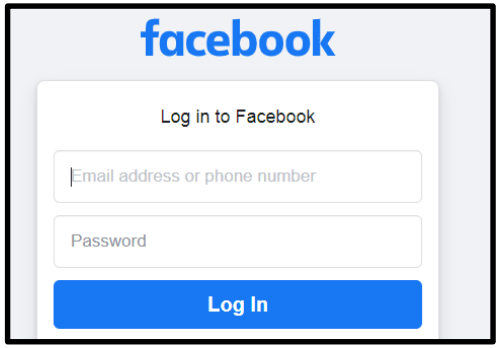
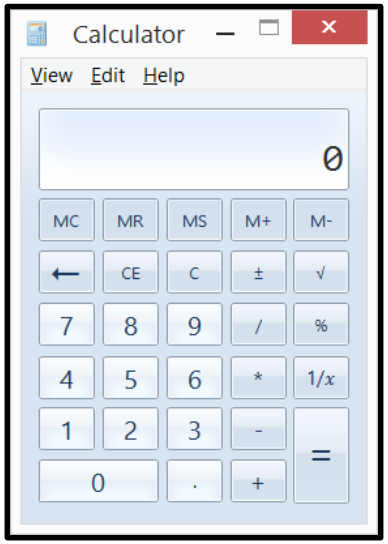

WhatsApp


facebook
Log in to Facebook
Email address or phone number
Password
Log In


Calculator
View Edit Help
0
MC MR MS M+ M-
← CE C ± √
7 8 9 / %
4 5 6 * 1/x
1 2 3 - =
0 . +

C

C++

C#

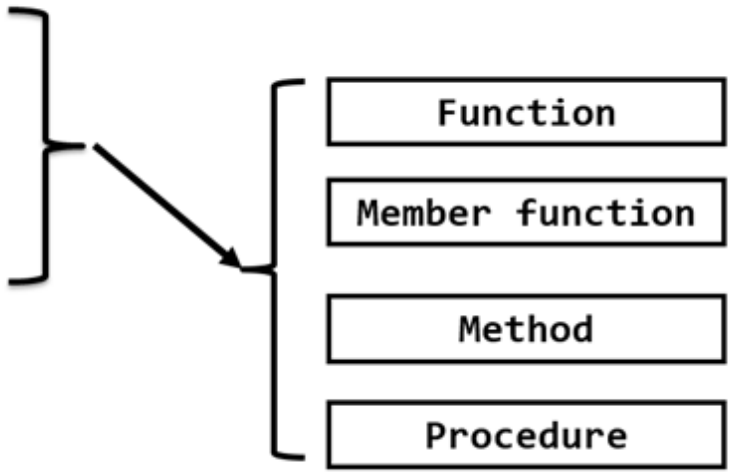
JAVA

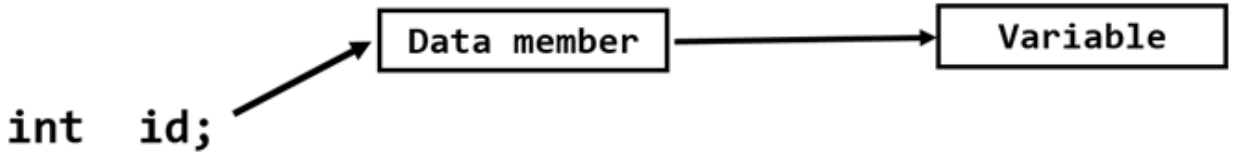
Python

Perl

```
void calculator()
{


}

void scientific_calculator()
{


}
```

```
void display()
{

}
```

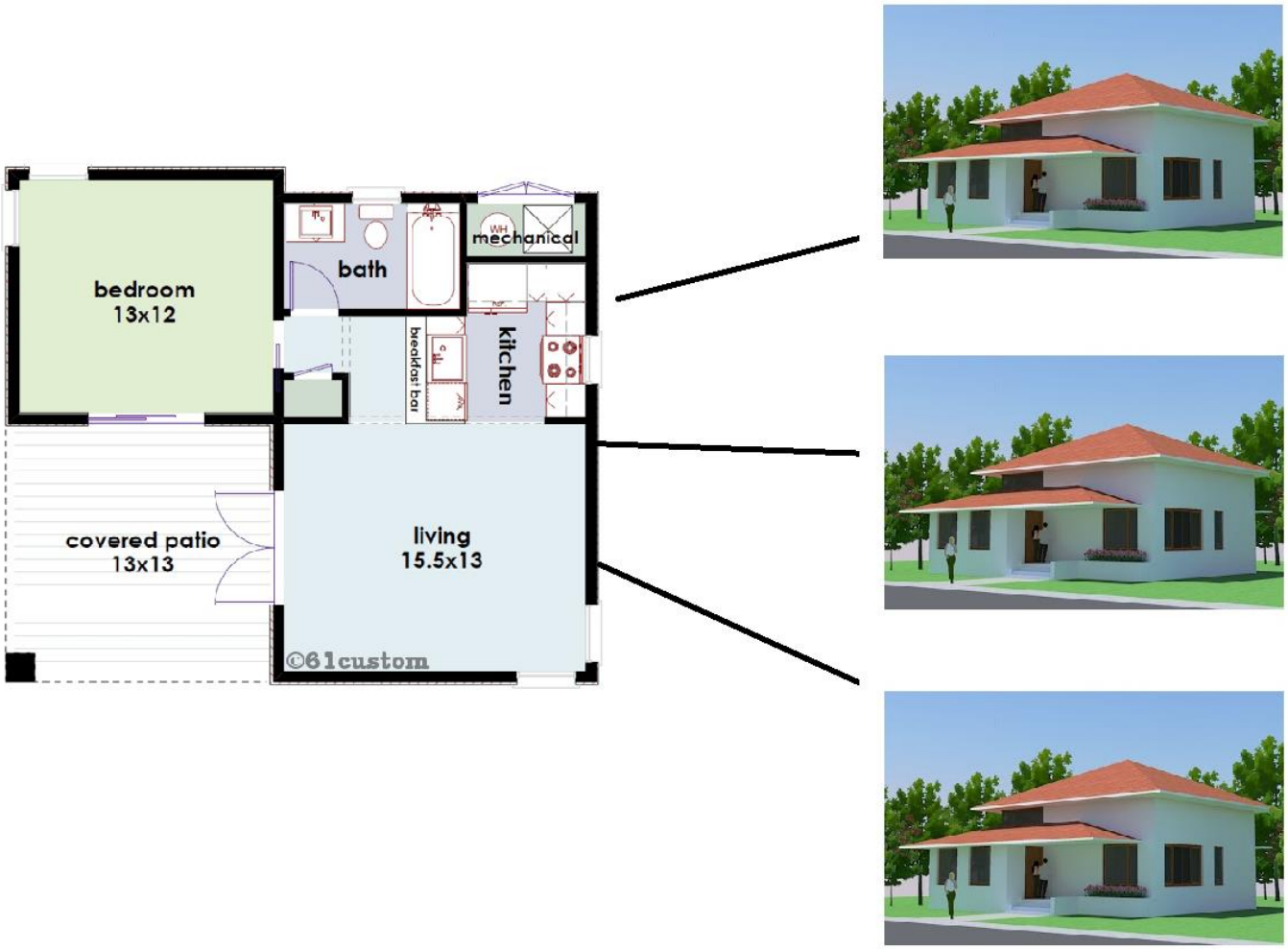





```
int  id;

void display()
{

}
```

1. Turmeric powder - 100 gms
2. Sugar - 1 kg
3. Jaggery - 1/2 kg
4. Idli rice/Boiled rice/Salem rice - 5-7 kgs
5. Steamed rice or Raw rice/Sona masoori - 5-7 kgs
6. High quality raw rice for Pongal - 1 kg
7. Dosa rice (optional) - 2 kgs
8. Basmati rice - 1 to 2 kgs

```
int     rollno
char    name[15];
char    city[15];

void display()
{

}
```


bedroom
13x12
bath
WH
mechanical
kitchen
breakfast bar
covered patio
13x13
living
15.5x13
©61custom

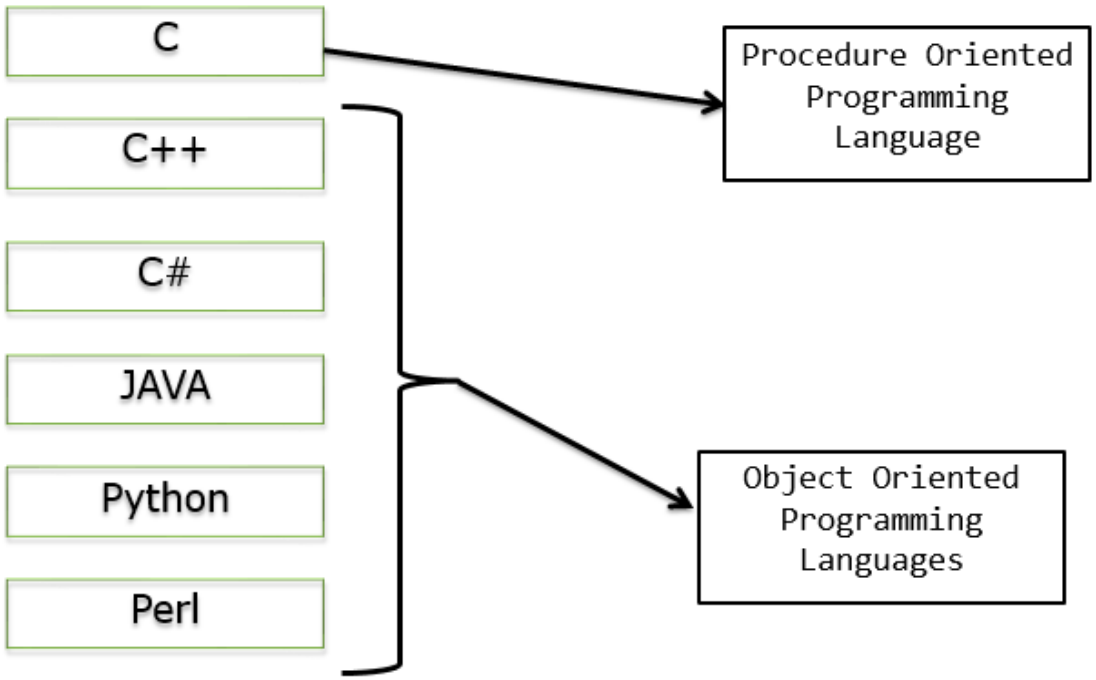
```
class StudentDetails
{
        int       rollno
        string    name;
        string    city;

        void display()
        {

        }
}
```

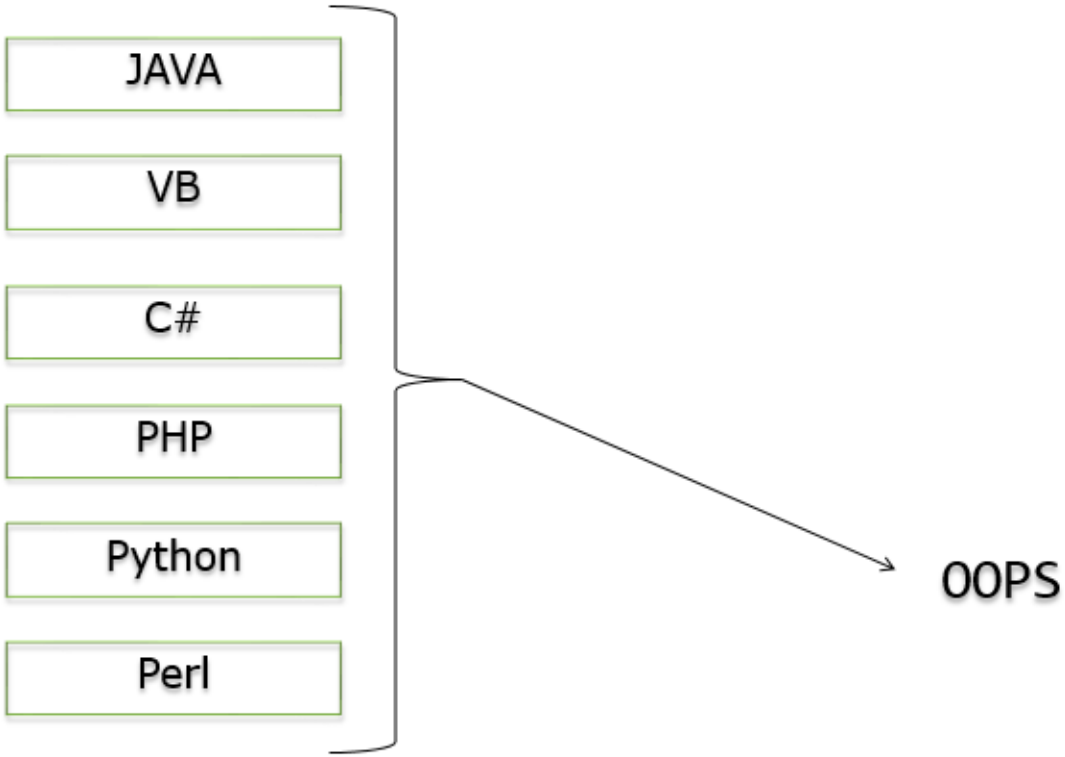
```
class StudentDetails
{
        int         rollno
        string      name;
        string      city;

        void display()
        {

        }
}
```

```
StudentDetails student1, student2, student3;

student1.rollno = 10;
student1.name   = "Ramesh";
student1.city   = "Salem";

student2.rollno = 20;
student2.name   = "Ganesh";
student2.city   = "Trichy";

student3.rollno = 30;
student3.name   = "Karthick";
student3.city   = "Chennai";
```

```
int x, y, z;

void display()
{

}
```

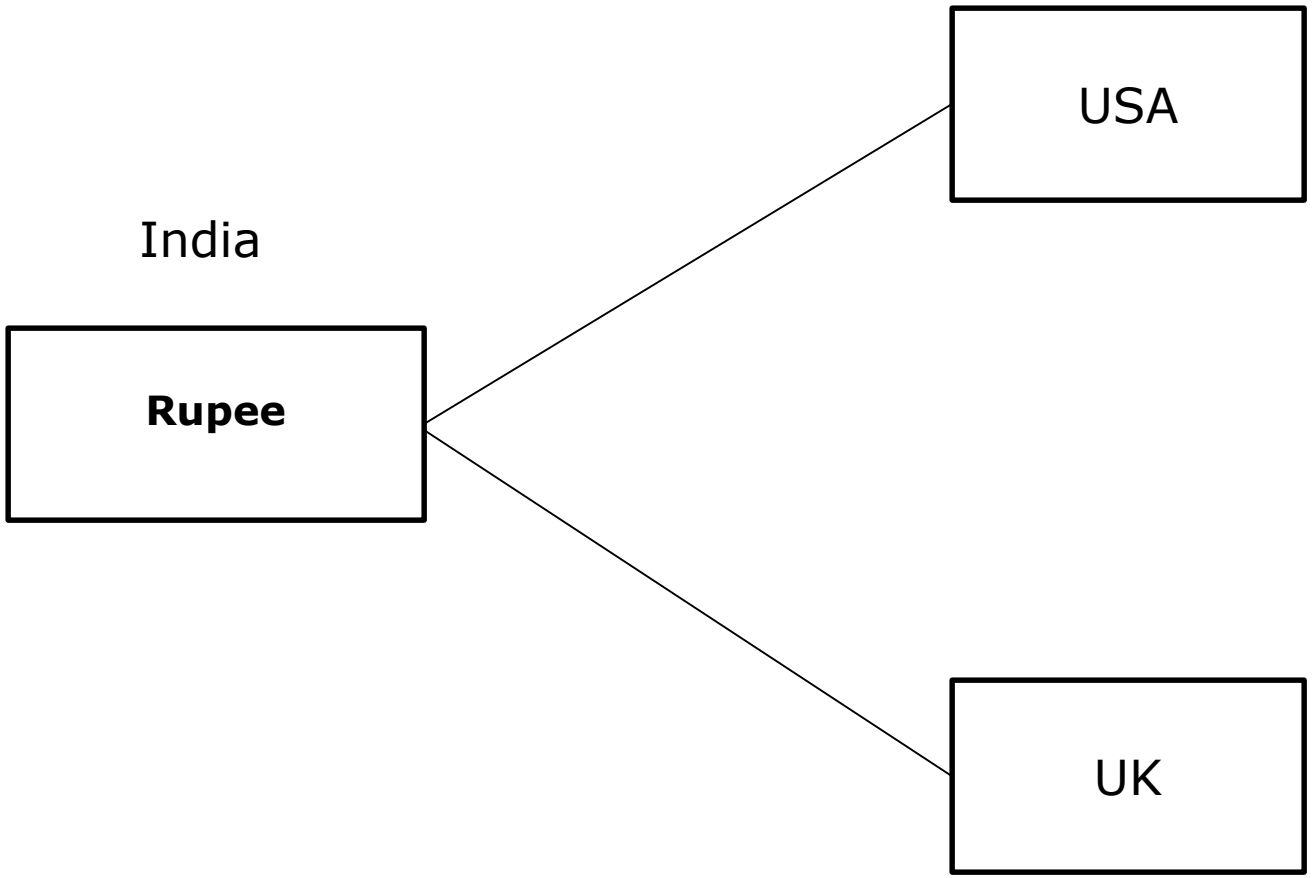
```
class Test
{
        int x, y, z;
        void show()
        {

        }
}
```

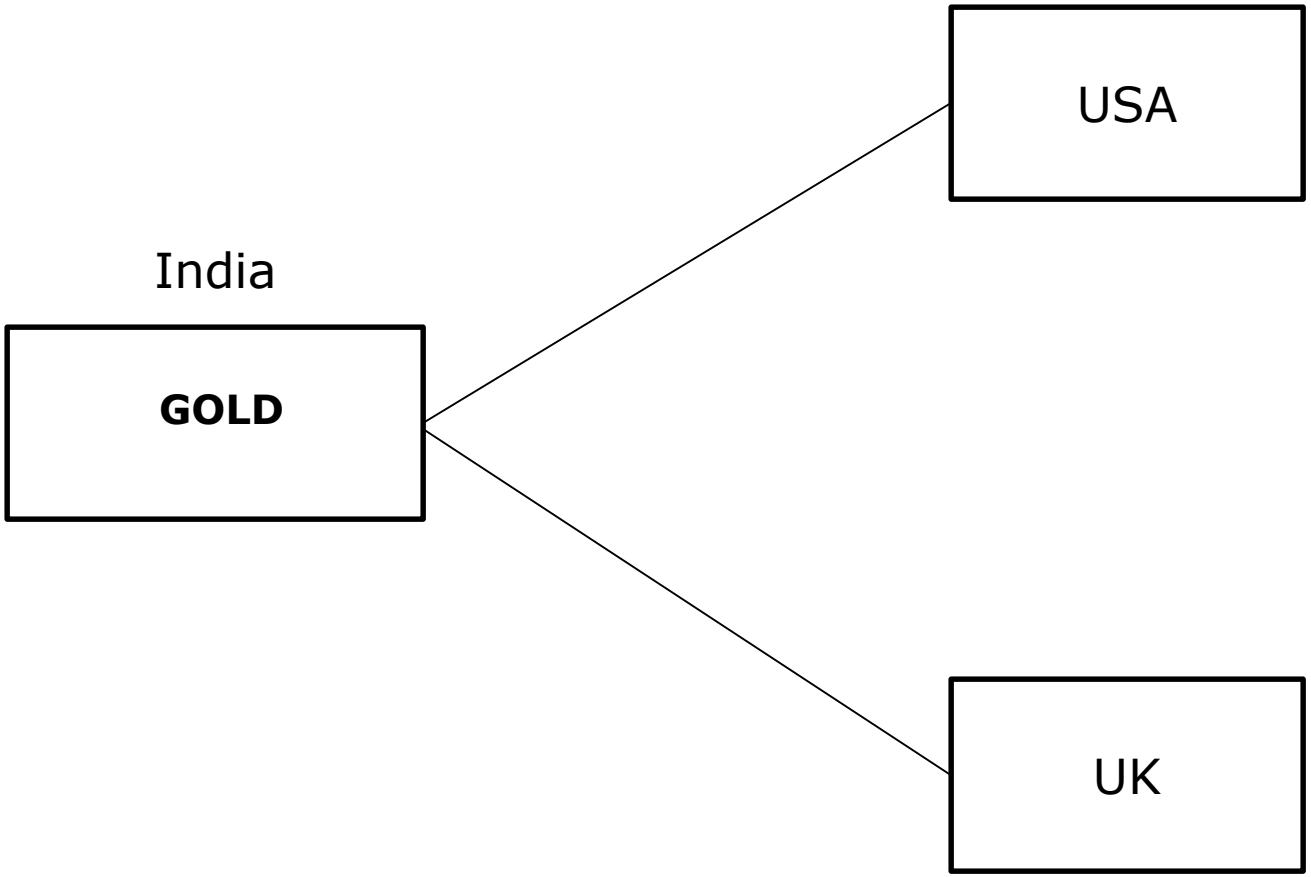

C
C++
C#
JAVA
Python
Perl
Procedure Oriented Programming Language
Object Oriented Programming Languages

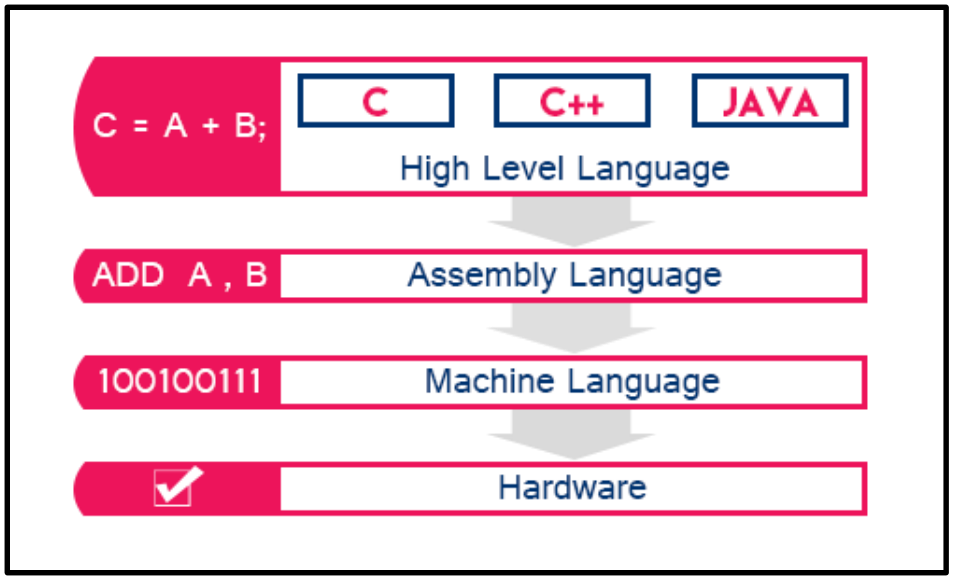

JAVA
VB
C#
PHP
Python
Perl
OOPS

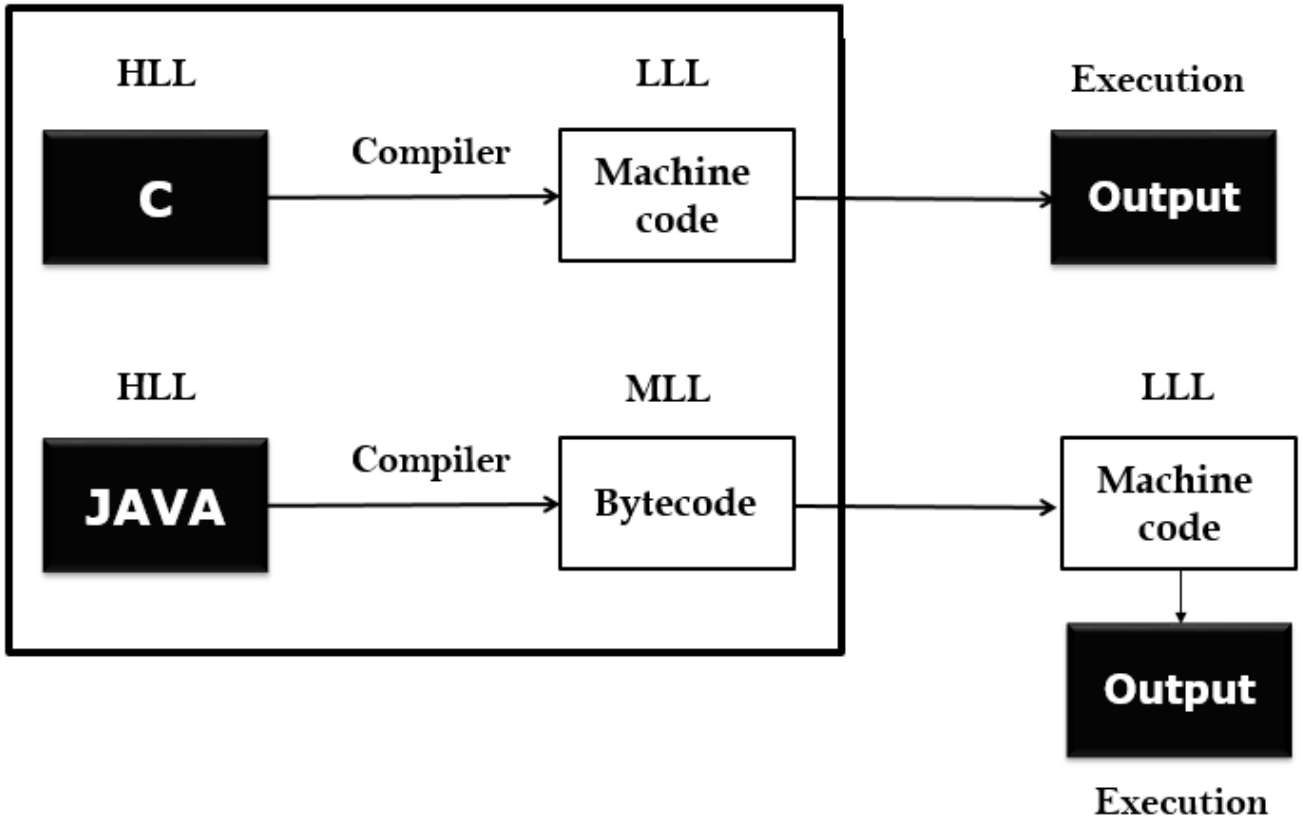
Java Features

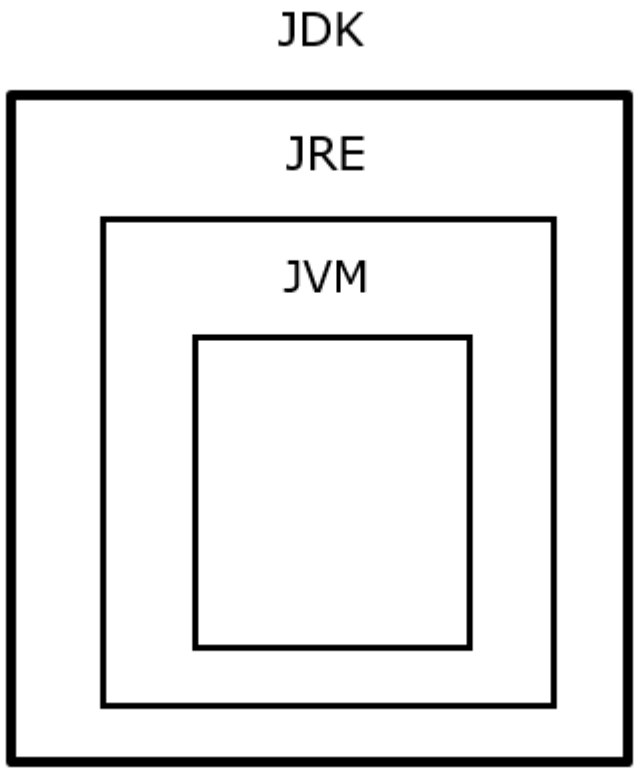
- Platform Independence
- Object Oriented Programming Language


USA
India
Rupee
UK


USA
India
GOLD
UK


C = A + B;
C
C++
JAVA
High Level Language
ADD A , B
Assembly Language
100100111
Machine Language
Hardware


HLL
LLL
Execution
C
Compiler
Machine code
Output
HLL
MLL
LLL
JAVA
Compiler
Bytecode
Machine code
Output
Execution

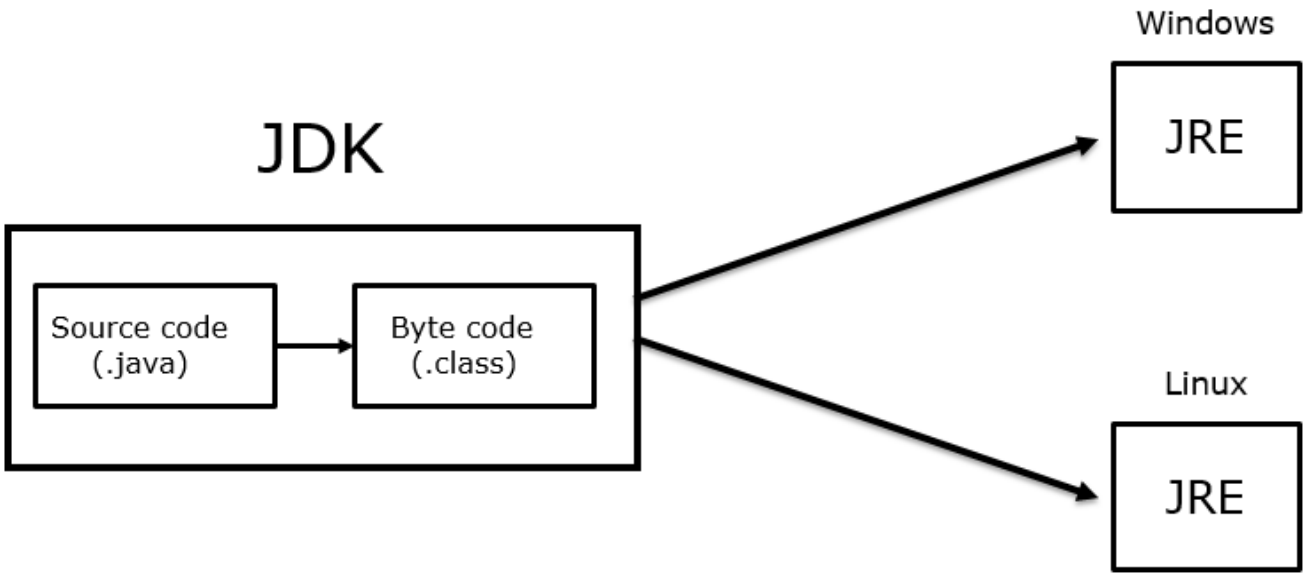

JDK
JRE
JVM

JDK → Java Development Kit

JRE → Java Runtime Environment

JVM → Java Virtual Machine

Platform Independence



Programming Basics

```
System.out.println(“Welcome”);
```

Data Types

- ✓ boolean
- ✓ char
- ✓ byte
- ✓ short
- ✓ int
- ✓ long
- ✓ float
- ✓ double

```
int     a    =  100;
char    b    =  'S';
float   c    =  20.4f;
```

Value Type

```
int     *x =  &a;
char    *y =  &b;
float   *z =  &c;
```

Reference Type

```
int     x   =   100;

char    y   =   'S';
```

Mapping

```
int  id;
```

Data Member

```
void display()
{
  printf(id);
}
```

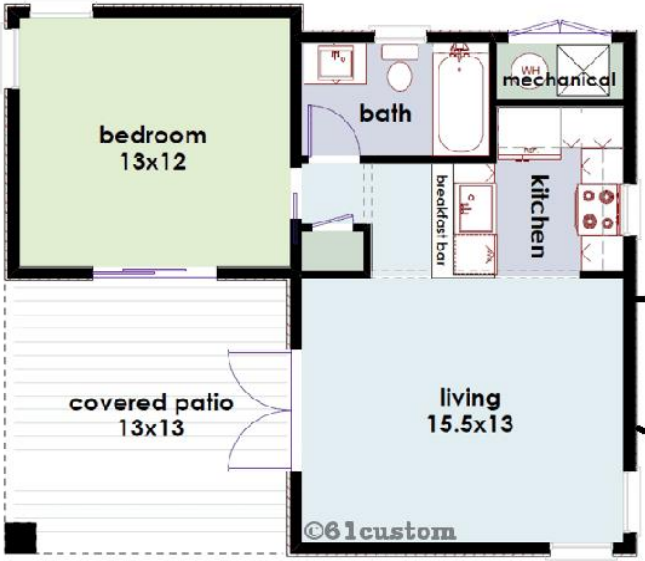
Member Function

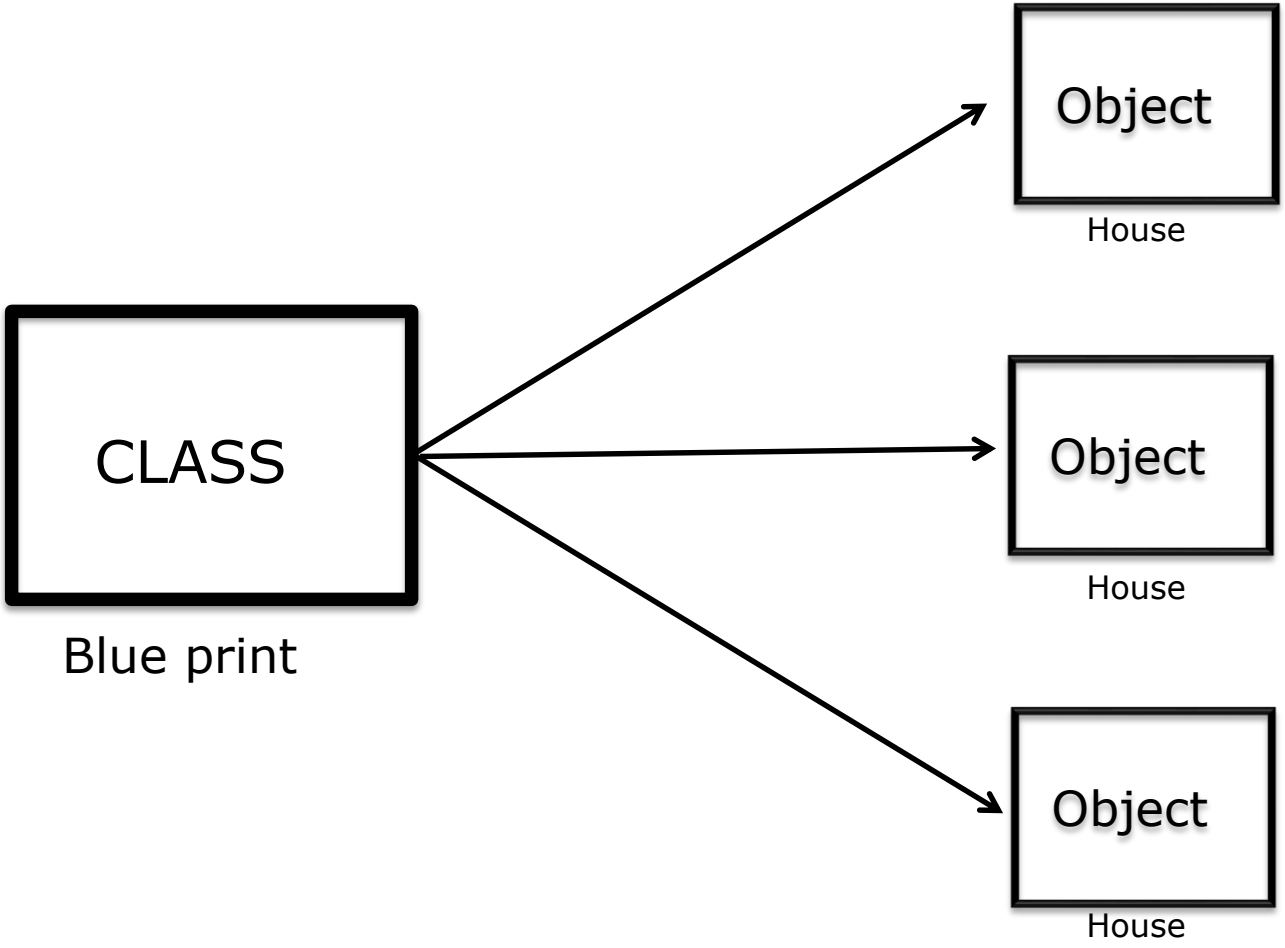
```
void calculator()
{

}
void scientific_calculator()
{

}
```

Object Oriented Programming Language

- Class
- Object

BLUE PRINT


bedroom
13x12
bath
mechanical
WH
kitchen
breakfast bar
covered patio
13x13
living
15.5x13
©61custom


CLASS
Blue print
Object
House
Object
House
Object
House

C Language

```
int  id;

void display()
{
  printf(id);
}
```

Data member

Member function

```
class Student
{
      int id;
      void display()
      {
        cout<<id;
      }
}
```

```
class Student
{
    int id;
    void display()
    {
     cout<<id;
    }
}
```

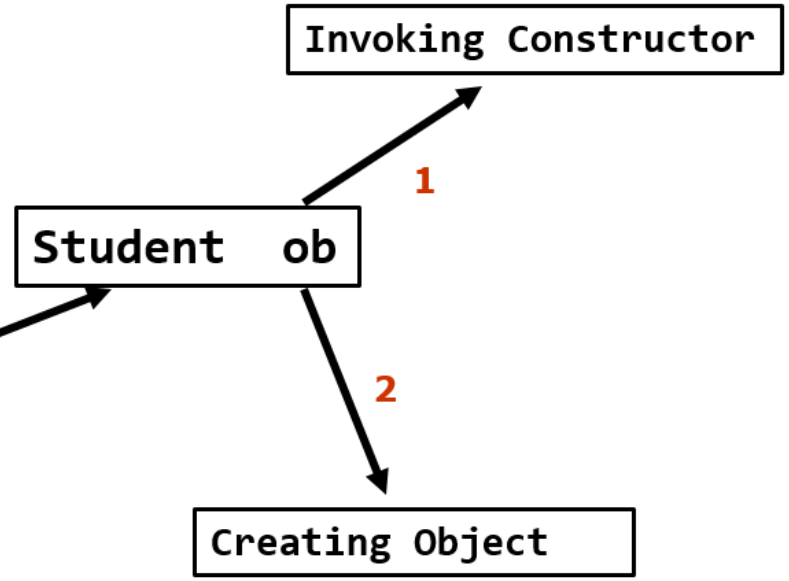
Blue Print

```
class Student
{
    int id;
    void display()
    {
     cout<<id;
    }
}
```

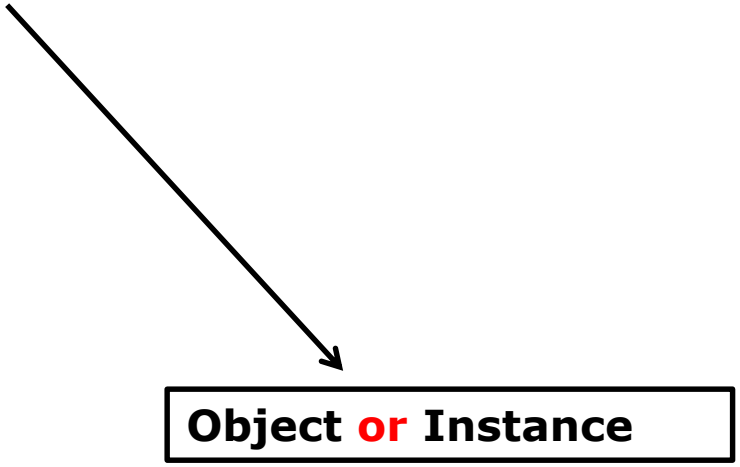
```
Student()
{
}
```

Blue Print

```
class Student
{
 int id;
  Student()
  {

  }
}
```



```
Student  ob;
```



```
int        x;
```

Pre-defined data type

```
Student  ob;
```

User-defined data type

```
class Demo
{
        int     x;
}
```

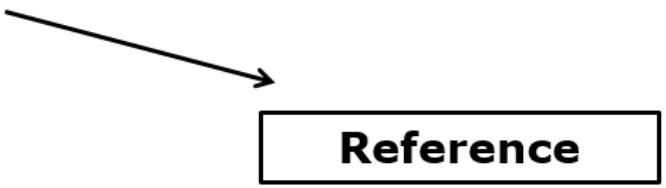
```
class Student
{
    int         rno;
    String      name;
}
```

```
class String
{

}
```

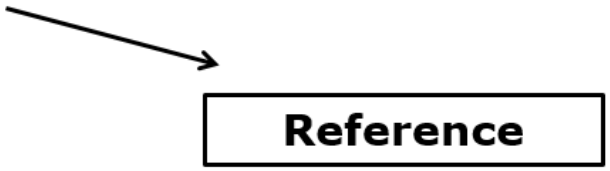
```
Student  s1;
```

Object creation of Student class

```
int        *x;
```




```
Student    *ob;
```



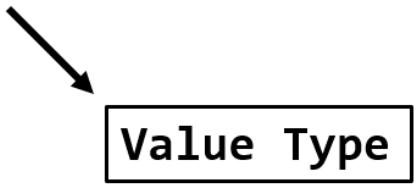
```
Student  ob;
```

Object creation

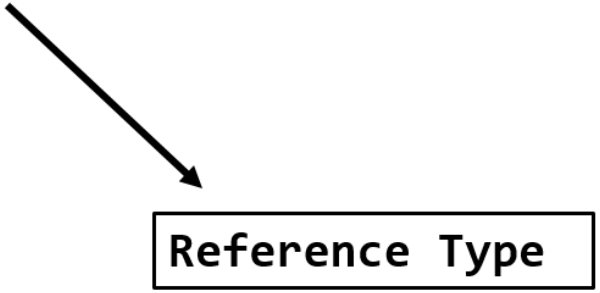
```
new Student()
```

Object creation

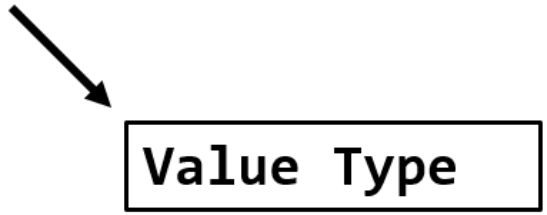
```
Student   ob;
```



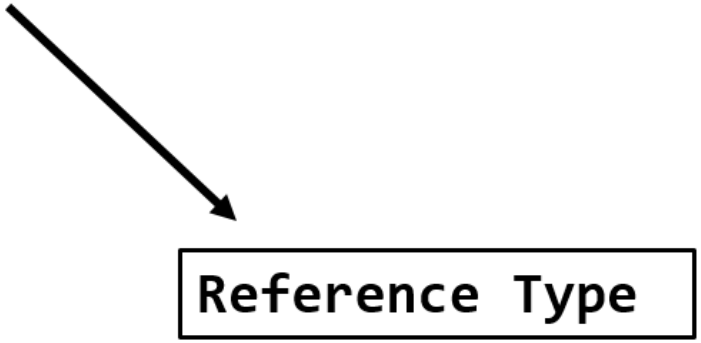

```
Student  *ptr = new Student();
```



```
Student   ob;
```




```
Student  *ptr = &ob;
```



```
Student   *ob1;
```

Reference creation in C++

Object creation in C++

```
Student    ob2;
```

Reference creation in JAVA

```
Student   *ob1;
```

Reference in C++

```
Student   ob2;
```

Reference in JAVA

```
Student  *ob;
```

Reference

```
new Student()
```

Object creation

Java

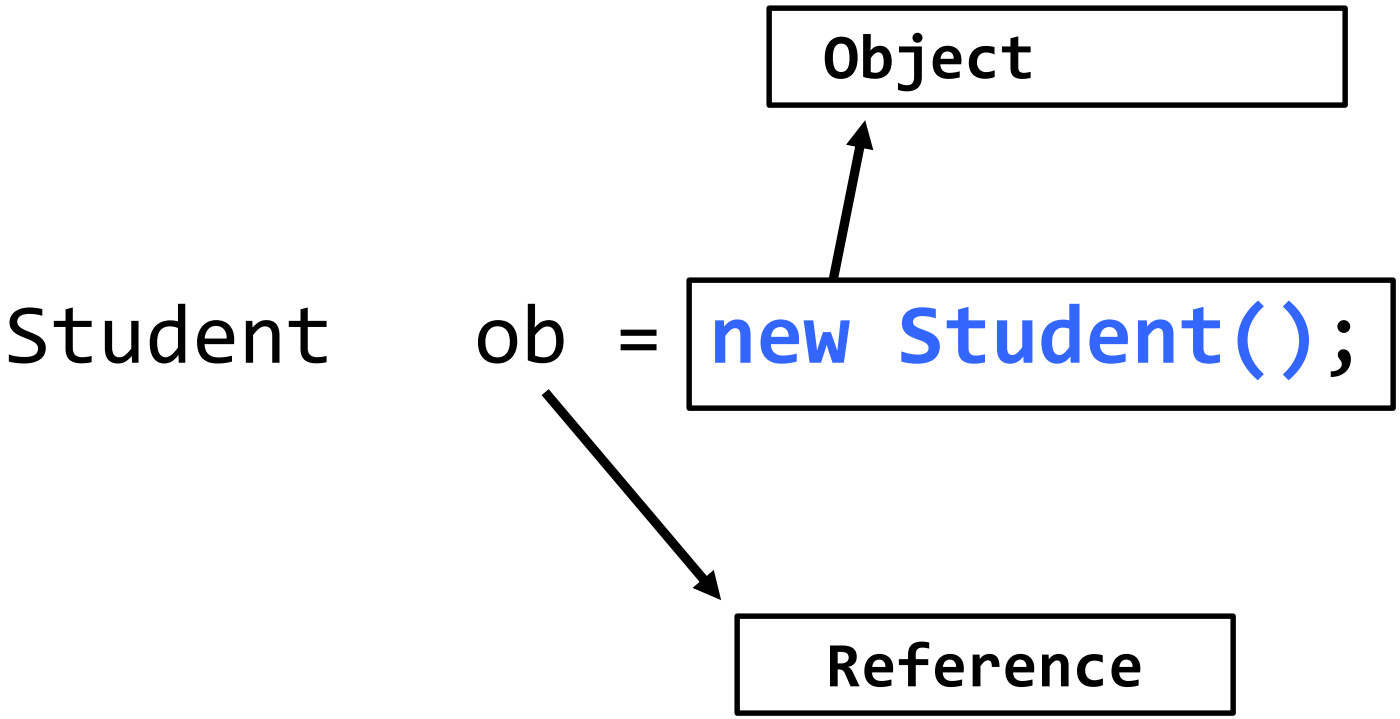
```
Student  ob;
```

Reference

```
new Student()
```

Object creation

```
Student    ob = new Student();
```


Object
Student ob = new Student();
Reference

Object

```
class Student
{
   int id;
   Student()
   {
   }
}
```

```
Student  ob = new Student();
```

Instance (or) Object

```
int    x;
float  y;


x = 100;
y = 205.f;


StudentDetails sd1;
StudentDetails sd2;

sd1 = new StudentDetails();
sd2 = new StudentDetails();
```

```
int    x  =  100;
float  y  =  20.5f;


StudentDetails sd1 = new StudentDetails();
StudentDetails sd2 = new StudentDetails();
```

```
class Employee
{
  int        id  =  100;
  Address    ob =  new Address();

}
```

```
class Address

{

}
```

```
class Student
{
    int id;
    void display()
    {
     cout<<id;
    }
}
```

Blue Print

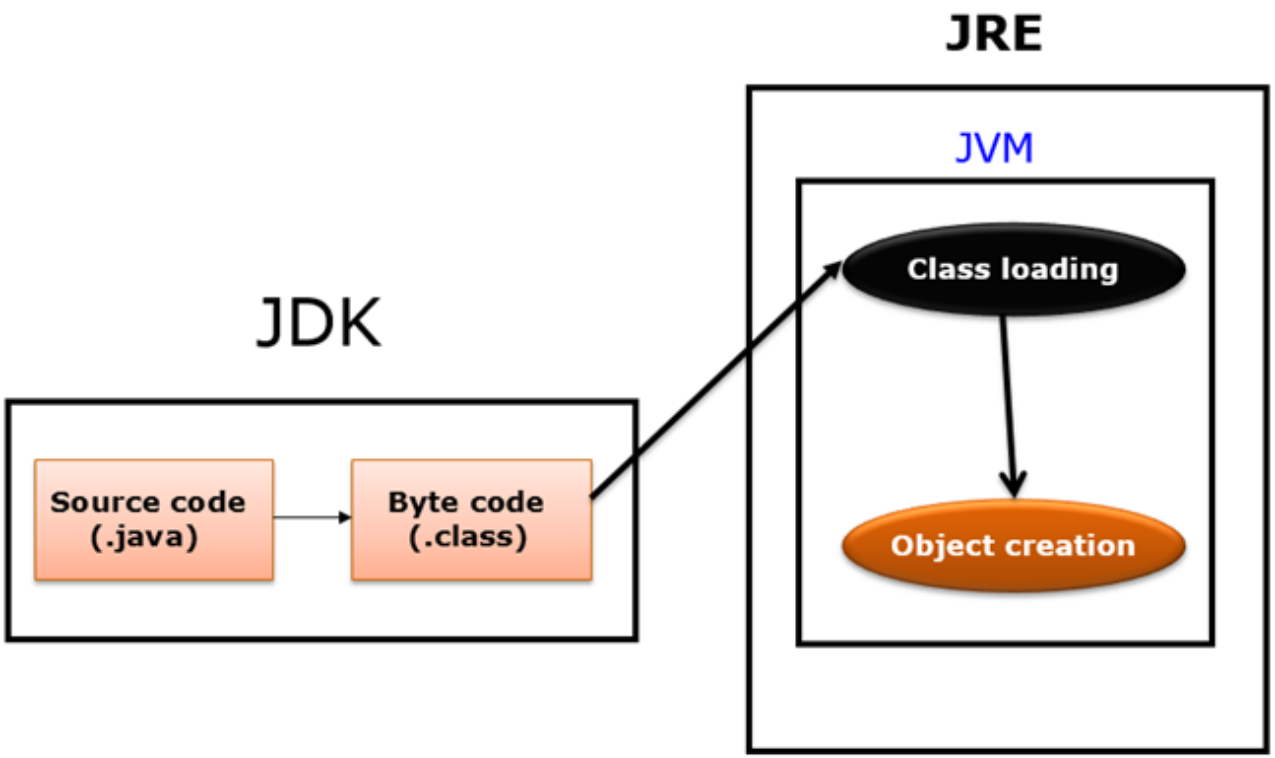
Types of Variables and Methods

- Instance variable and Instance Method (non-static).
- Class variable and Class Method (static).
- Local Variable

```
class Demo
{
    static int a;
            int b;
}
```

Static Member

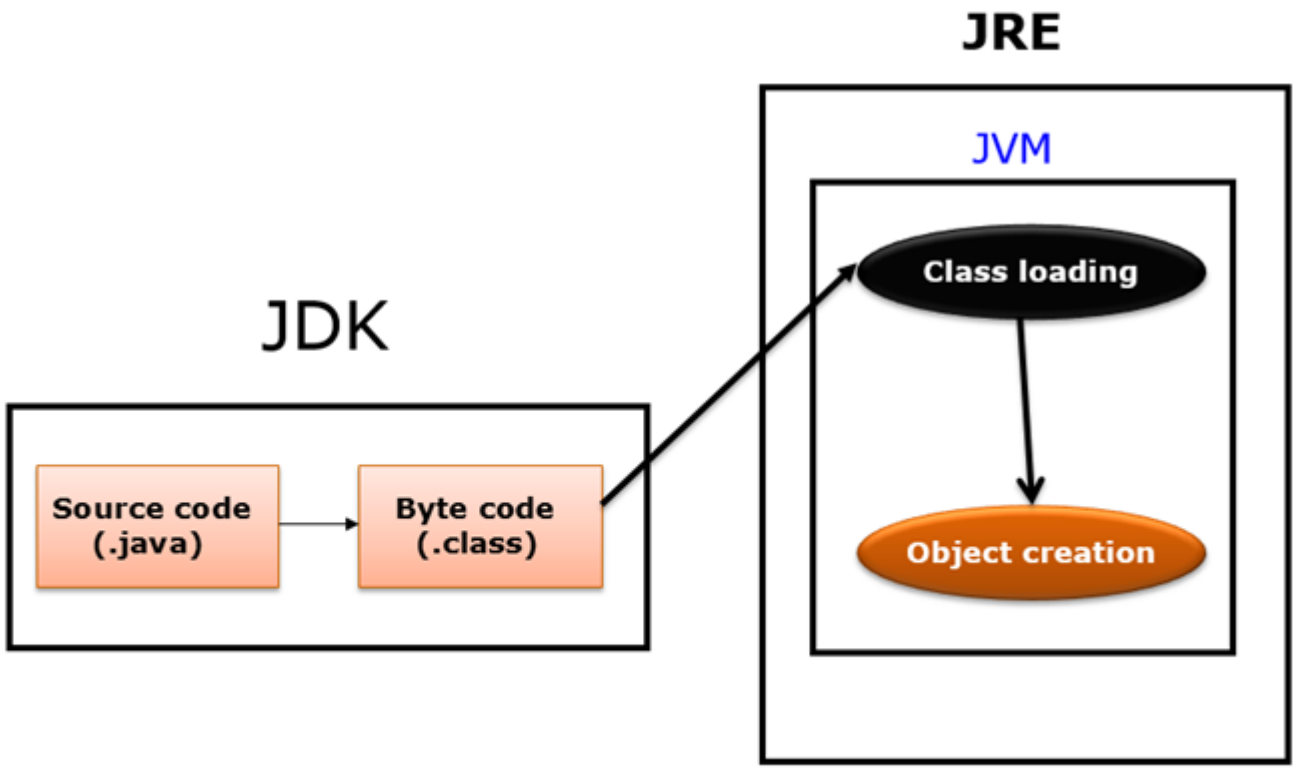
Non-Static Member


JRE
JVM
Class loading
JDK
Source code
(.java)
Byte code
(.class)
Object creation

```
class Demo
{
    static int a;
            int b;
}
```

Static Member

Non-Static Member



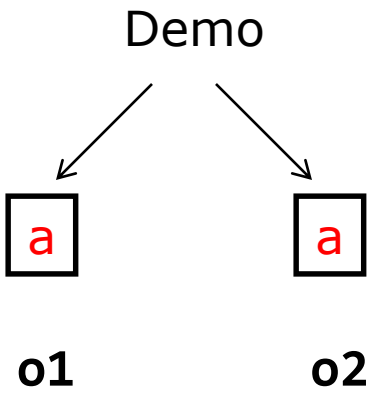
```
class Demo
{
    static int a;
           int b;
}
```

```
Demo.a = 5000;
```

```
Demo obj=new Demo();
obj.b=1000;
```

```
class Demo
{
    int a;
    void sum()
    {
     cout<<a;
    }
}
```

```
Demo o1=new Demo();
o1.a=1000;
o1.sum();

Demo o2=new Demo();
o2.a=3000;
o2.sum();
```



```
class Demo
{
    int a;
    void sum()
    {
     cout<<a;
    }
}
```

Instance Variable

Instance Method

```
class Demo
{
    static int a;
    static void sum()
    {
     cout<<a;
    }
}
```

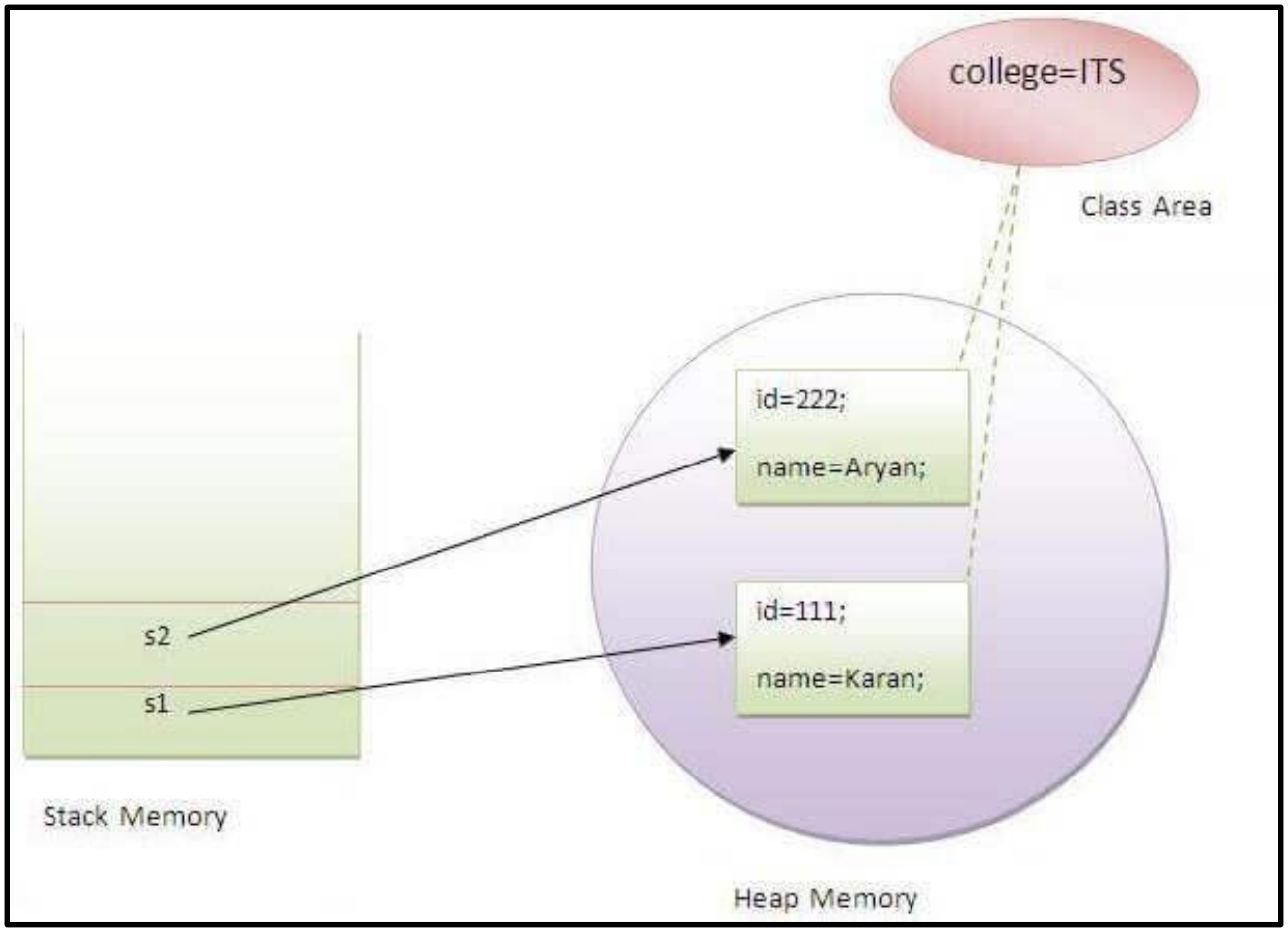
```
Demo.a = 5000;
Demo.sum();
```

```
class Demo
{
    static int a;
    static void sum()
    {
     cout<<a;
    }
}
```

Class Variable

Class Method

```
class Student{

     int rollno;
     String name;
     String college="ITS";

}


class Student{

            int rollno;
            String name;
     static String college="ITS";

}
```


college=ITS
Class Area
id=222;
name=Aryan;
id=111;
name=Karan;
s2
s1
Stack Memory
Heap Memory

```
class Demo
{
    void sum()
    {
        int a;
        cout<<a;
    }
}
```

Local Variable

Packages

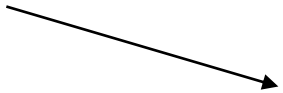
```
package yahoo;

class Registration
{
}
```

Sub packages

Classes

Interfaces

```
java.lang.*;
```



Object

System

```
System.out.println(“Welcome”);
```

System Class

```
class System
{
    int x;
    int y;
}
```

```
System ob1 = new System();

ob1.x = 100;
ob1.y = 200;
```

System Class

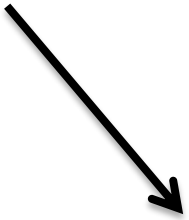
```
class System
{
    static int x;
    static int y;
}
```

```
System.x = 100;
System.y = 100;
```

```
class PrintStream
{
        void println(int);
        void println(String);
}
```

```
class System
{
  static PrintStream out = new PrintStream();
}
```

```
PrintStream  out = new PrintStream();
```



Object of PrintStream class

Class

Variable (static)

```
System.out
```

Class

Variable (static)

```
System.out
```

Object of PrintStream Class

```
class PrintStream
{
      void println(int);
      void println(String);
}
```

Class

Variable (static)

1

2

```
System.out.println("Welcome");
```

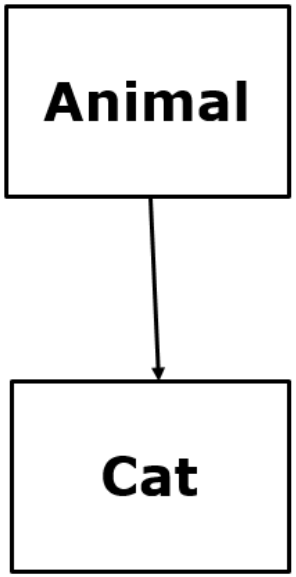
3

4

Method

Object of PrintStream Class

```
class Animal
{

}
class Cat extends Animal
{

}
```



```
class Demo
{

}
```

```
class Demo extends Object
{

}
```

```
import java.lang.*;

class Demo extends Object
{
    Demo()
    {
    }
}
```

```
class Demo
{

}
```

Demo.java

```
class Demo
{
  public static void main(String args[])
  {
     System.out.println(“Welcome”);
  }
}
```

Java Program Execution Steps

1. Type the Java Program in Notepad and save in any user directory

   eg. E:\Test\Demo.java

2. Go to Command Prompt and change the directory location

   eg. cd E:\Test

3. Set Path to Java Installed Directory

   E:\>Test> set path = c:\Program Files\Java\jdk1.8.0_111\bin

4. Compile and Run the Java Program

   E:\>Test> javac Demo.java

   E:\>Test> java Demo

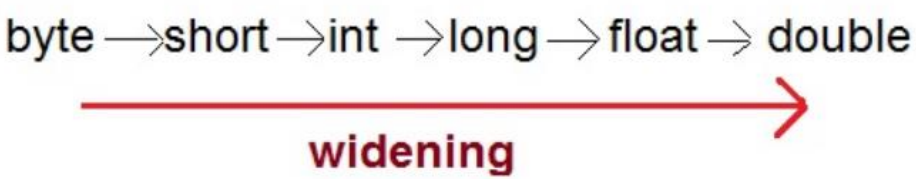
Type Casting

Type	Size (in bits)	Range
byte	8	-128 to 127
short	16	-32,768 to 32,767
int	32	-2^{31} to $2^{31}-1$
long	64	-2^{63} to $2^{63}-1$
float	32	1.4e-045 to 3.4e+038
double	64	4.9e-324 to 1.8e+308
char	16	0 to 65,535
boolean	1	true or false

Type Casting

In Java, type casting is classified into two types,

- Widening Casting(Implicit)



- Narrowing Casting(Explicitly done)

```
int  i  = 100;

long a1  =  i;

long a2  =  (long) i;
```

```
int i = 100;

long l = i;

float f = i;
```

```
int i = 100;

long l = (long) i;

float f = (float) i;
```

```
int    a = 100;

char  b = (char)a;
```

✓

```
int    a = 100;

char  b = a;
```

✗

```
double d = 100.04;

long l = (long)d;        //  explicit type casting required

int i = (int)l;          //  explicit type casting required
```

```
class A
{

}
class B extends A
{

}
```

A

```
new A();
```

A B

```
new B();
```

Reference Creation

```
class A
{

}
class B extends A
{

}
```

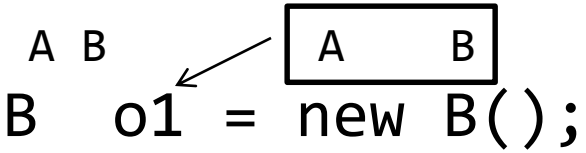
A

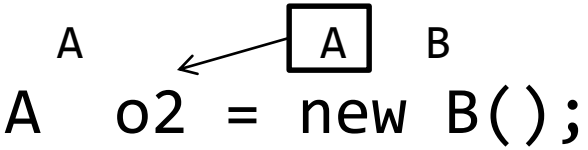
```
A    ob1;
```

A B

```
B    ob2;
```

```
class A
{

}
class B extends A
{

}
```





```
class A
{
}
class B extends A
{
}
```

```
A s1 = new B();

B s2 = (B)s1;
```

Up-casting

```
A s1 = new B();
```

Down-casting

```
B s2 = (B)s1;
```

A class **Object** may be under
same-class reference or **base**-class reference

```
B  r1 = new B();

A  r2 = new B();
```

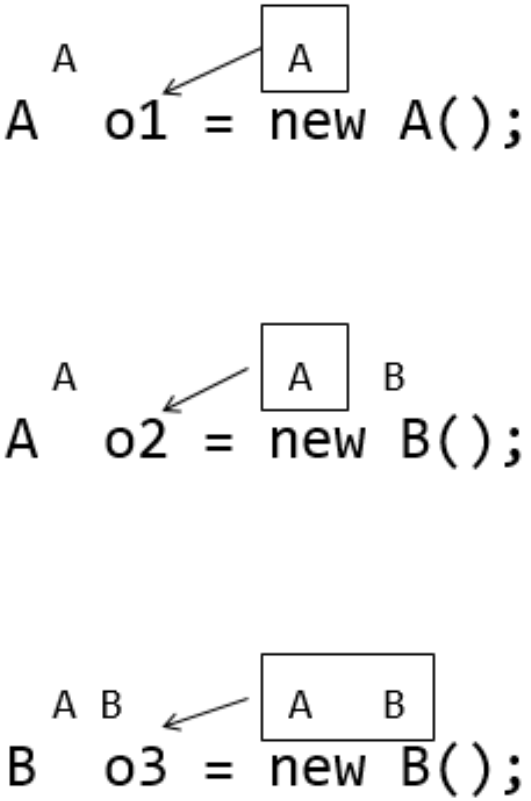
A class **Reference** may contain
same-class Object or **sub**-class Object

```
A  r1 = new A();

A  r2 = new B();
```

```
A   r1 = new B();
```

```
A   r2 = new B();
```

```
class A
{

}
class B extends A
{

}
```



```java
class A{

    int x;
    void test(){
        System.out.println(" X  : "+x);
    }

}

class B extends A{

    int y;
    void show(){
        System.out.println(" X  : "+x+" Y  : "+y);
    }

}
```

```
B  ob1 = new B();

A  ob2 = ob1;

A  ob3 = new B();

A  ob4 = (A)ob1;

A  ob5 = (A)new B();
```

```
B ob1 = new B();

A ob2 = ob1;

A ob3 = (A)ob1;

B ob4 = ob2;

B ob5 = (B)ob2;
```

```
class A
{
}
class B
{
}
class C
{
}
```

O A

```
new A();
```

O B

```
new B();
```

O C

```
new C();
```

Tightly Coupled

```
class A
{
}
class B
{
}
class C
{
}
```

```
A o1 = new A();

B o2 = new B();

C o3 = new C();
```

```
class A
{
}
class B
{
}
class C
{
}
```

```
                    O   A
Object o1 = new A();

                    O   B
Object o2 = new B();

                    O   C
Object o3 = new C();
```

Loosely Coupled

```
class A
{
}
class B
{
}
class C
{
}
```

```
                    O   A
Object o1 = new A();

                    O   B
Object o2 = new B();

                    O   C
Object o3 = new C();
```

```
class A
{
}
class B
{
}
class C
{
}
```

```
                    O   A
Object s1 = new A();

A      s2 = (A)s1;
```

https://www.javatpoint.com/java-tutorial

- Java Object Class
  - Java OOPs Concepts
  - Naming Convention
  - Object and Class
  - Constructor
  - static keyword
  - this keyword
- Java Inheritance
  - Inheritance(IS-A)
  - Aggregation(HAS-A)

- Java Polymorphism
  - Method Overloading
  - Method Overriding
  - Covariant Return Type
  - super keyword
  - Instance Initializer block
  - final keyword
  - Runtime Polymorphism
  - Dynamic Binding
  - instanceof operator
- Java Abstraction
  - Abstract class
  - Interface
  - Abstract vs Interface

Methods Overloading

```
class Demo
{
      void add()
      {
      }
      void add(int x)
      {
      }
      void add(int x,int y)
      {
      }
}
```

There are two ways to overload the method in java

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

```
class Demo
{
     int x;
     void add()
     {
     }
     void add(int x)
     {
            this.x = x;
     }
}
```

```
Demo o1=new Demo();
o1.add();
o1.add(50);
```

```
void calculation();
```

Method Declaration

```
void calculation()
{
  int a=200;
  int b=400;
  S.o.p(a+b);
}
```

Method Declaration

Method Definition

```
void calculation();
```

Abstract Method

```
void calculation()
{
  int a=200;
  int b=400;
  S.o.p(a+b);
}
```

Non-Abstract Method

or

Concrete Method

```
class Demo{
            void calculation(){

            }
}

-----------------------------------------

abstract class Demo{

            abstract void calculation();

}
```

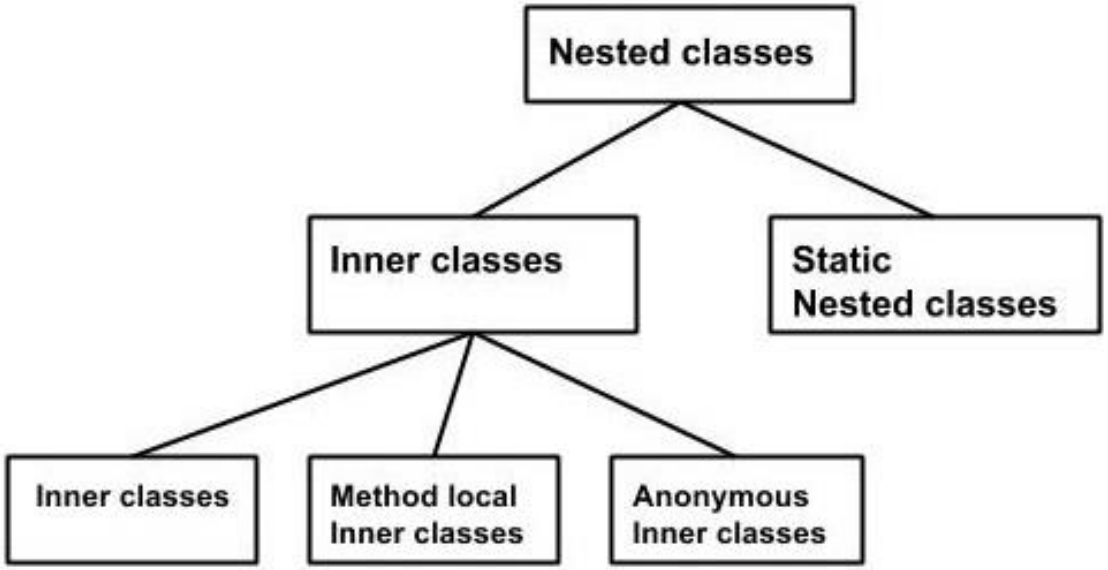
Methods Overriding

```
class A
{
     void add()
     {
         System.out.println(“Hai”);
     }
}
class B extends A
{

}
```

```
class A
{
     void add()
     {
       System.out.println(“Hai”);
     }
}
class B extends A
{
     void add()
     {
        System.out.println(“Welcome”);
     }
}
```

Inner classes

```
class Car{
        class Benz{
        }
}
```


Nested classes
Inner classes
Static
Nested classes
Inner classes
Method local
Inner classes
Anonymous
Inner classes

```
class A{

}

class B extends A {

}

class Demo1 {
        public static void main(String ss[]){

        }
}
```

A.class
B.class
Demo1.class

```
class A {

}

class B extends A {

}

class Demo1 {
        public static void main(String args[]) {

                new A() { };
                new B() { };


        }
}
```

A.class

B.class

Demo1.class

Demo1$1.class

Demo1$2.class

```
new A() { };   //class A is extended into Anonymous Class.

new A() { };   // Anonymous Object of Anonymous Class.

new A() { };   // Sub-class Object of class A
```

```
new A(){
        void add()
        {
                System.out.println(" Anonymous Class ");
        }
}.add();
```

```
A s1 = new A(){
        void add()
        {
                System.out.println(" Anonymous Class ");
        }
};
```

Constructor

```
class A
{
    int x;

    A()
    {
    }

    A(int y)
    {
      this.x=y;
    }
}
```

```
A o1 = new A();

A o2 = new A(40);
```

Super Keyword in Java

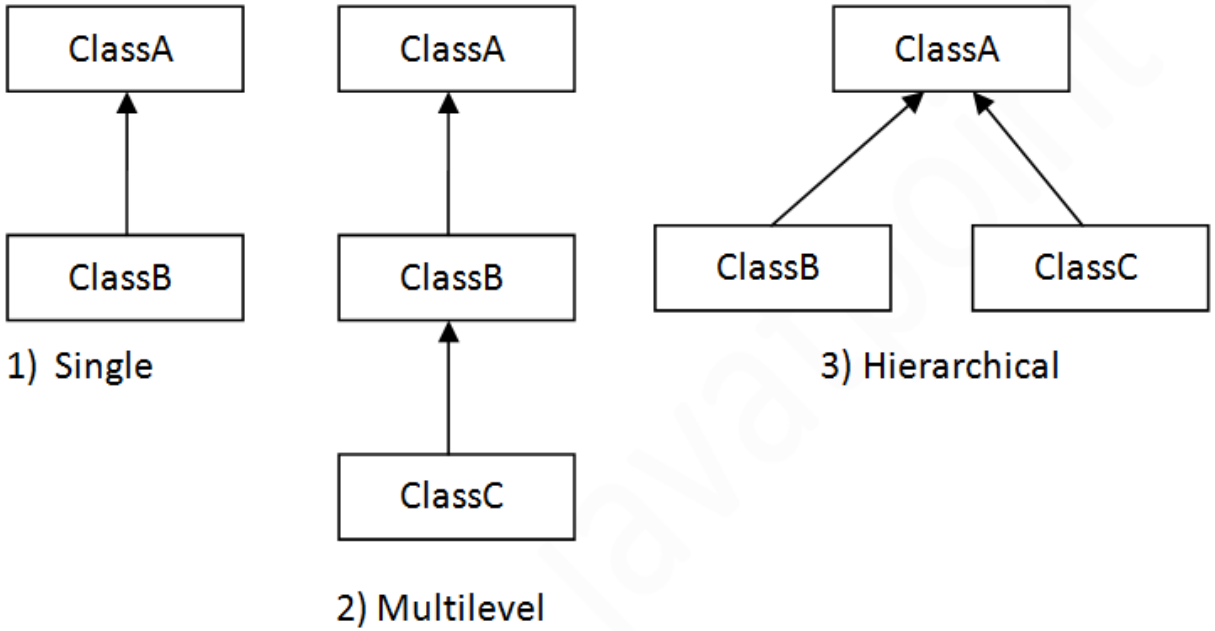
Usage of Java super Keyword

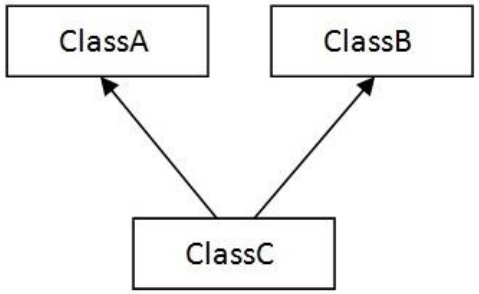
1. super can be used to refer immediate parent class instance variable.
2. super can be used to invoke immediate parent class method.
3. super() can be used to invoke immediate parent class constructor.

Inheritance

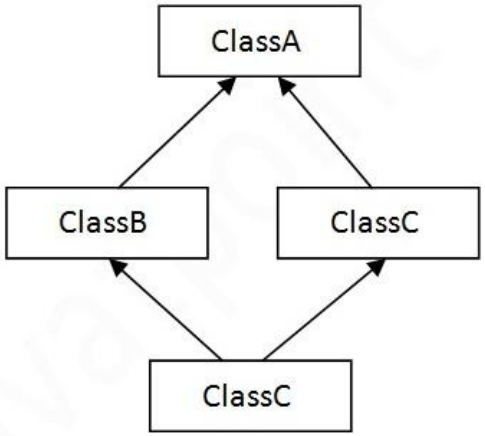
- Single
- Multilevel
- Hierarchal
- Multiple
- Hybrid

Inheritance





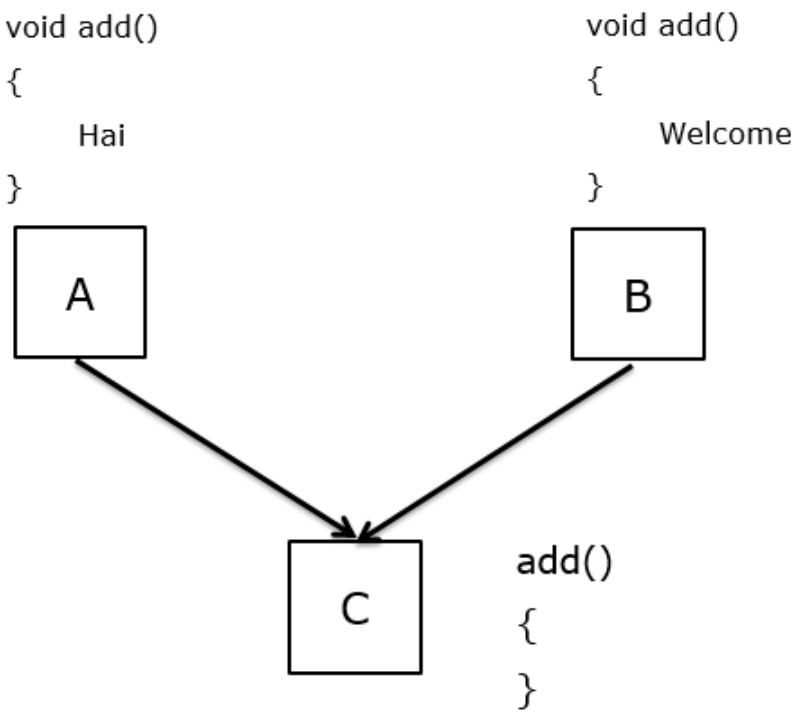
4) Multiple

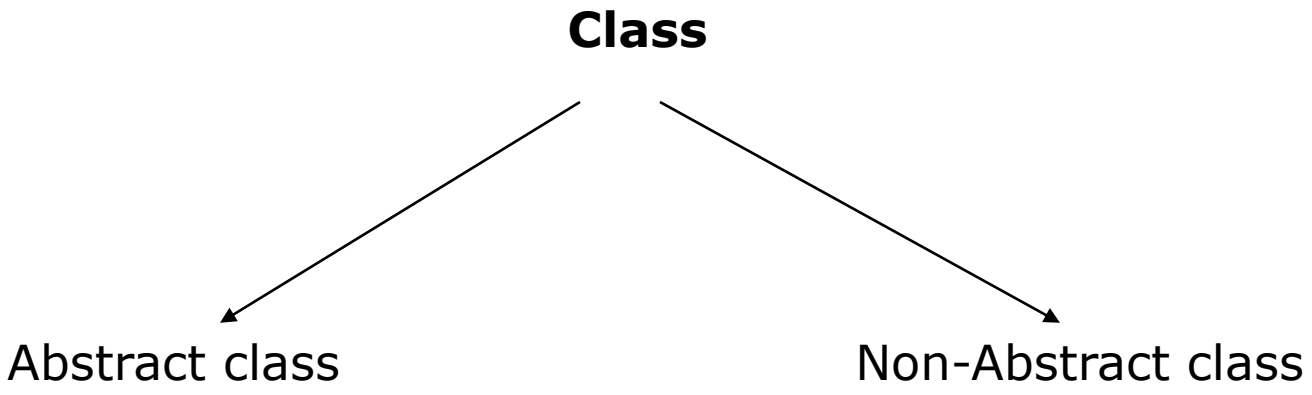


5) Hybrid

.

Multiple Inheritance




Class
Abstract class
Non-Abstract class

```
class Demo{
            void calculation(){

            }
}

-----------------------------------------

abstract class Demo{

            abstract void calculation();

}
```

```
interface Car
{

}
```

Interface is a type of Abstract Class

```
void calculation();
```

Abstract Method

```
void calculation()
{
  int a=200;
  int b=400;
  S.o.p(a+b);
}
```

Non-Abstract Method

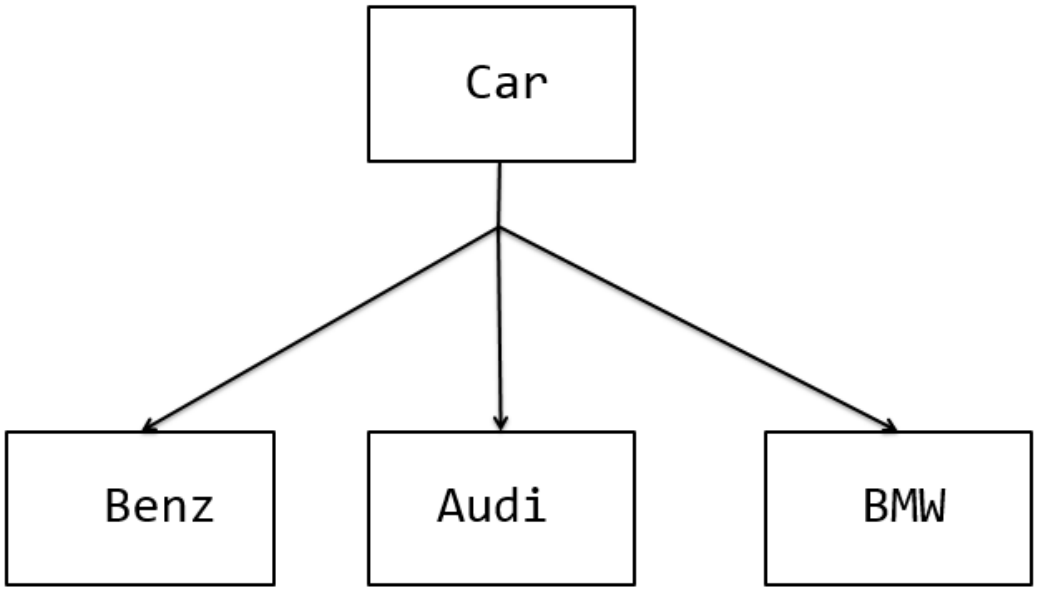
or

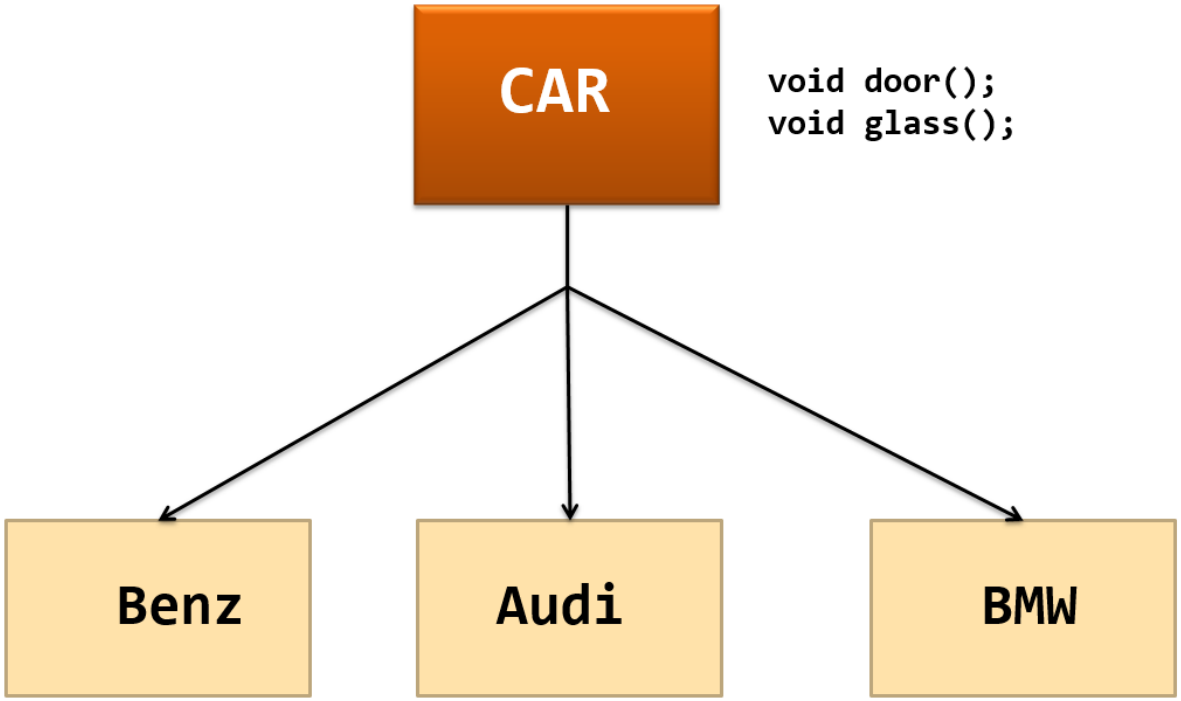
Concrete Method

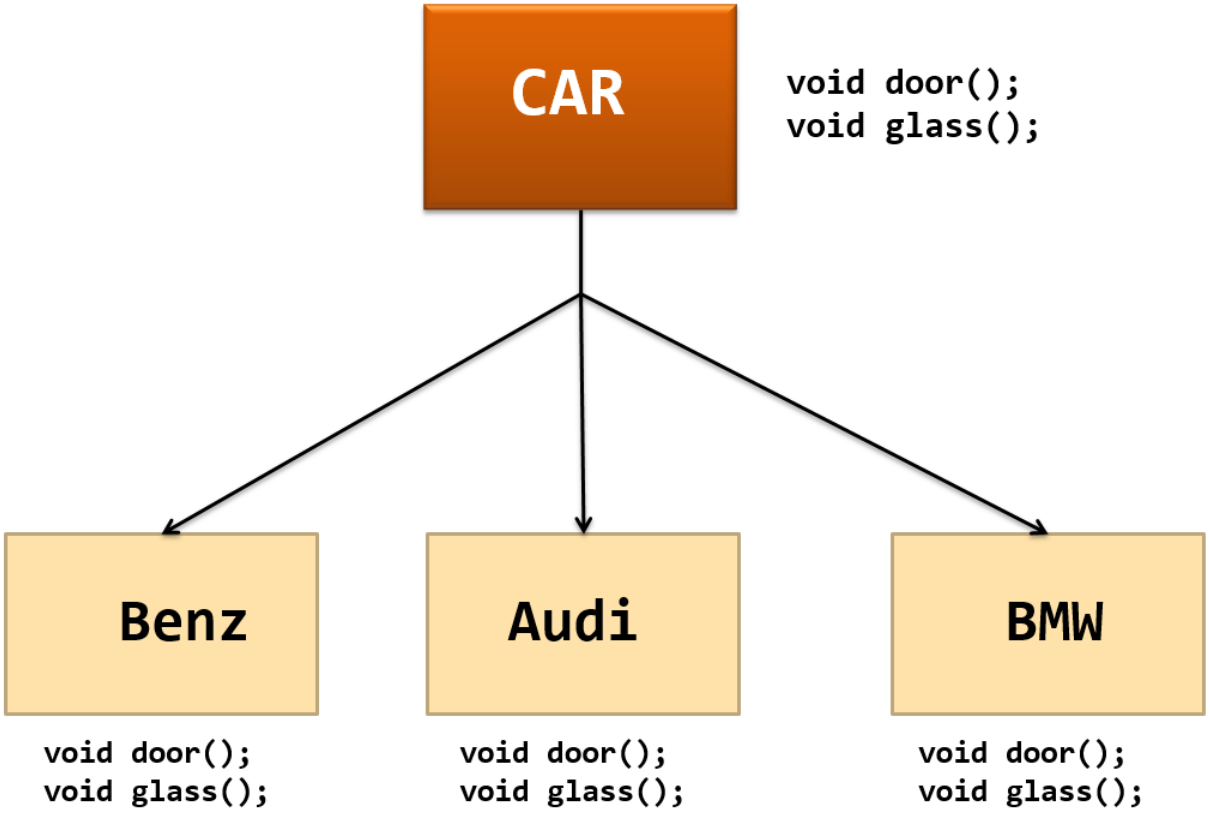
```
abstract void calculation();
```

```
abstract class Car
{
     abstract void door();
     abstract void glass();
}
```

```
abstract class Car
{
      abstract void door();
      abstract void glass();
      void wheel()
      {
        System.out.println(“Wheel”);
      }
}
```


Car
Benz
Audi
BMW

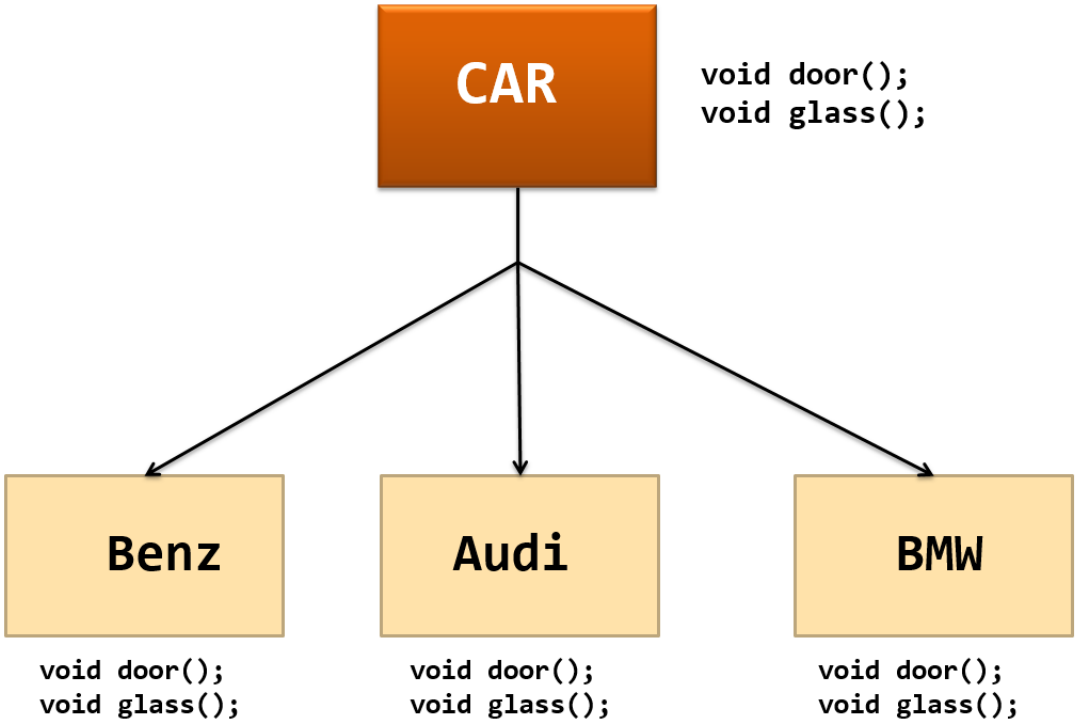

CAR
void door();
void glass();
Benz
Audi
BMW

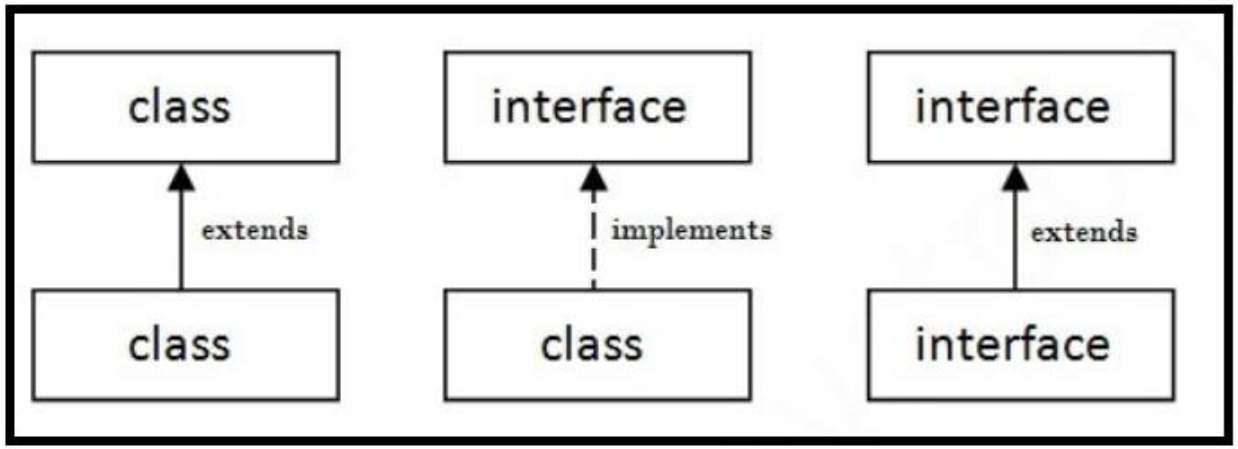

CAR
void door();
void glass();
Benz
Audi
BMW
void door();
void glass();
void door();
void glass();
void door();
void glass();

```
abstract class Car
{
      abstract void door();
      abstract void glass();
      void wheel()
      {
        System.out.println(“Wheel”);
      }
}
```

```
class Lancer  extends  Car
{
        void door()
        {
              System.out.println("Lancer door");
        }
        void glass()
        {
              System.out.println("Lancer glass");
        }
}
```

Abstract Class	Interface
Contains abstract and non-abstract methods.	Contains only abstract methods.


CAR
void door();
void glass();
Benz
void door();
void glass();
Audi
void door();
void glass();
BMW
void door();
void glass();

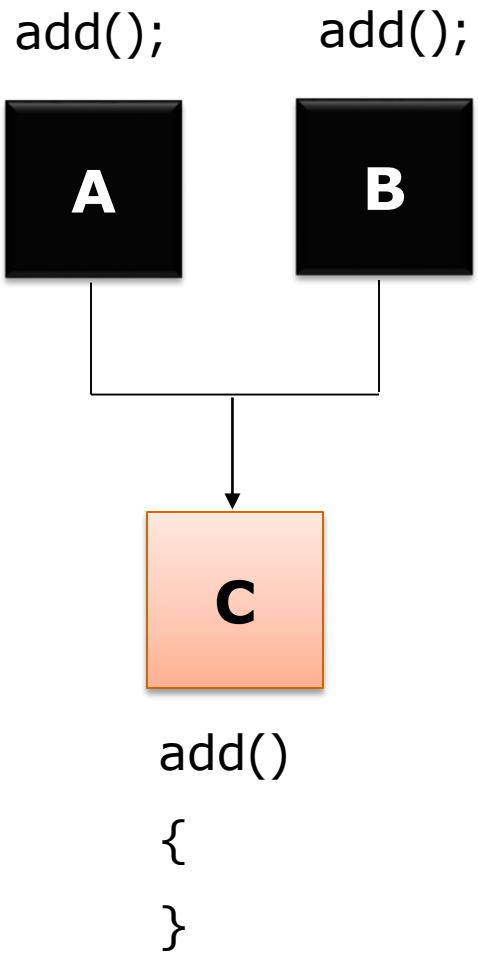

class
interface
interface
extends
implements
extends
class
class
interface

```
interface Car
{
        void door();
        void glass();
}
```

```
interface Car
{
     public abstract   void door();
     public abstract   void glass();
}
```

```
interface Car
{
            void door();
            void glass();
}
```

```
class Lancer  implements  Car
{
    public void door()
    {
     System.out.println("Lancer door");
    }
    public void glass()
    {
     System.out.println("Lancer glass");
    }
}
```


add();
add();
A
B
C
add()
{
}

```
interface Mail
{
      void register();
      void validation();
}
```

new Mail();

```
abstract class Car
{
      void door();
      void glass();
}
```

new Car();

```
interface Mail
{
      void register();
      void validation();
}

abstract class Car
{
      void door();
      void glass();
}
```

new Yahoo();

new Benz();

```
Mail ob1 = new Mail();
```

```
Car c1  = new Car();
```

```
Mail ob1 = new Yahoo();
```

```
Car c1 = new Benz();
```