



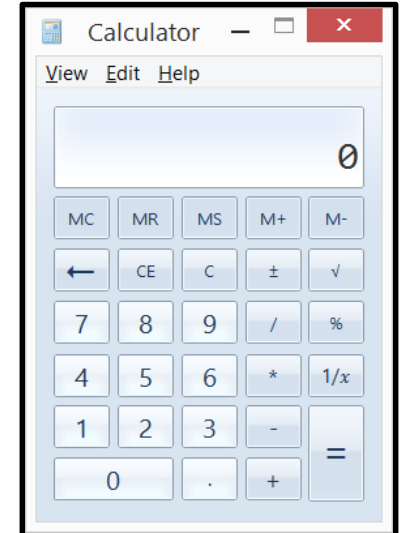
facebook

Log in to Facebook

Email address or phone number

Password

Log In



C

C++

C#

JAVA

Python

Perl

```
void calculator()
```

```
{
```

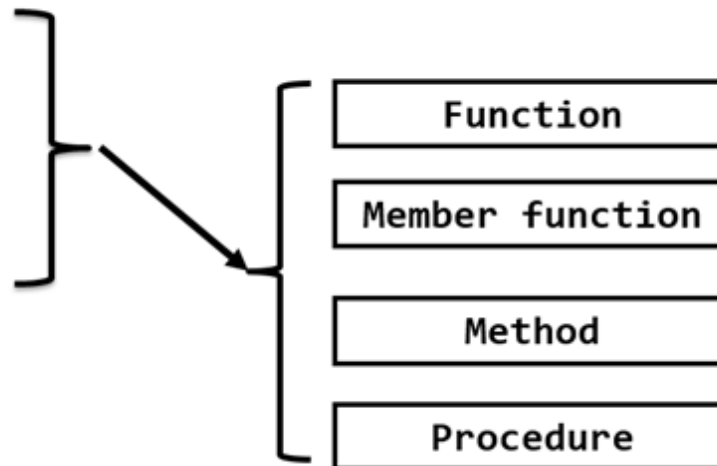
```
}
```

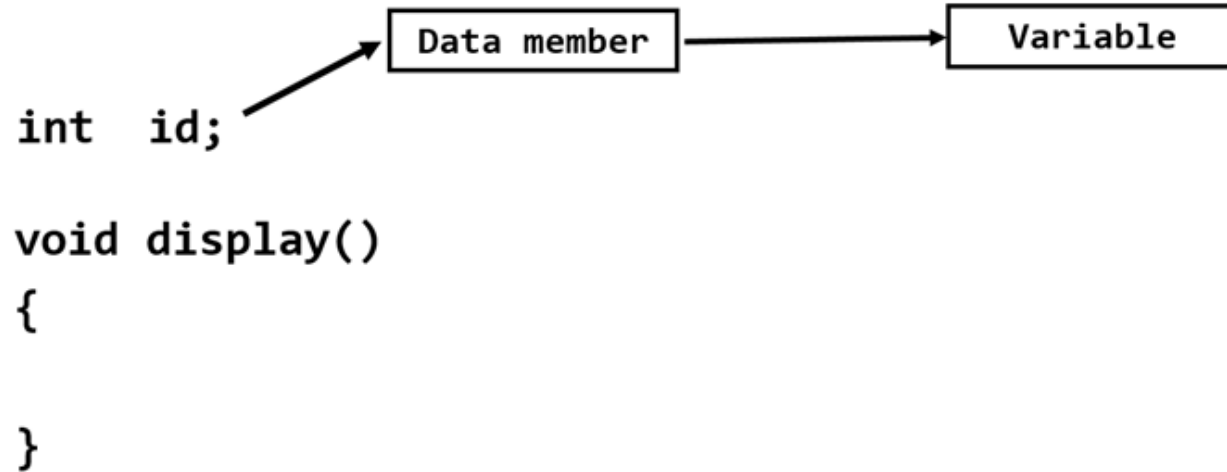
```
void scientific_calculator()
```

```
{
```

```
}
```

```
void display()  
{  
  
}
```





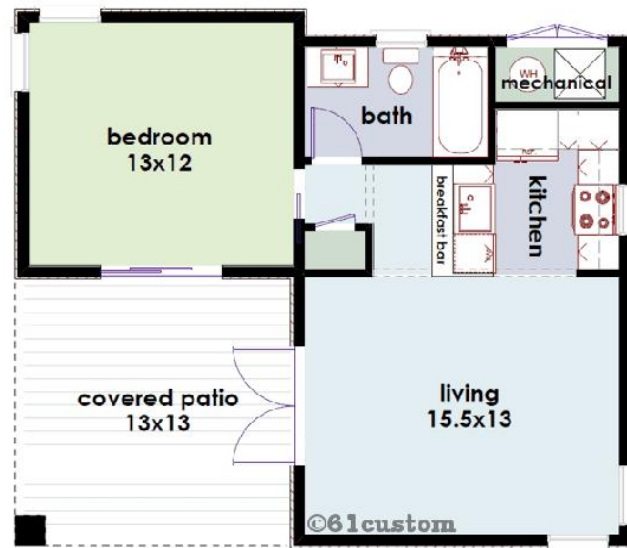
-
1. Turmeric powder - 100 gms
 2. Sugar - 1 kg
 3. Jaggery - 1/2 kg
 4. Idli rice/Boiled rice/Salem rice - 5-7 kgs
 5. Steamed rice or Raw rice/Sona masoori - 5-7 kgs
 6. High quality raw rice for Pongal - 1 kg
 7. Dosa rice (optional) - 2 kgs
 8. Basmati rice - 1 to 2 kgs



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

```
void display()
{

}
```




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

    void display()
    {

    }
}
```

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

    void display()
    {

    }
}
```

```
StudentDetails student1, student2, student3;

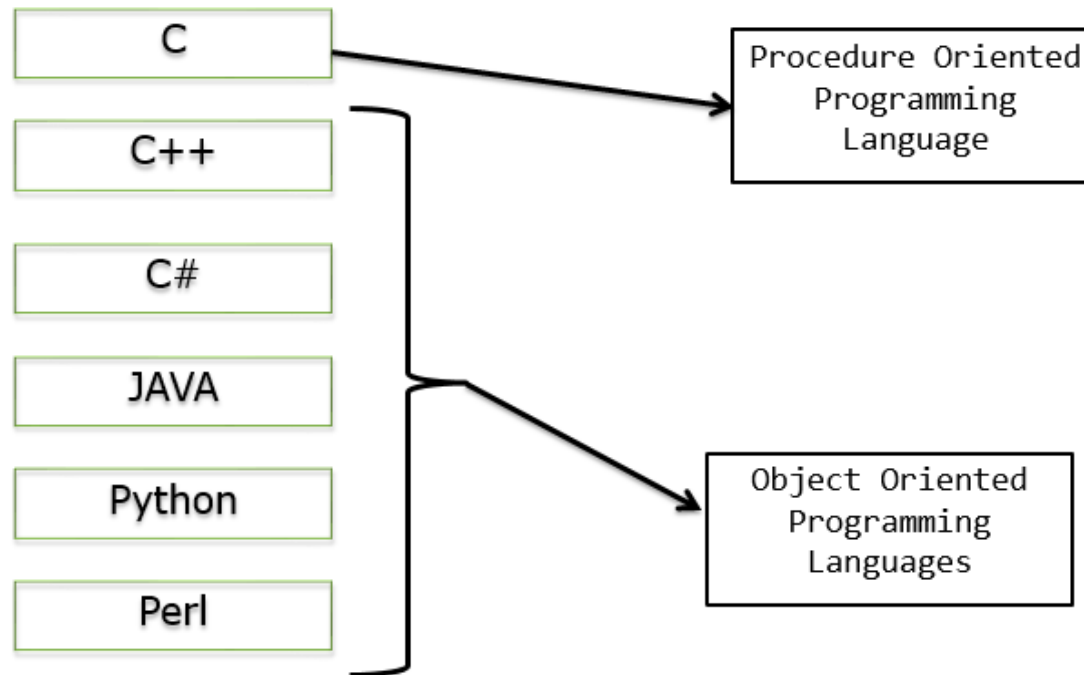
student1.rollno = 10;
student1.name    = "Ramesh";
student1.city    = "Salem";

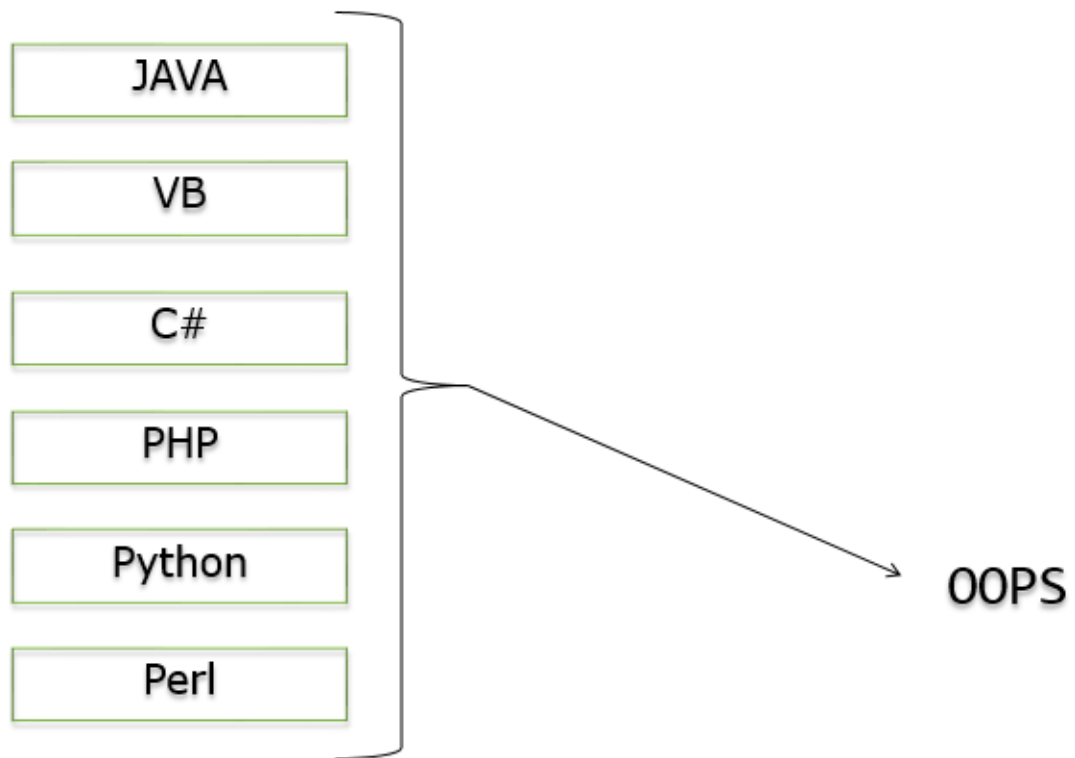
student2.rollno = 20;
student2.name    = "Ganesh";
student2.city    = "Trichy";

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

```
int x, y, z;  
void display()  
{  
  
}
```

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





Java Features

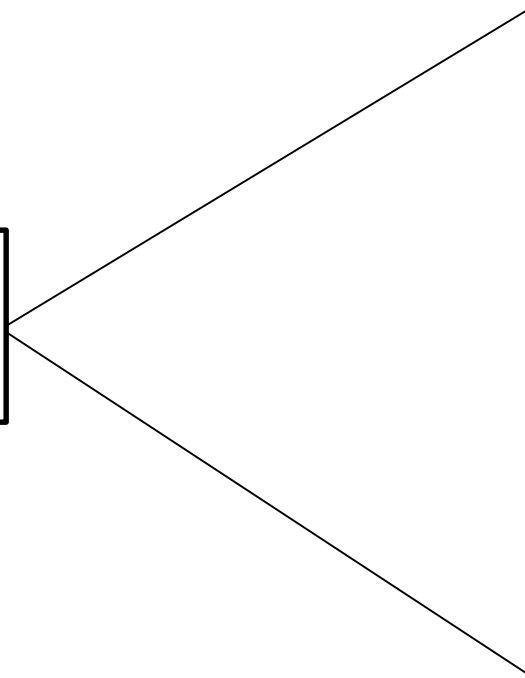
- Platform Independence
- Object Oriented Programming Language

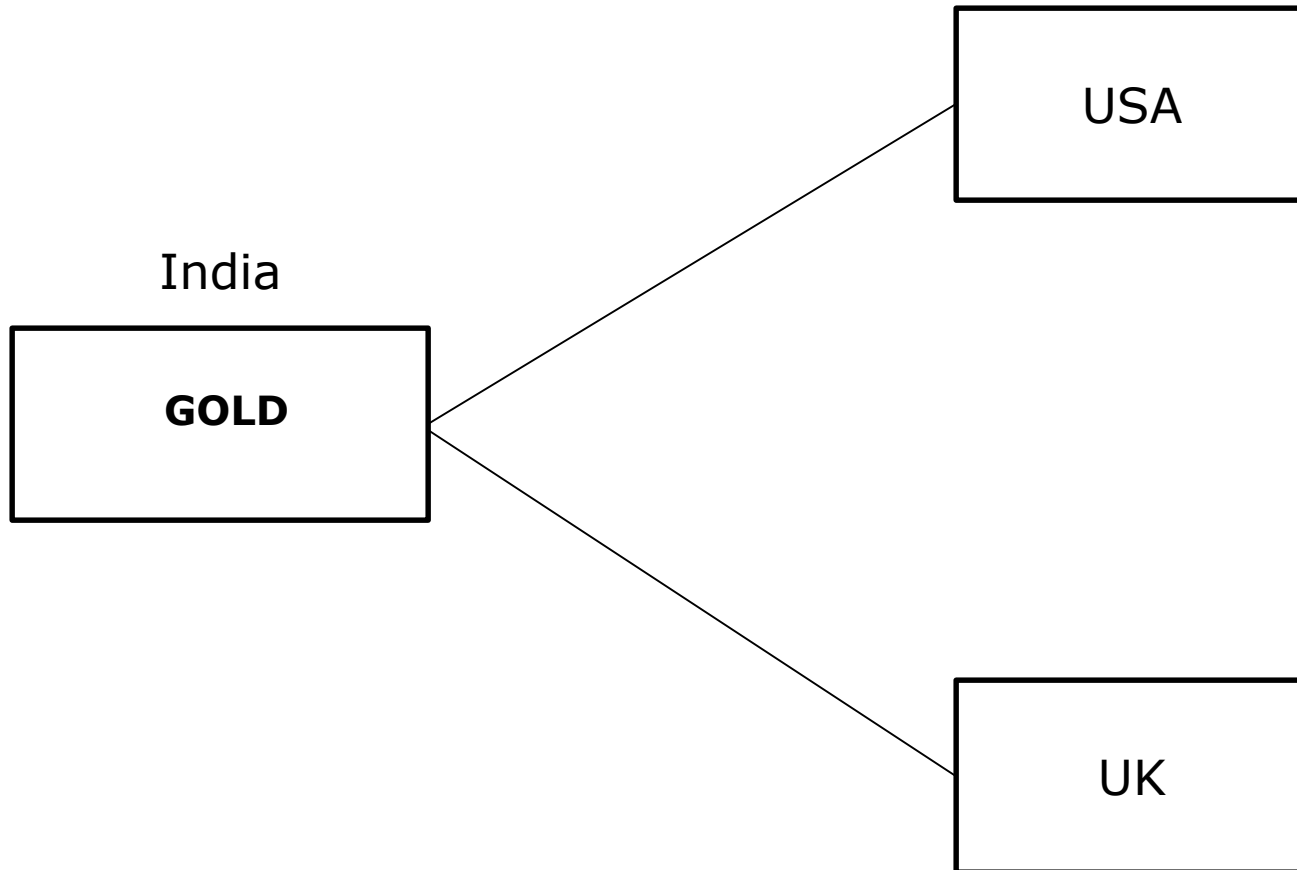
India

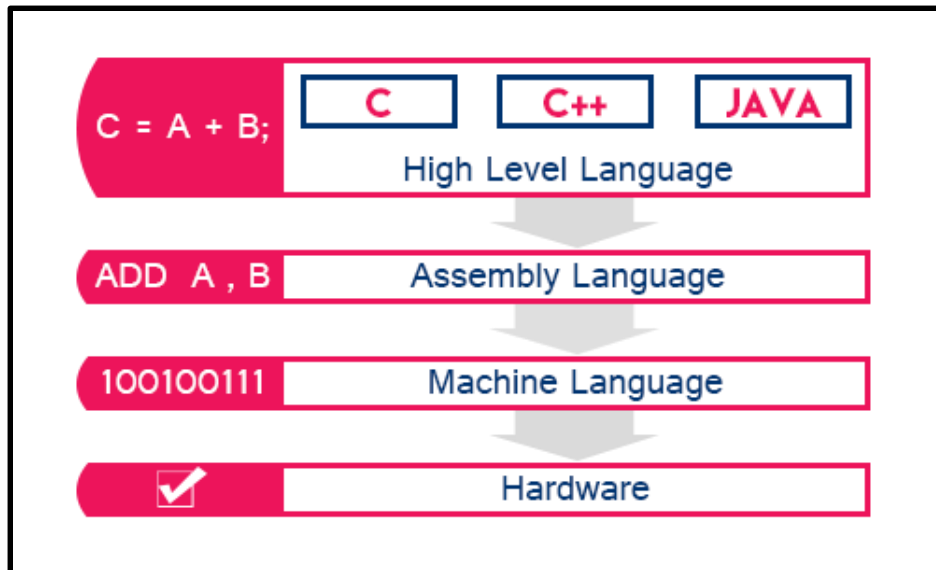
Rupee

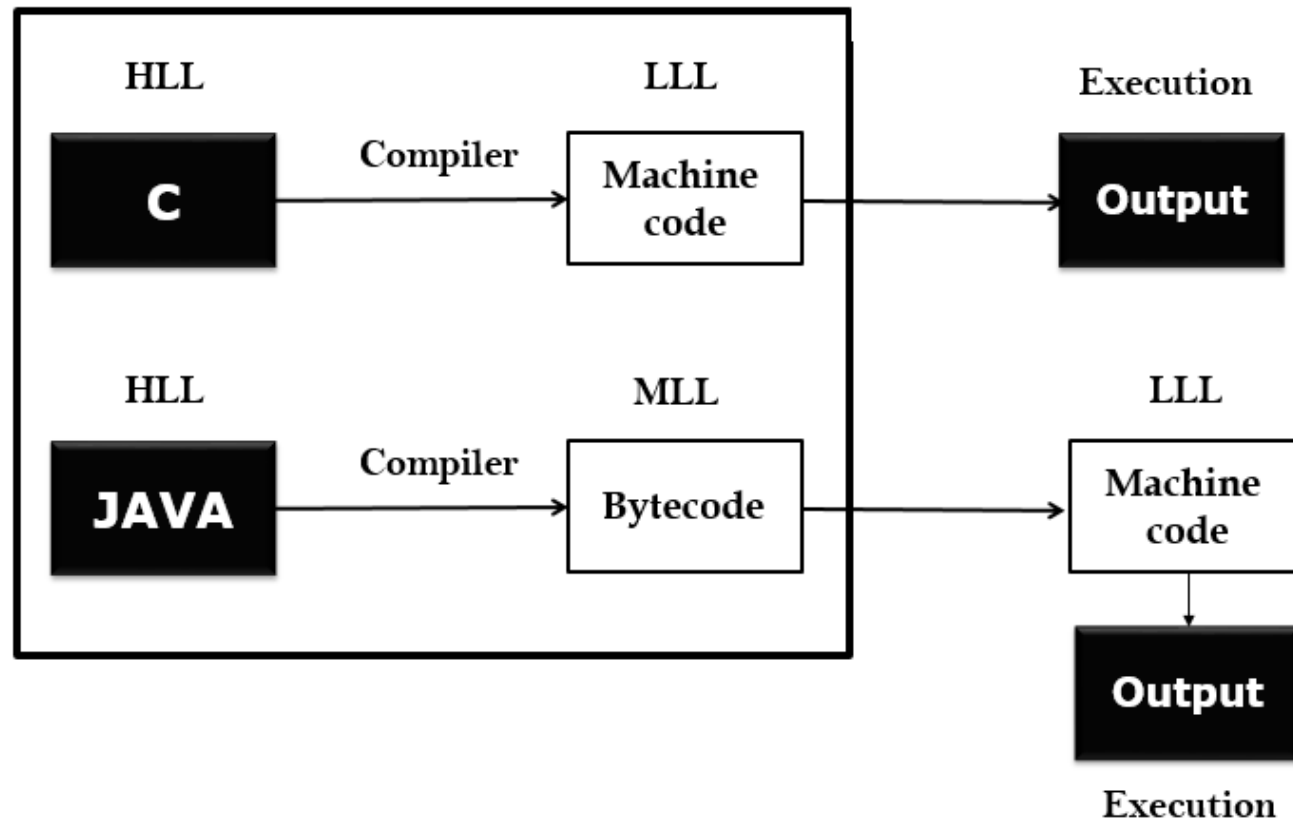
USA

UK









JDK

JRE

JVM

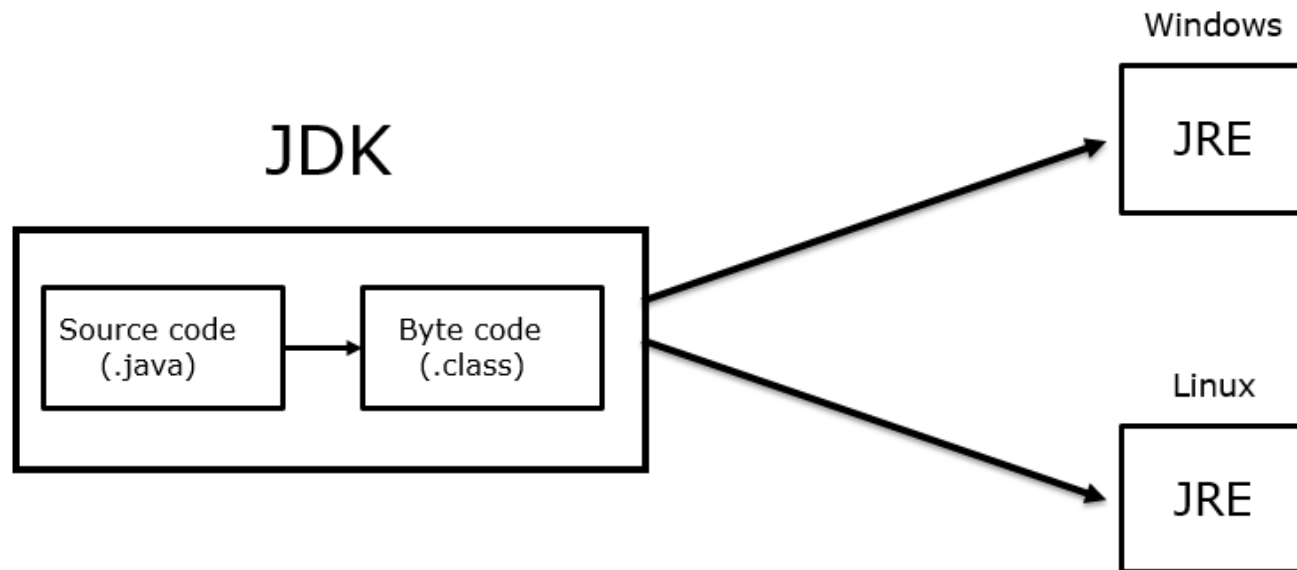


JDK → Java Development Kit

JRE → Java Runtime Environment

JVM → Java Virtual Machine

Platform Independence



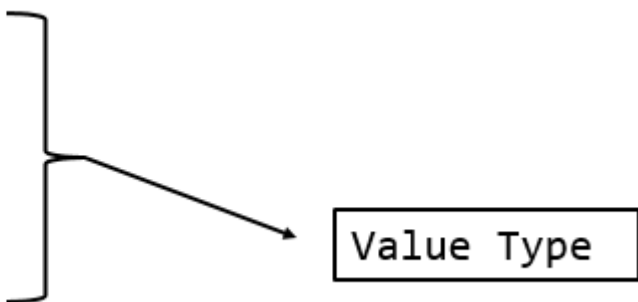
Programming Basics

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

Data Types

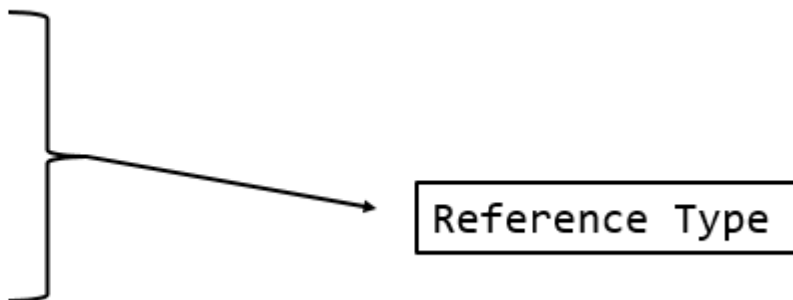
- ✓ boolean
- ✓ char
- ✓ byte
- ✓ short
- ✓ int
- ✓ long
- ✓ float
- ✓ double

```
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';`

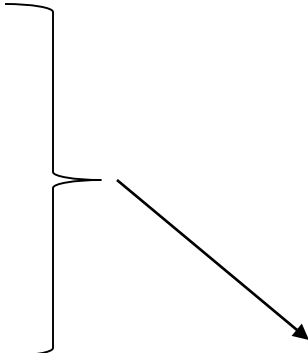
} Mapping

```
int id;
```



Data Member

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



Member Function

```
void calculator()
```

```
{
```

```
}
```

```
void scientific_calculator()
```

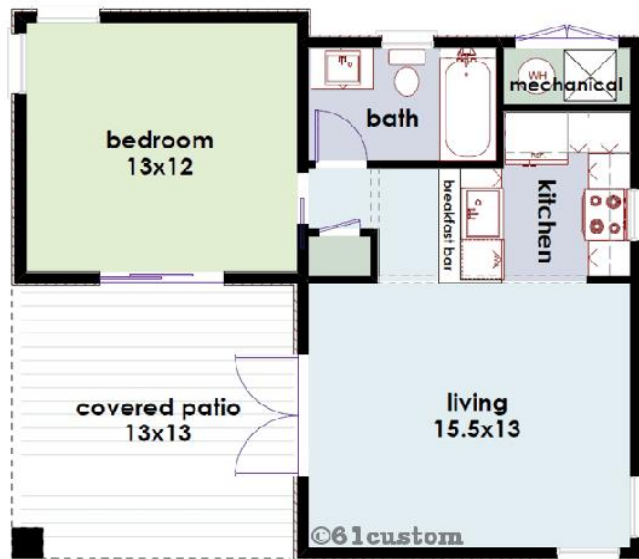
```
{
```

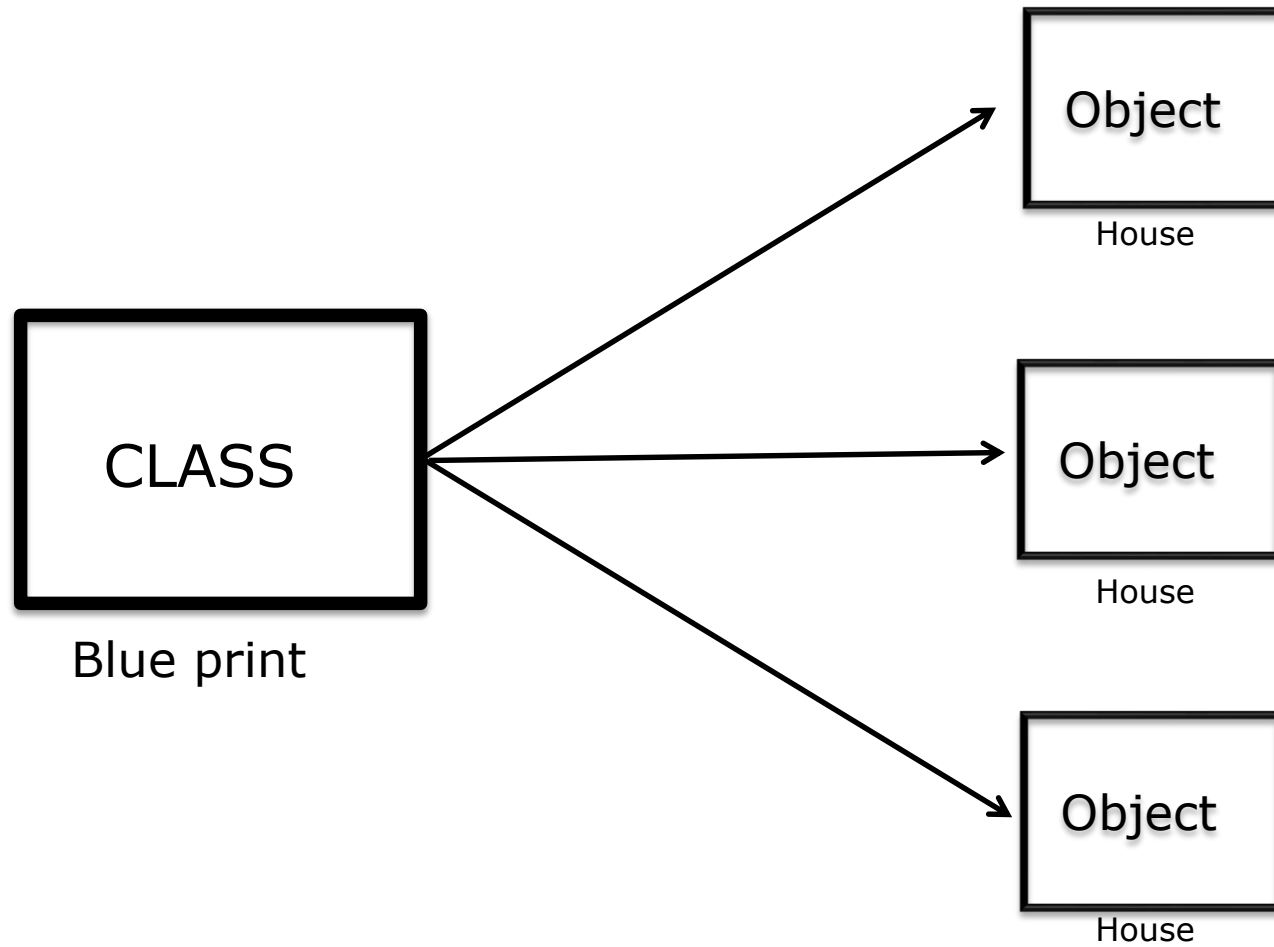
```
}
```

Object Oriented Programming Language

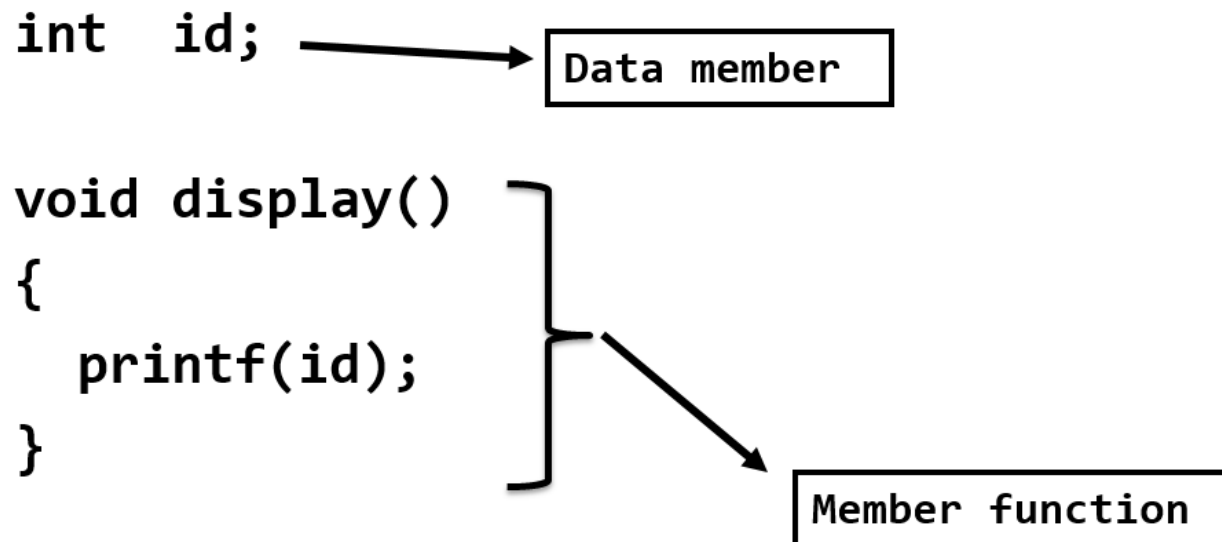
- Class
- Object

BLUE PRINT





C Language



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

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



Blue Print

```
class Student
```

```
{
```

```
    int id;
```

```
Student()
```

```
{  
}
```

```
void display()
```

```
{
```

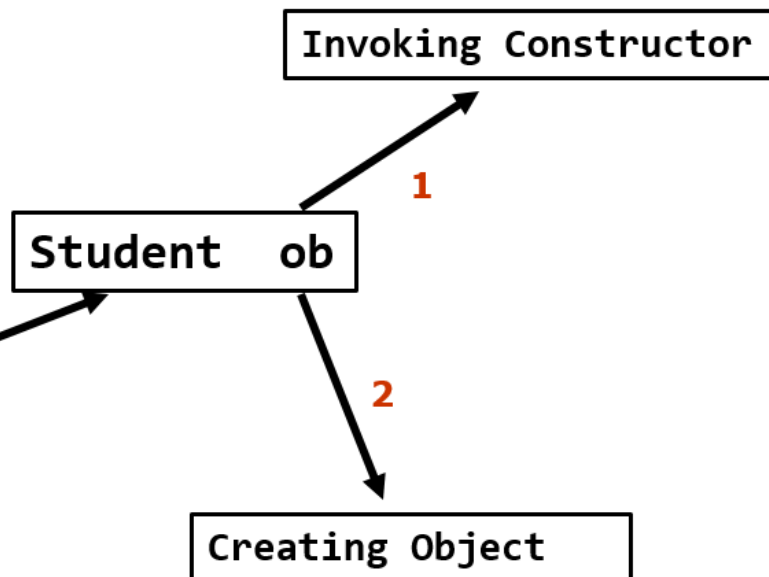
```
    cout<<id;
```

```
}
```

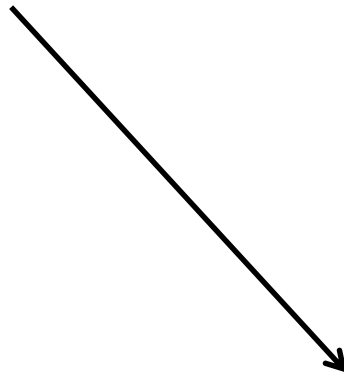
```
}
```

```
Blue Print
```

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

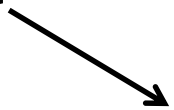


Student ob;



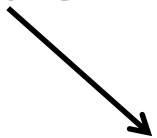
Object or Instance

int x;



Pre-defined data type

Student ob;



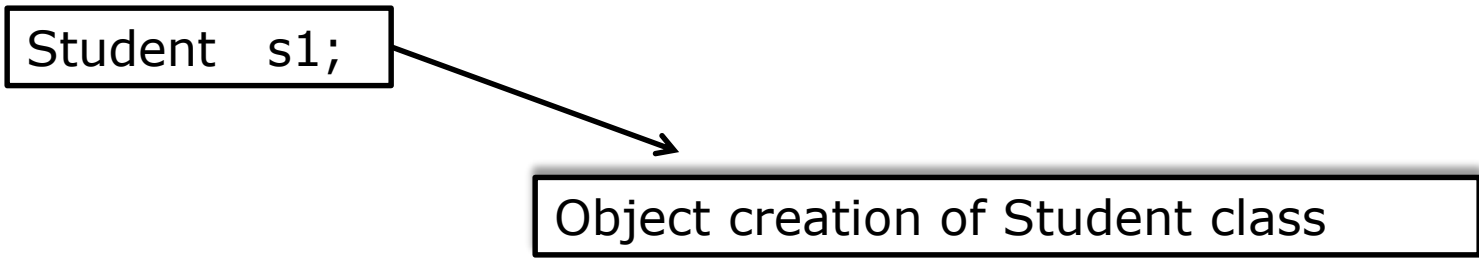
User-defined data type

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

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

```
class String
{
}
```

```
Student s1;
```

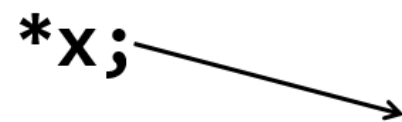


Object creation of Student class

The diagram illustrates the process of object creation. A box containing the code 'Student s1;' has an arrow pointing to a box containing the text 'Object creation of Student class'.

int

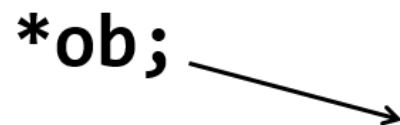
***x;**



Reference

Student

***ob;**



Reference

Student ob;



Object creation

new Student()



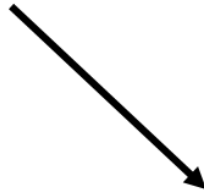
Object creation

Student ob;



Value Type

Student *ptr = new Student();



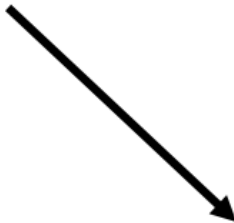
Reference Type

Student ob;



Value Type

Student *ptr = &ob;



Reference Type

Student *ob1;

Reference creation in C++

Student ob2;

Object creation in C++

Reference creation in JAVA

Student *ob1;



Reference in C++

A diagram illustrating a pointer variable in C++. The text 'Student *ob1;' is shown. An arrow points from the asterisk in the code to a rectangular box containing the text 'Reference in C++'.

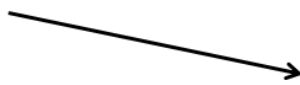
Student ob2;



Reference in JAVA

A diagram illustrating a reference variable in Java. The text 'Student ob2;' is shown. An arrow points from the variable name 'ob2' to a rectangular box containing the text 'Reference in JAVA'.

Student ***ob;**



`Reference`

`new Student()`



`Object creation`

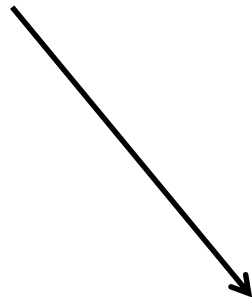
Java

Student ob;



Reference

new Student()



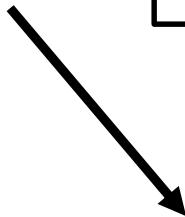
Object creation

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

Student

ob = new Student();

Object



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;
```

```
    }
```

```
}
```

The diagram illustrates the concept of a class as a blueprint. A large right-facing curly brace groups the entire class definition, from 'class Student' down to the final closing brace. An arrow points from the middle of this brace to a rectangular box labeled 'Blue Print'.

```
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;
```

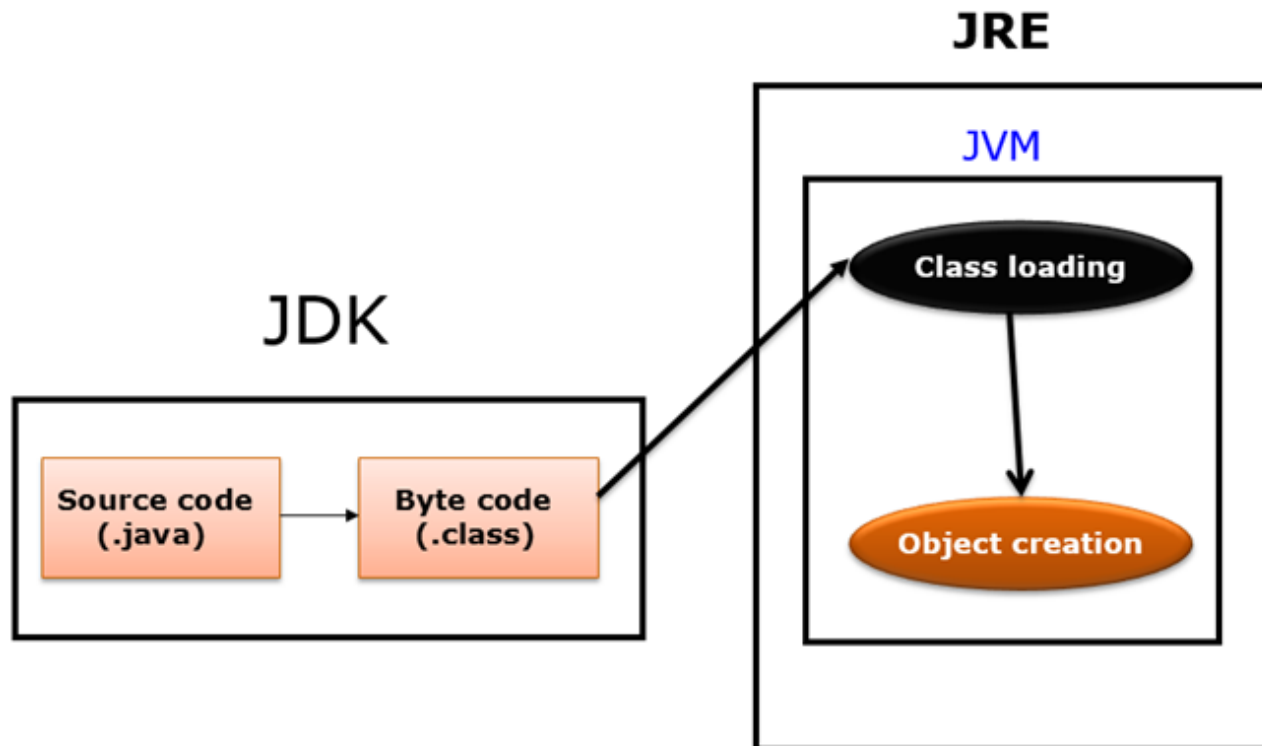
```
    int b;
```

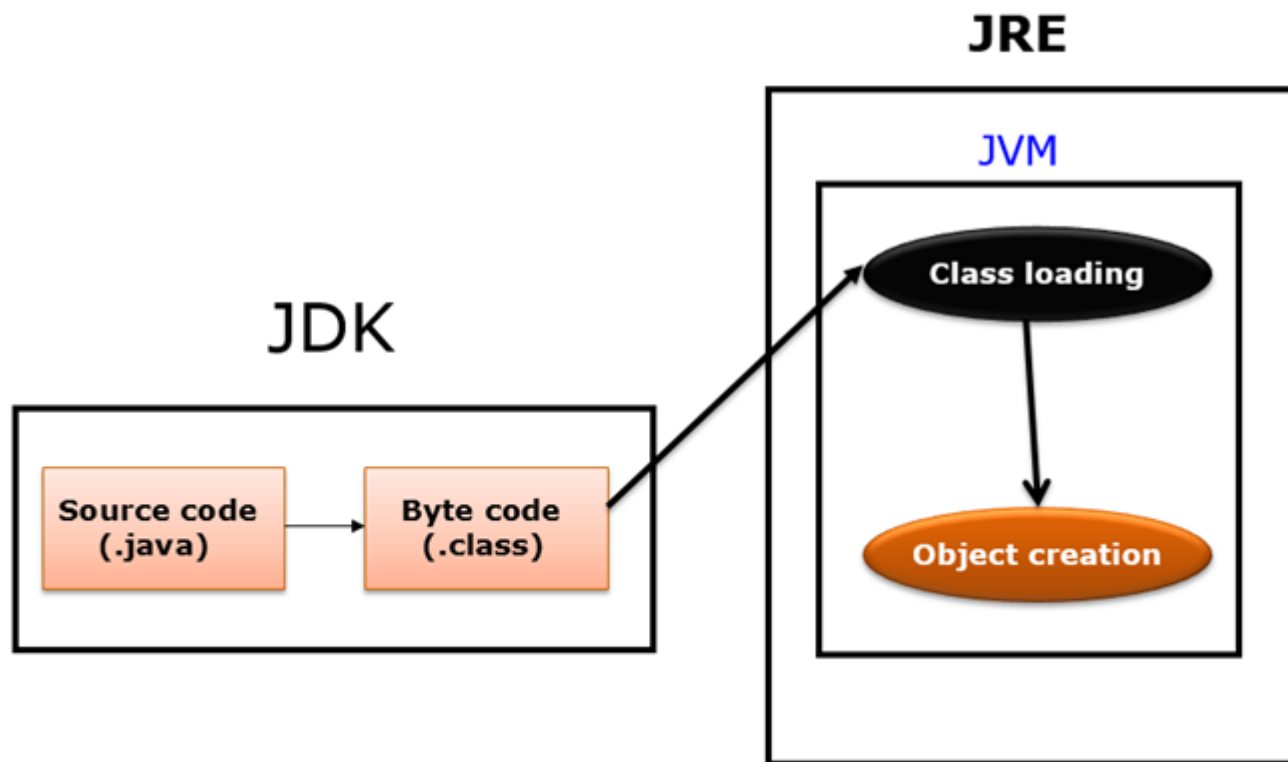
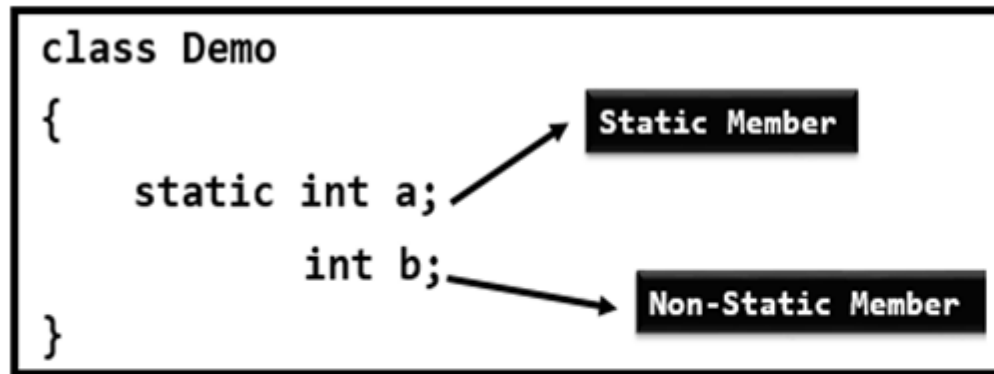
```
}
```



Static Member

Non-Static Member





```
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();
```

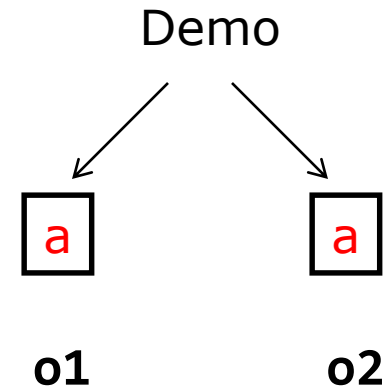
```
o1.a=1000;
```

```
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;
```

Class Variable

```
    static void sum()
```

```
    {
```

```
        cout<<a;
```

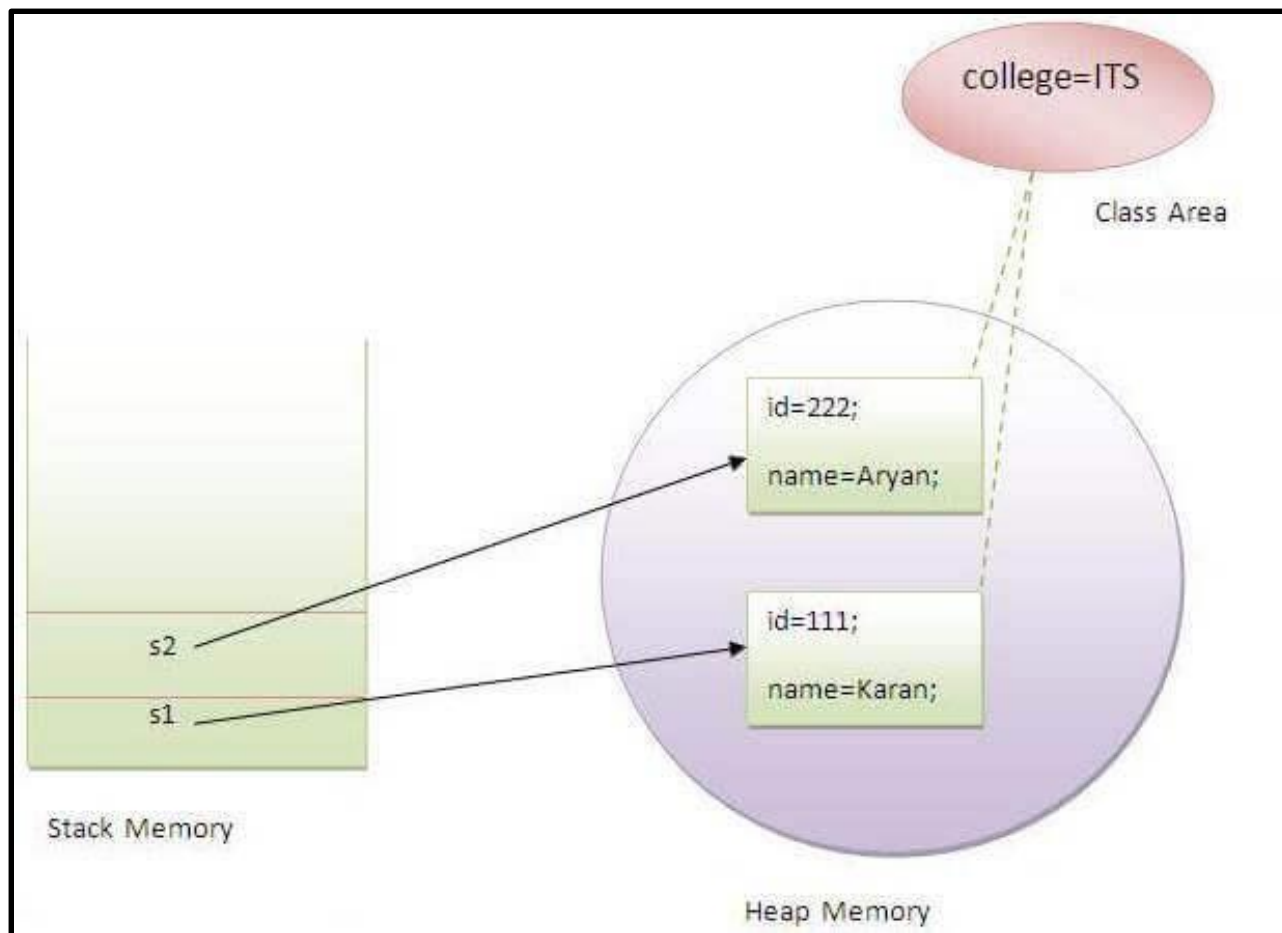
```
    }
```

```
}
```

Class Method

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

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



```
class Demo
```

```
{
```

```
    void sum()
```

```
    {
```

```
        int a;
```

```
        cout<<a;
```

```
    }
```

```
}
```




Local Variable

The diagram illustrates the scope of a local variable. An arrow points from the variable 'a' in the code to a rectangular box labeled 'Local Variable', indicating that the variable is local to the function 'sum()'.

Packages

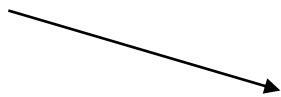
```
package yahoo;
```

```
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);
}
```

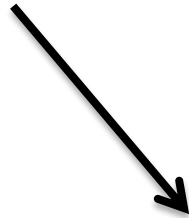
```
class System
```

```
{
```

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

```
}
```

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

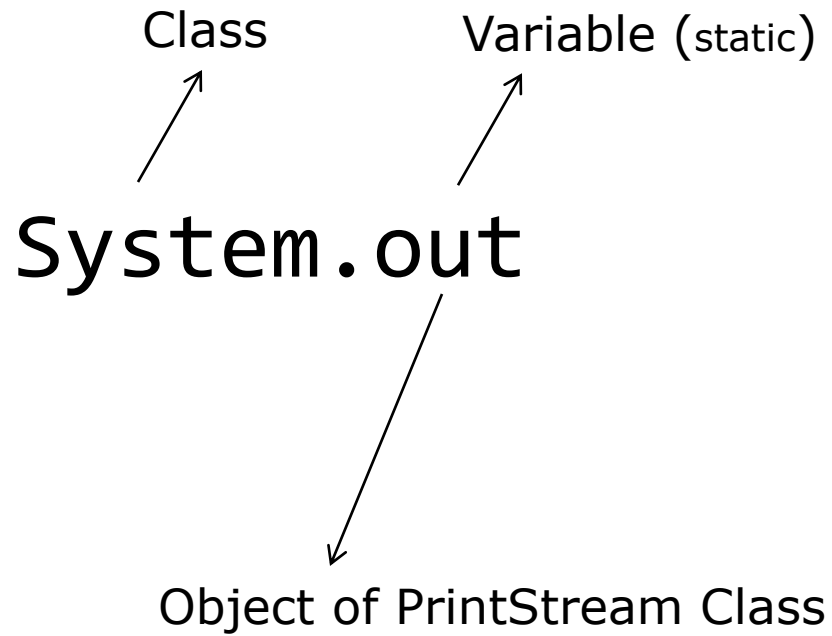


Object of PrintStream class

Class Variable (static)

↗ ↗

System.out



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

Class
1 ↗

Variable (static)
2 ↗

System.out.println("Welcome");

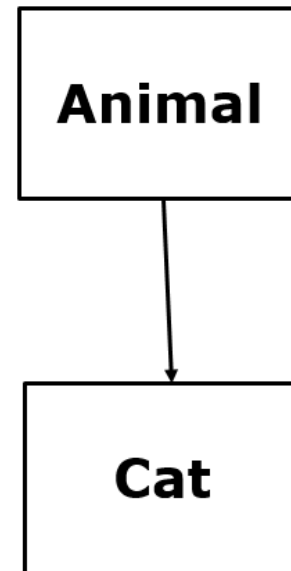
3 ↘
Object of PrintStream Class

4 ↘
Method

```
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)

byte → short → int → long → float → double

widening

- Narrowing Casting(Explicitly done)

double → float → long → int → short → byte

Narrowing

```
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  
{
```

A

A ob1;

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

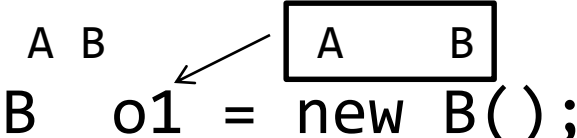
A B

B ob2;

```
}
```

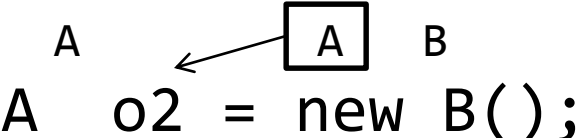
```
class A
{
}
class B extends A
{
}
```

^{A B}
B o1 = new B();



The diagram illustrates the memory layout for the first line of code. A rectangular box is divided into two sections, the left one labeled 'A' and the right one labeled 'B'. An arrow originates from the 'B' section of this box and points to the variable 'o1' in the code 'B o1 = new B();'. Above the box, the labels 'A' and 'B' are positioned. To the left of the variable 'o1' is the label 'B'.

^A
A o2 = new B();



The diagram illustrates the memory layout for the second line of code. A rectangular box contains the label 'A'. An arrow originates from this box and points to the variable 'o2' in the code 'A o2 = new B();'. Above the box is the label 'A'. To the left of the variable 'o2' is the label 'A'.

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

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

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

Up-casting

A s1 = new B();

B s2 = (B)s1;

Down-casting

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();

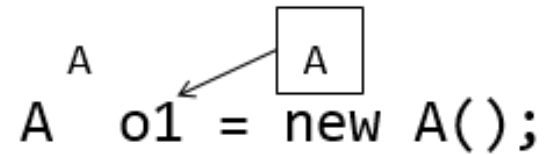


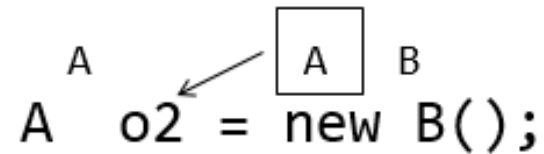
A r2 = new B();

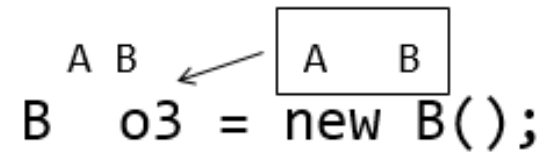


```
class A
{
```

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


A^A o1 = new A();


A^A o2 = new B();


B^B o3 = new B();

```
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
```

```
{  
}
```

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

```
class B
```

```
{  
}
```

```
B o2 = new B();
```

```
class C
```

```
{  
}
```

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

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

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

```
Object o2 = new B();
```

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

Loosely Coupled

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

Object o1 = ^{o A}new A();

Object o2 = ^{o B}new B();

Object o3 = ^{o C}new C();

```
class A
```

```
{  
}
```

```
class B
```

```
{  
}
```

```
class C
```

```
{  
}
```

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

```
A s2 = (A)s1;
```

✓ 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()
```

Method Declaration

```
{
```

```
    int a=200;
```

```
    int b=400;
```

```
    S.o.p(a+b);
```

```
}
```

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();  
}
```

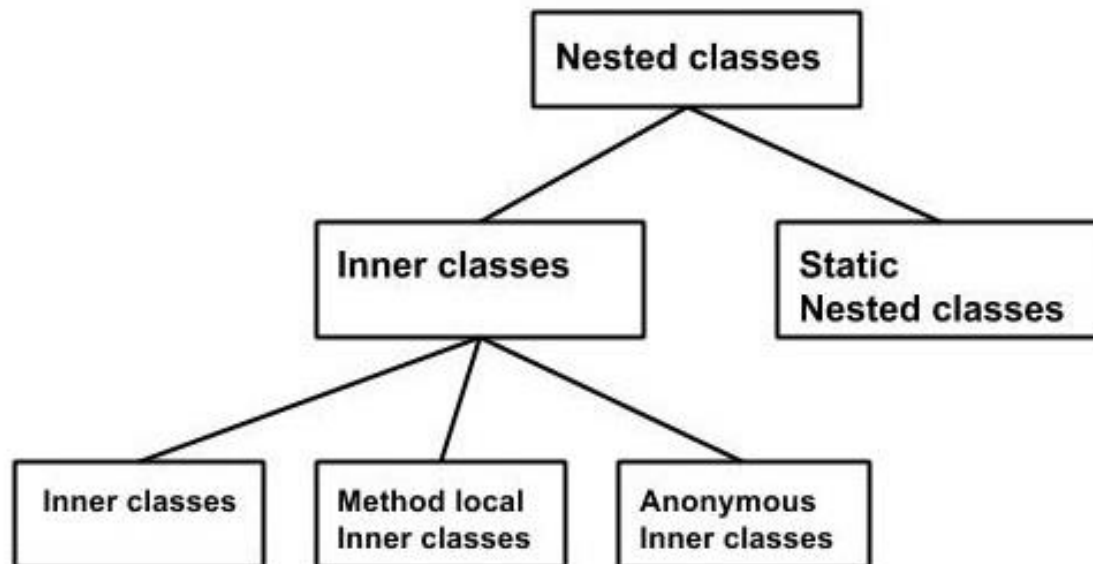
Methods Overriding

```
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{  
    }  
}
```



```
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.