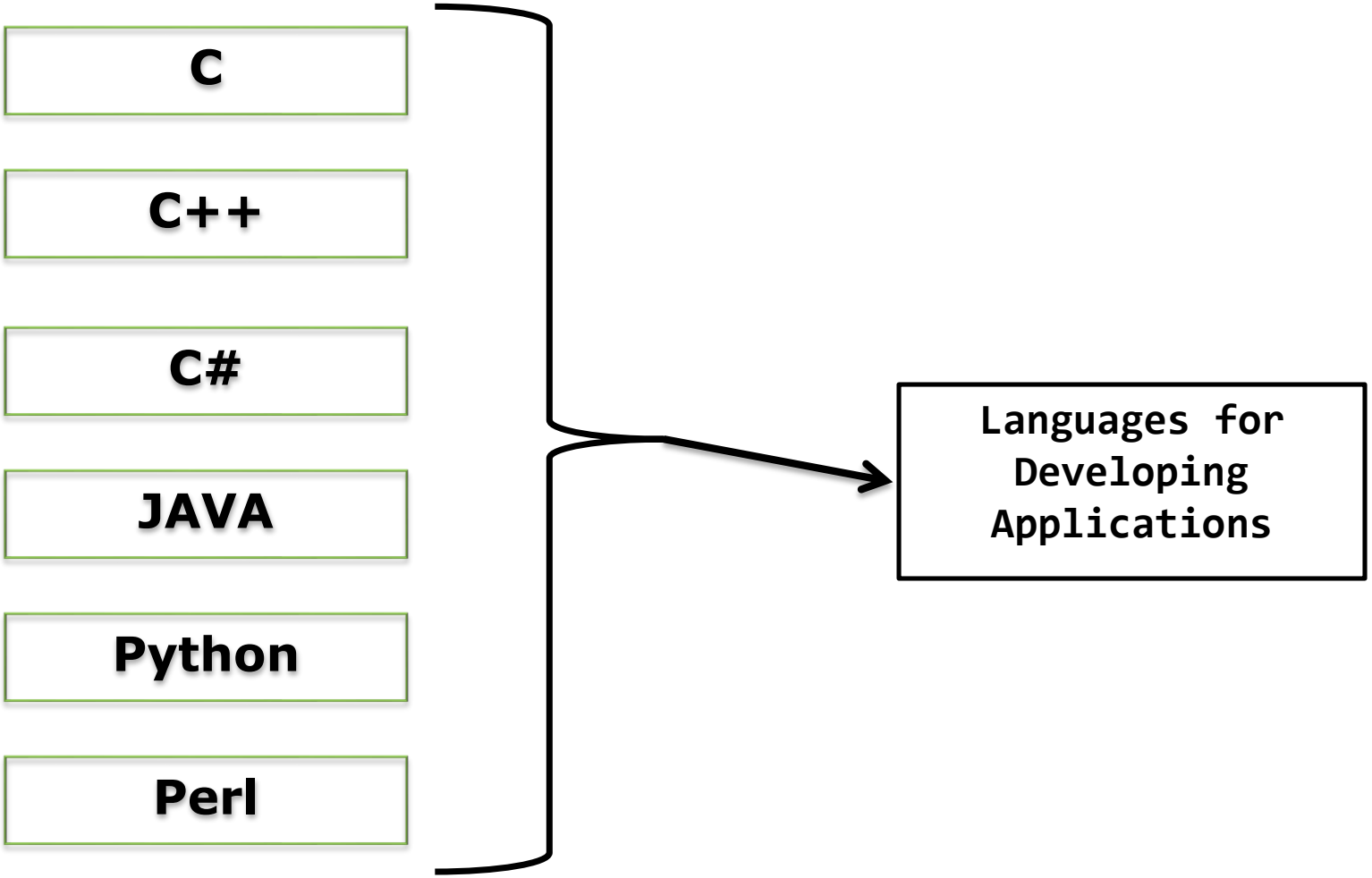
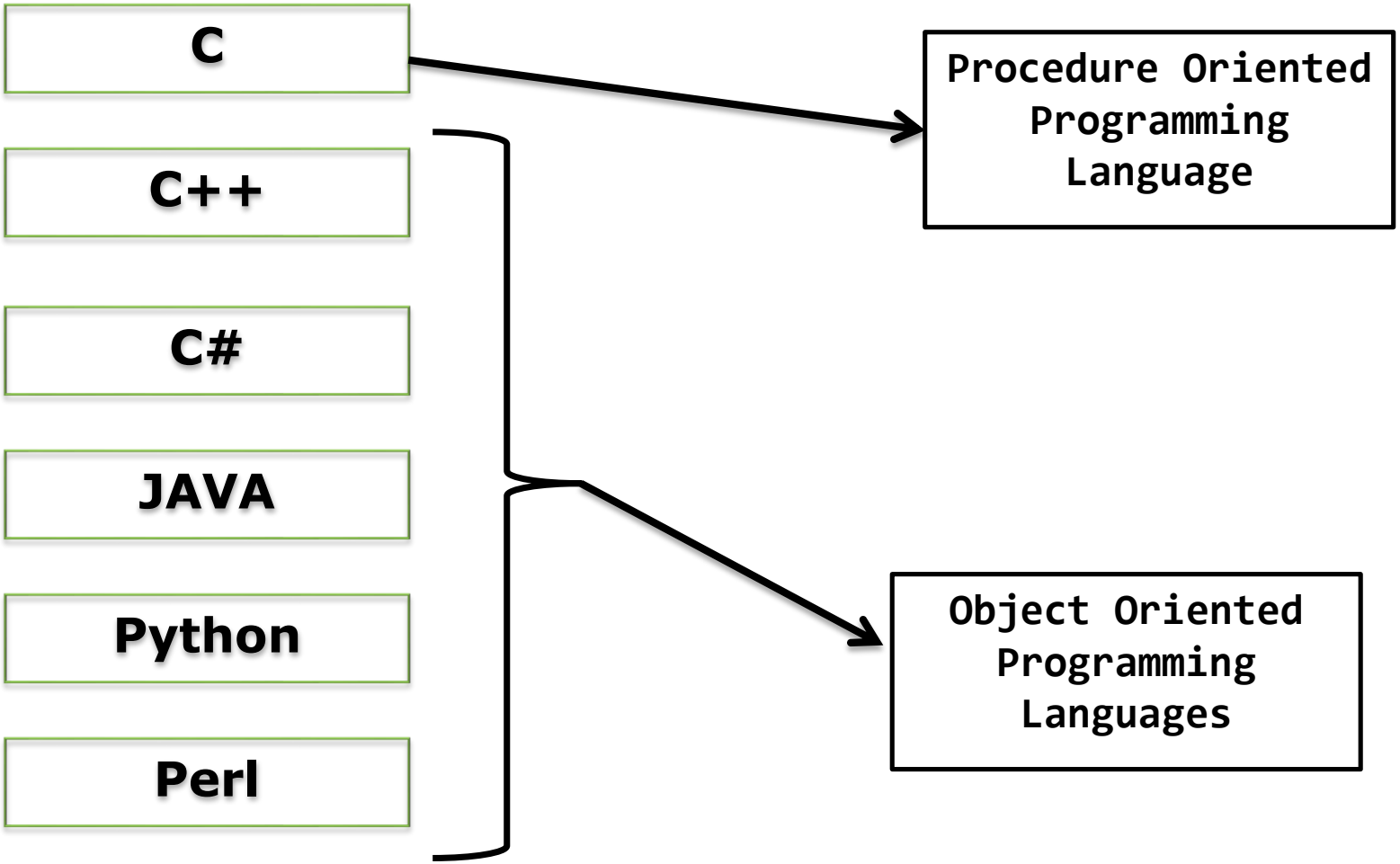
Java Programming Language

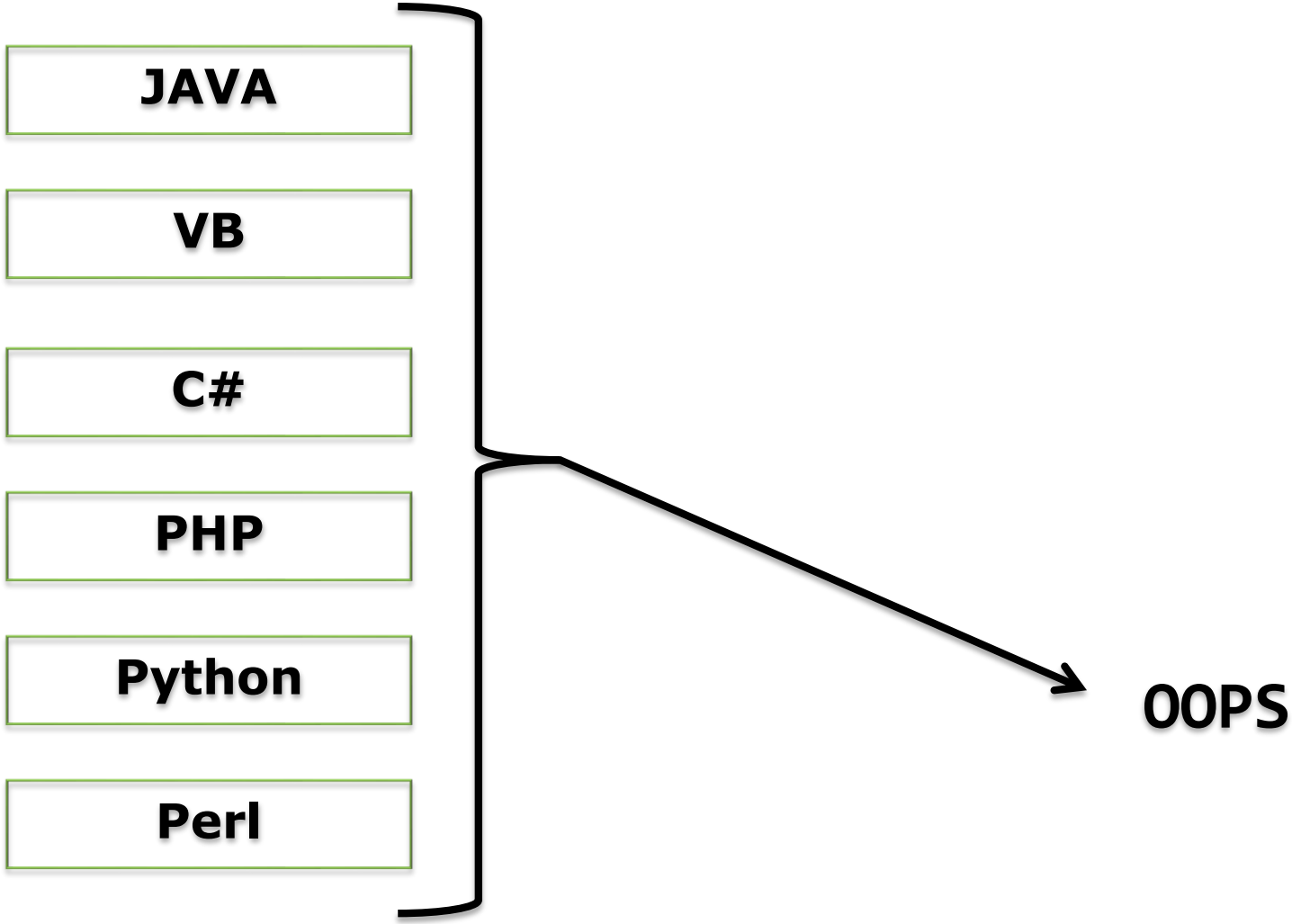
Programming Languages



Programming Languages

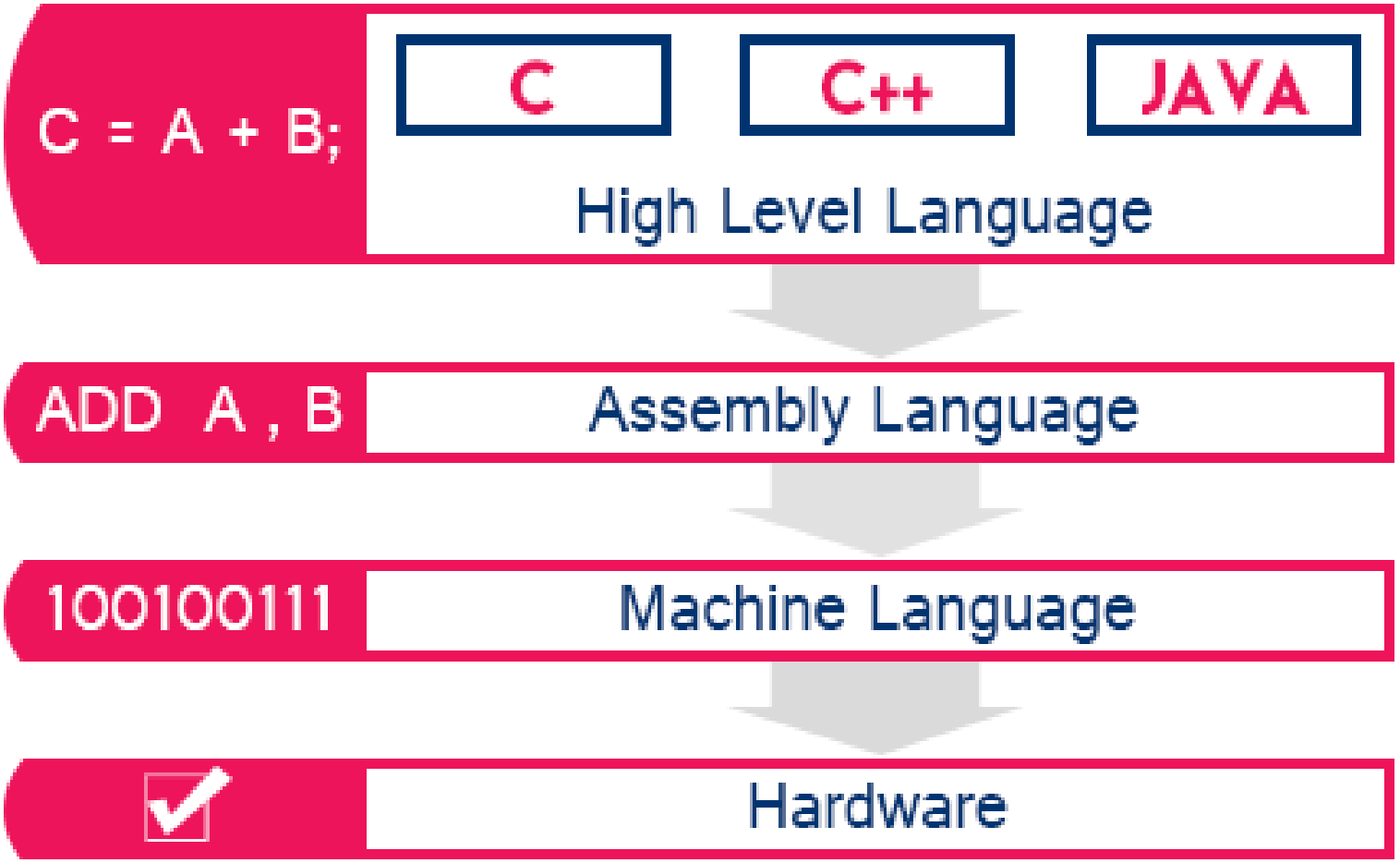


Programming Languages

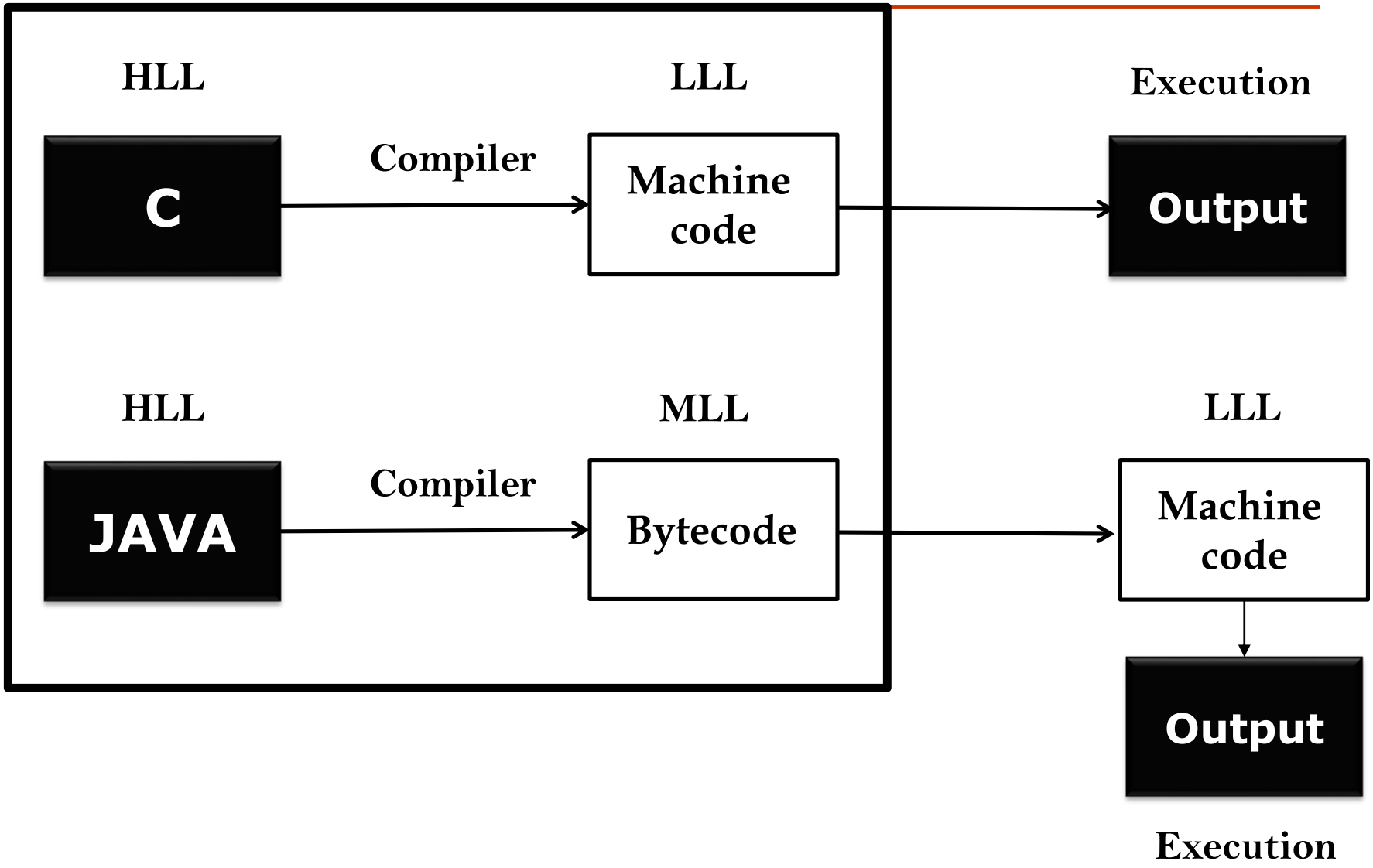


Java Features

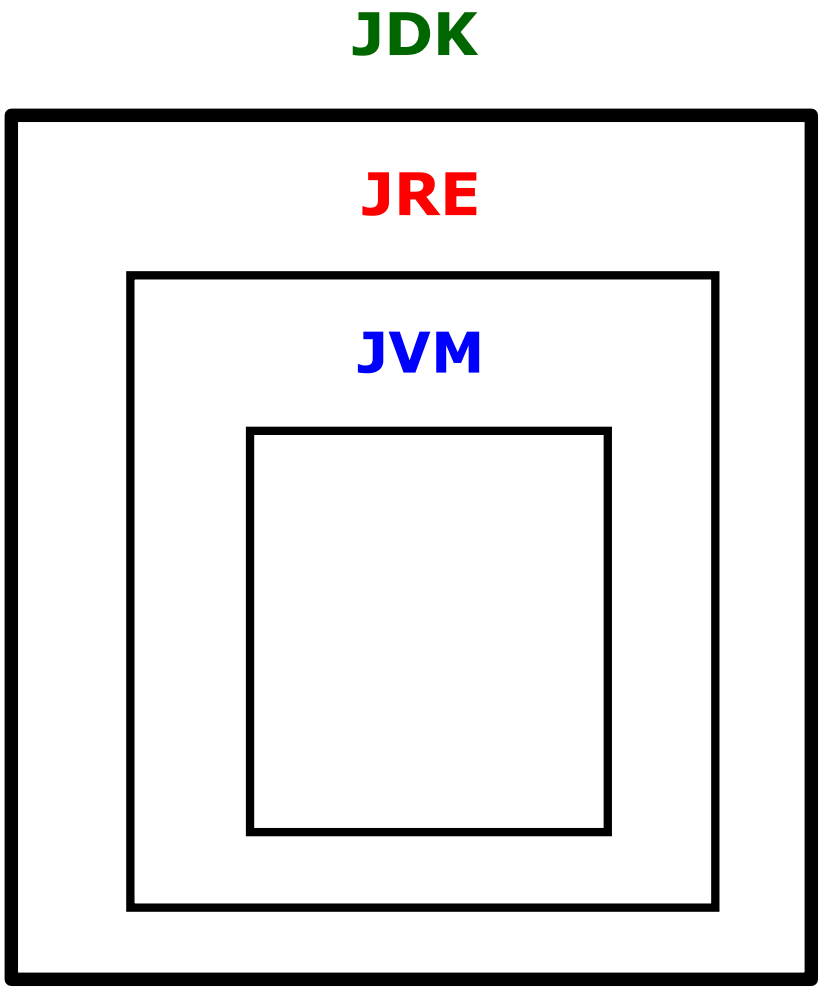
- Platform Independence
- Object Oriented Programming Language


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

Programming Execution



JDK Architecture



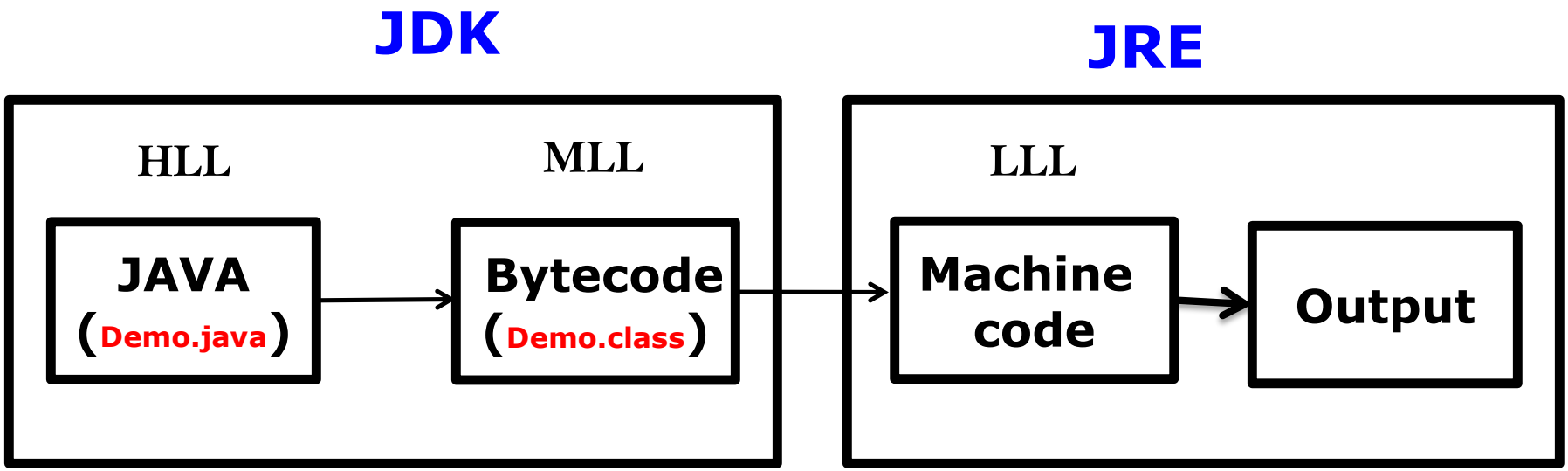
JDK Architecture

JDK → Java Development Kit

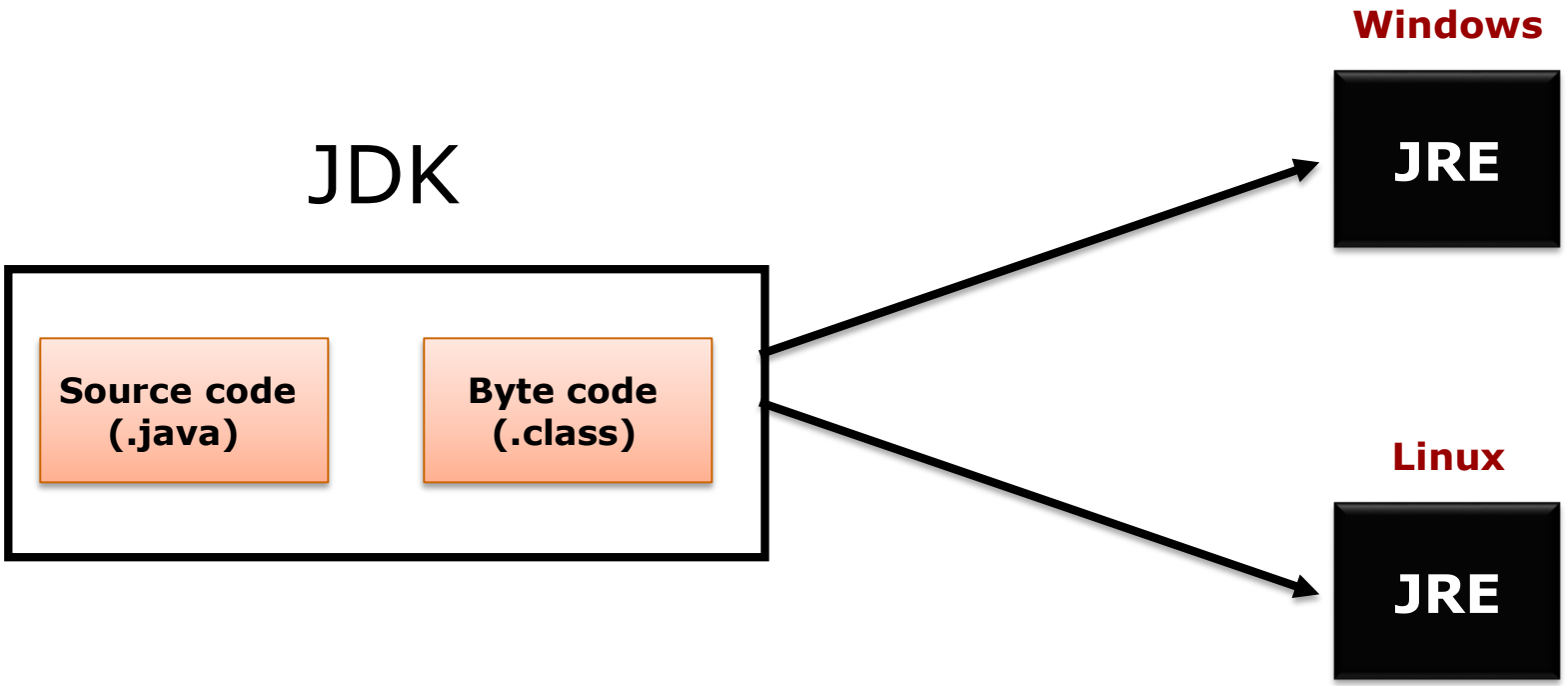
JRE → Java Runtime Environment

JVM → Java Virtual Machine

Programming Execution



Platform Independence



Programming Basics

Printing Statement

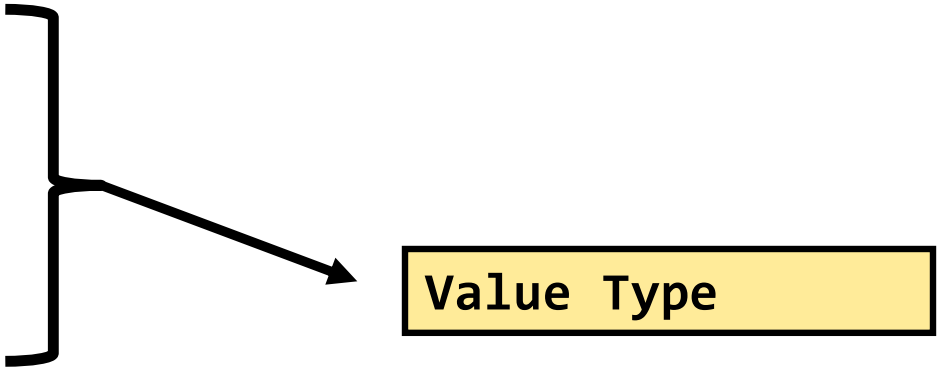
```
System.out.println("Welcome");
```

Data Types

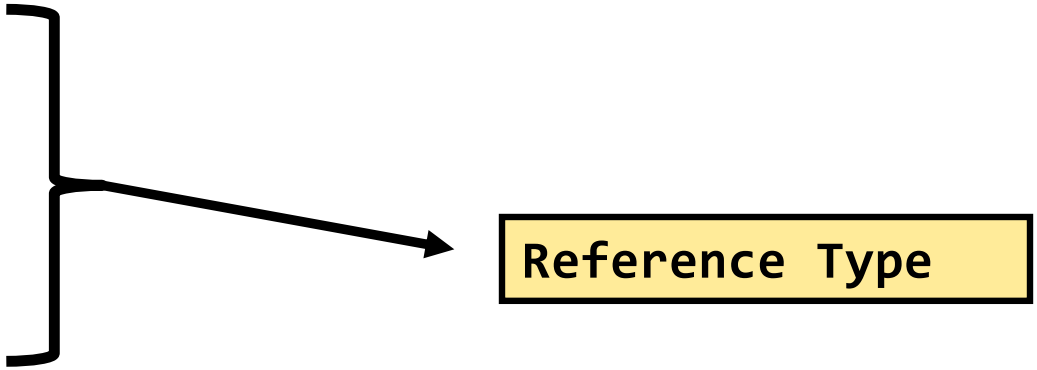
- ✓ boolean
- ✓ char
- ✓ byte
- ✓ short
- ✓ int
- ✓ long
- ✓ float
- ✓ double

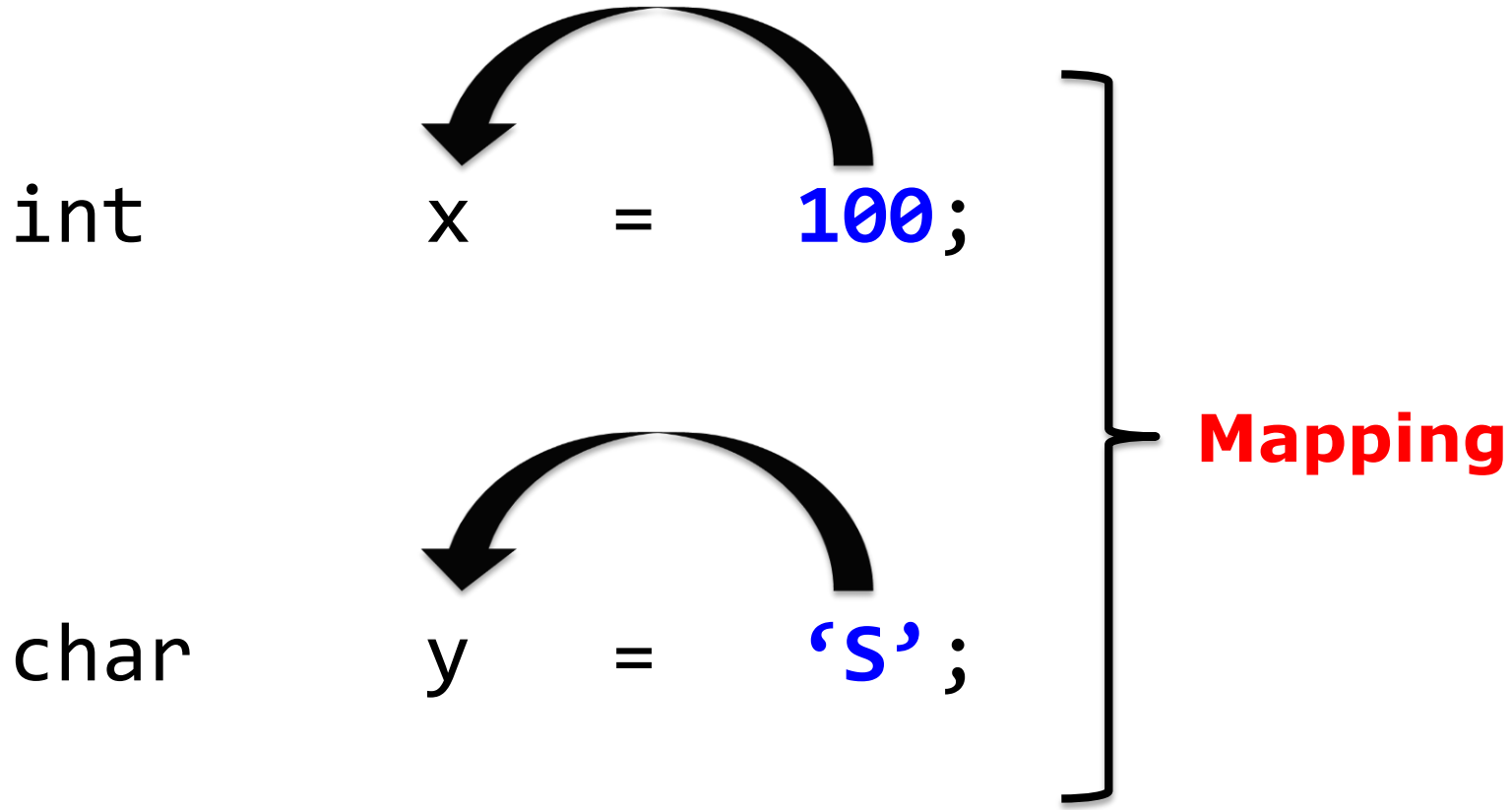
Pointers

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



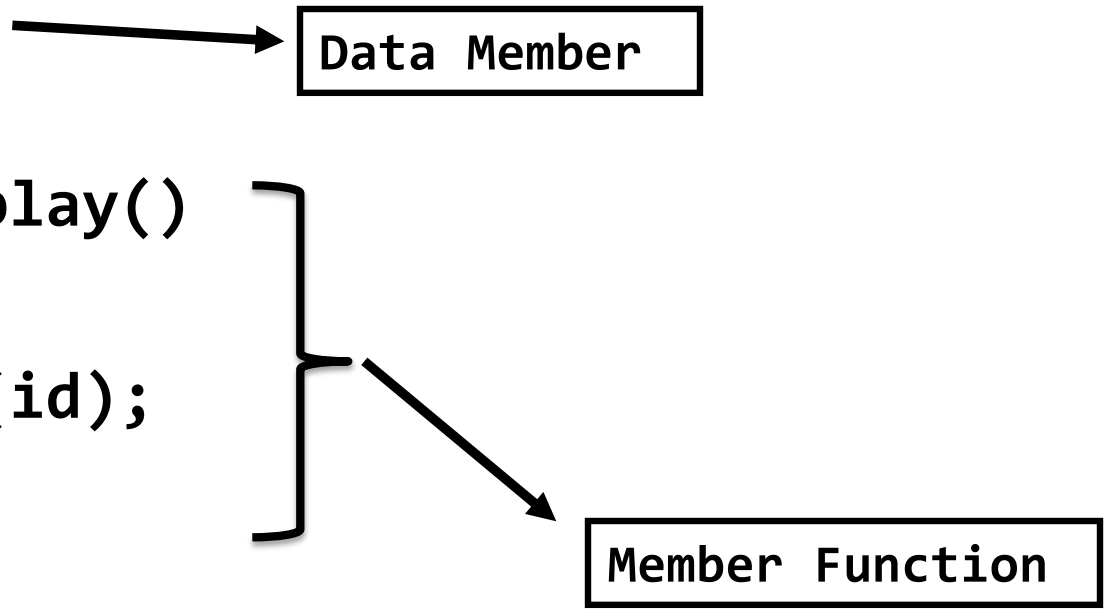

```
int     *x =   &a;
char    *y =   &b;
float   *z =   &c;
```



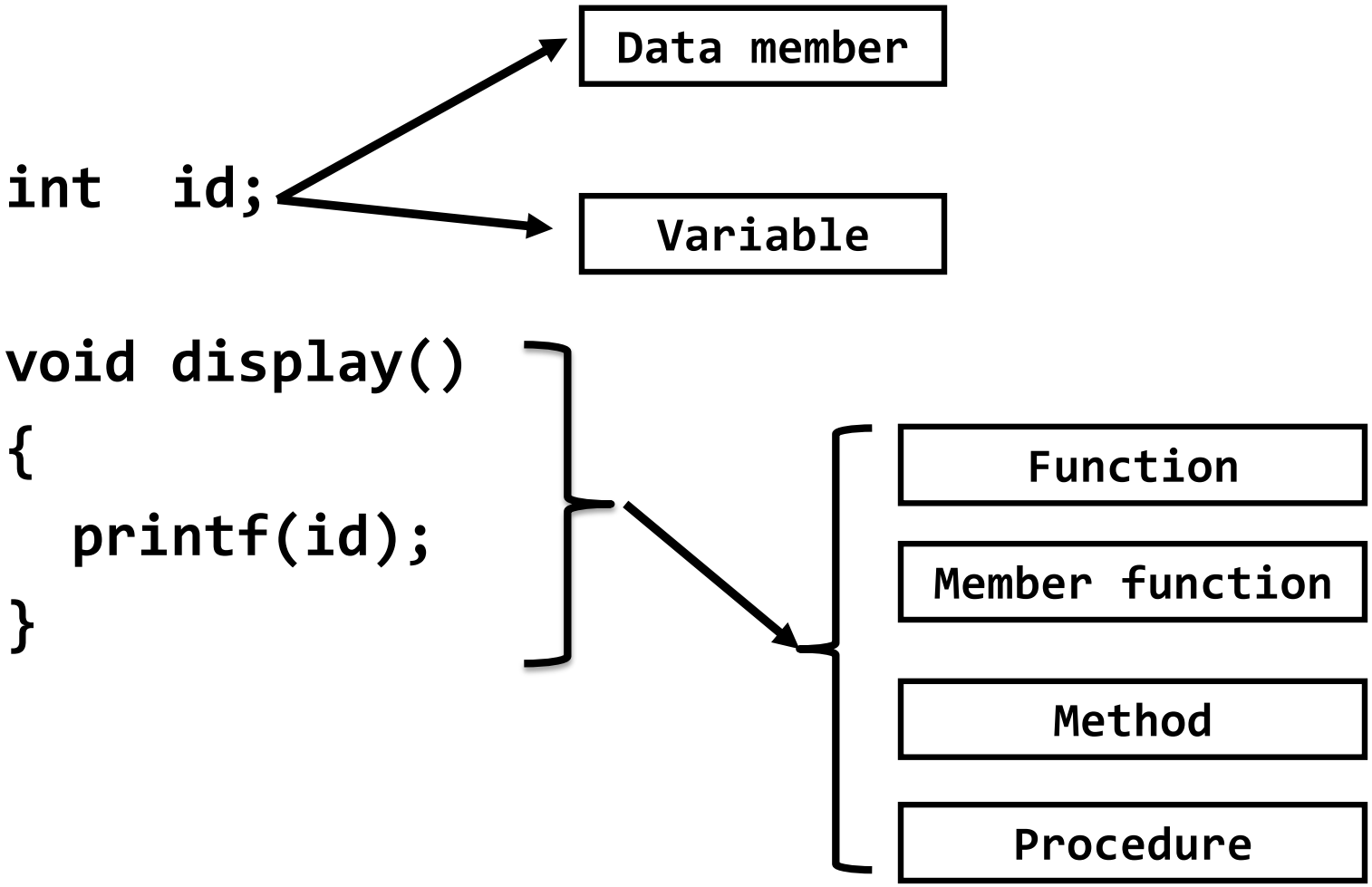

int x = 100;
char y = 'S';
Mapping

C Language

```
int  id;

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



C Language


int id;
Data member
Variable
void display()
{
printf(id);
}
Function
Member function
Method
Procedure

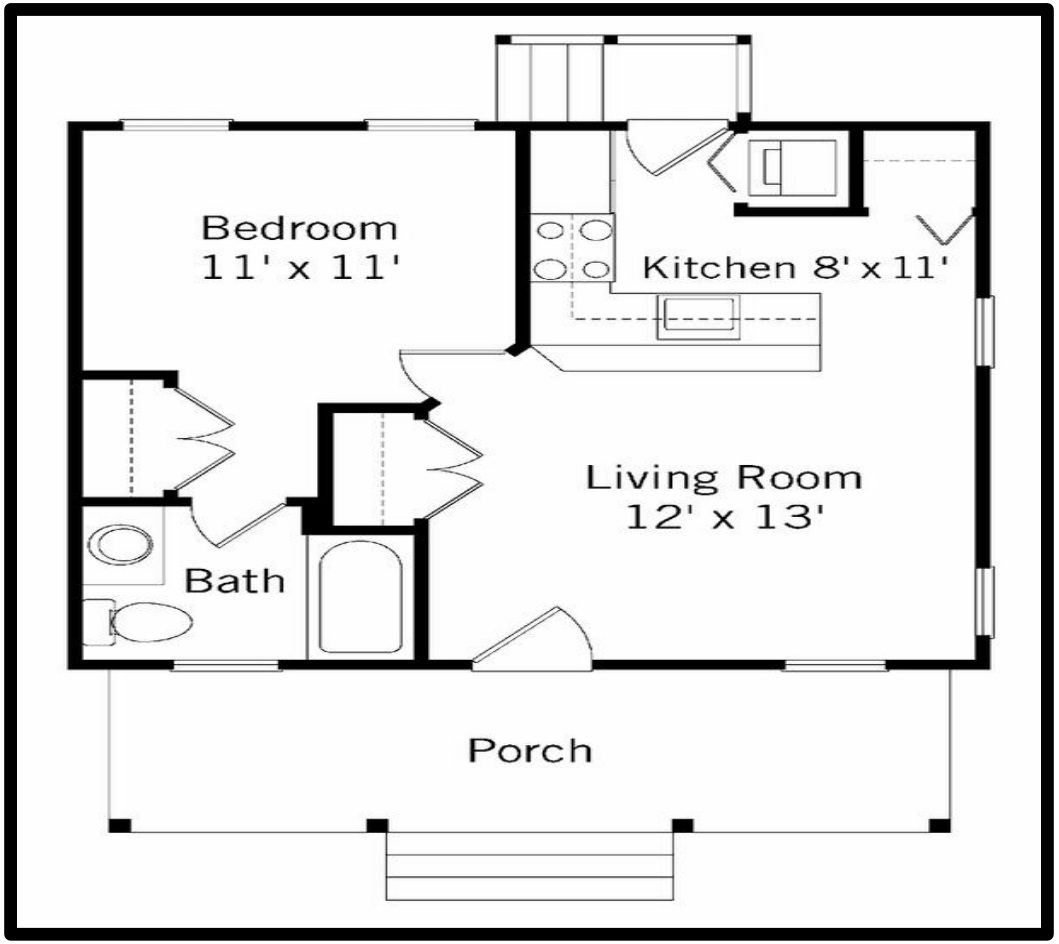
C Language

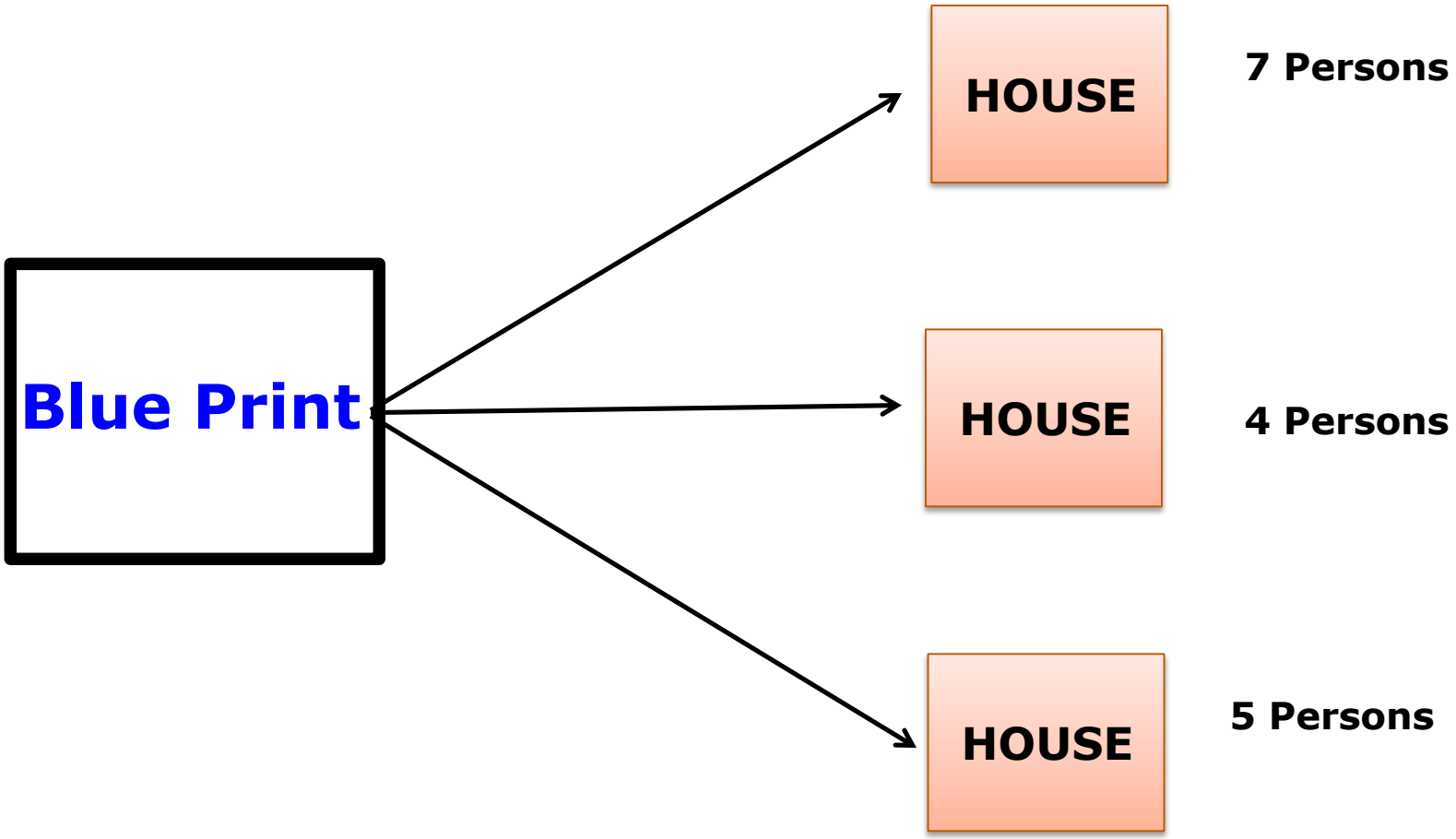
```
void calculator()
{
  // 100 Lines of code
}
void scientific_calculator()
{
  // 200 Lines of code
}
```

Object Oriented Programming Language

- Class
- Object

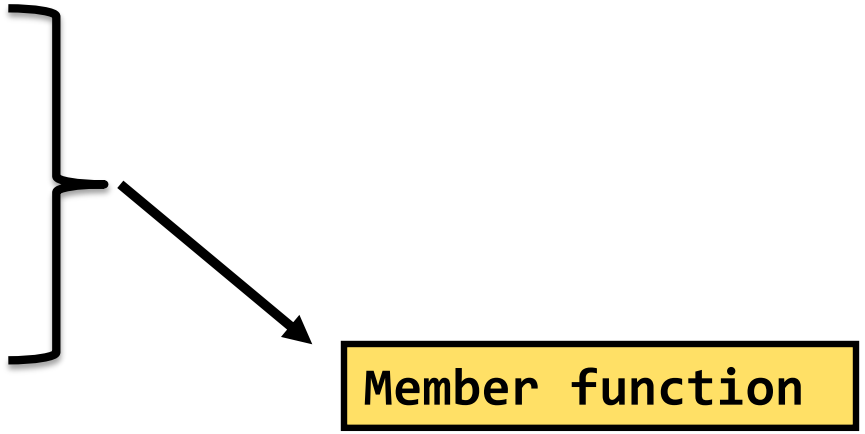
House Blue Print




Blue Print
HOUSE
HOUSE
HOUSE
7 Persons
4 Persons
5 Persons

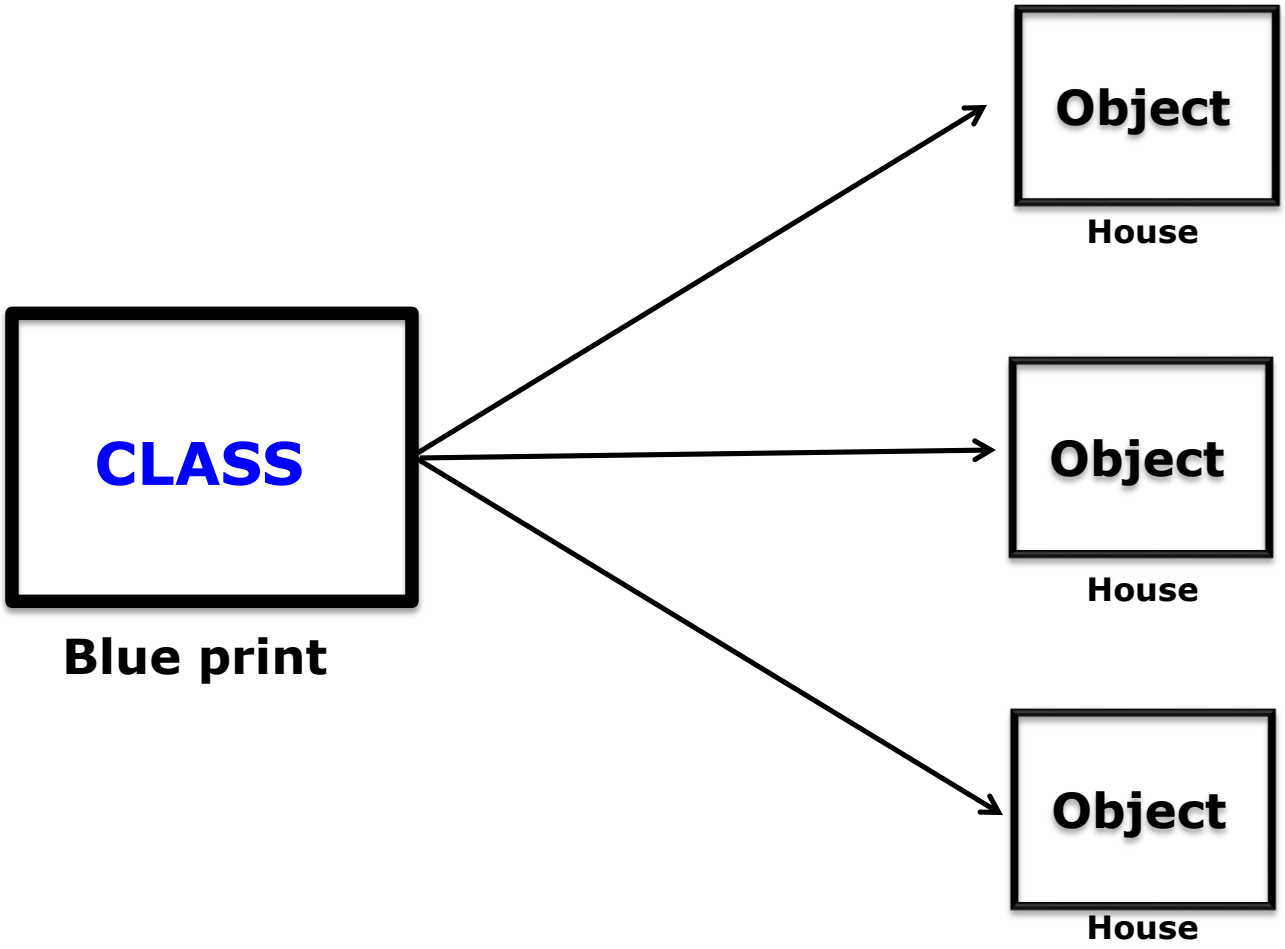
C Language

```
int  id;

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

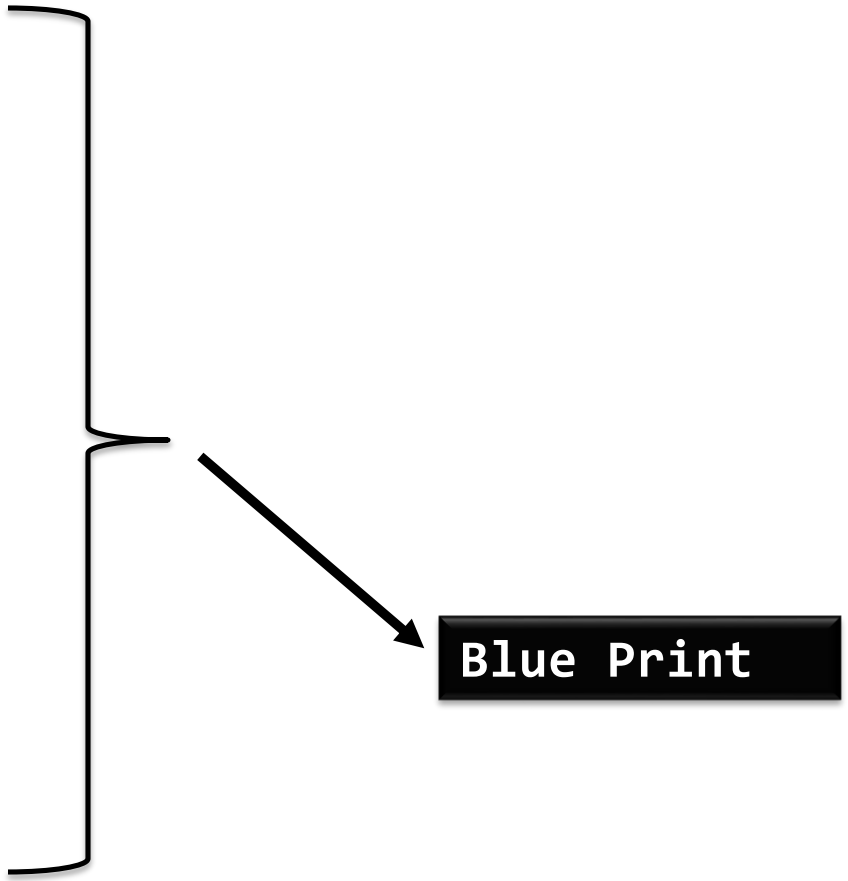


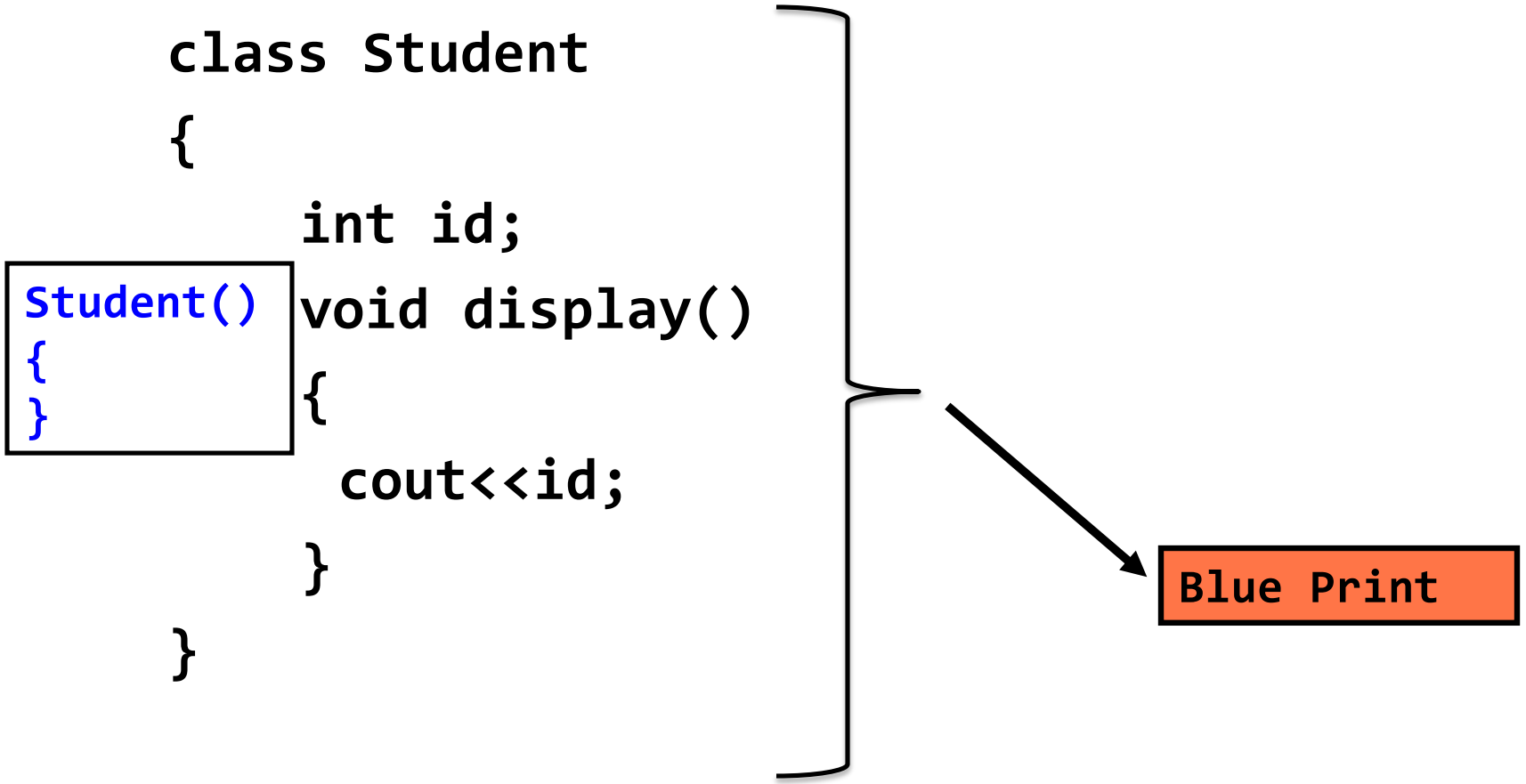
```
class Student
{
    int id;
    void display()
    {
     cout<<id;
    }
}
```

Class and Objects

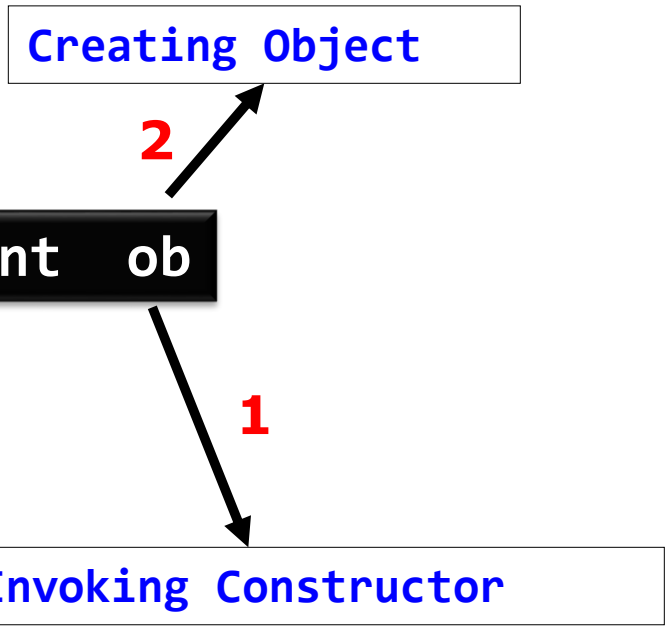


```
class Student
{
    int id;
    void display()
    {
     cout<<id;
    }
}
```

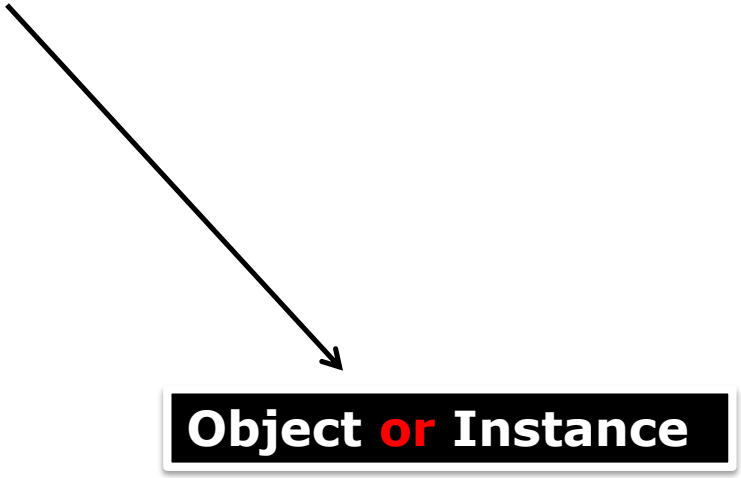



class Student
{
int id;
Student()
{
}
void display()
{
cout<<id;
}
}
Blue Print

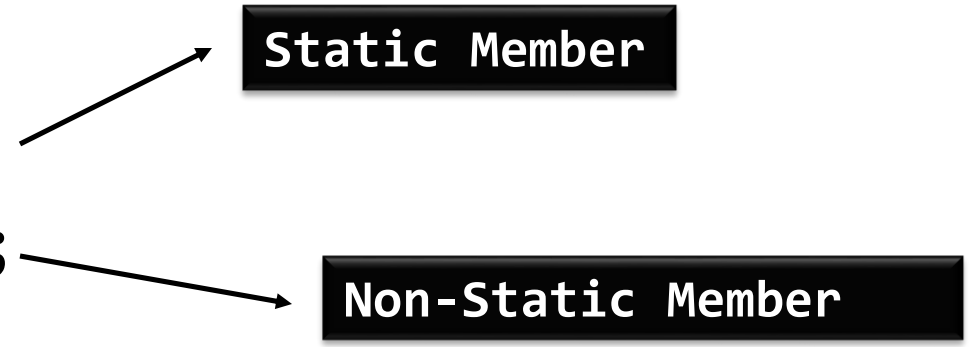
```
class Student
{
 int id;
   Student()
   {

   }
}
```



```
Student  ob;
```



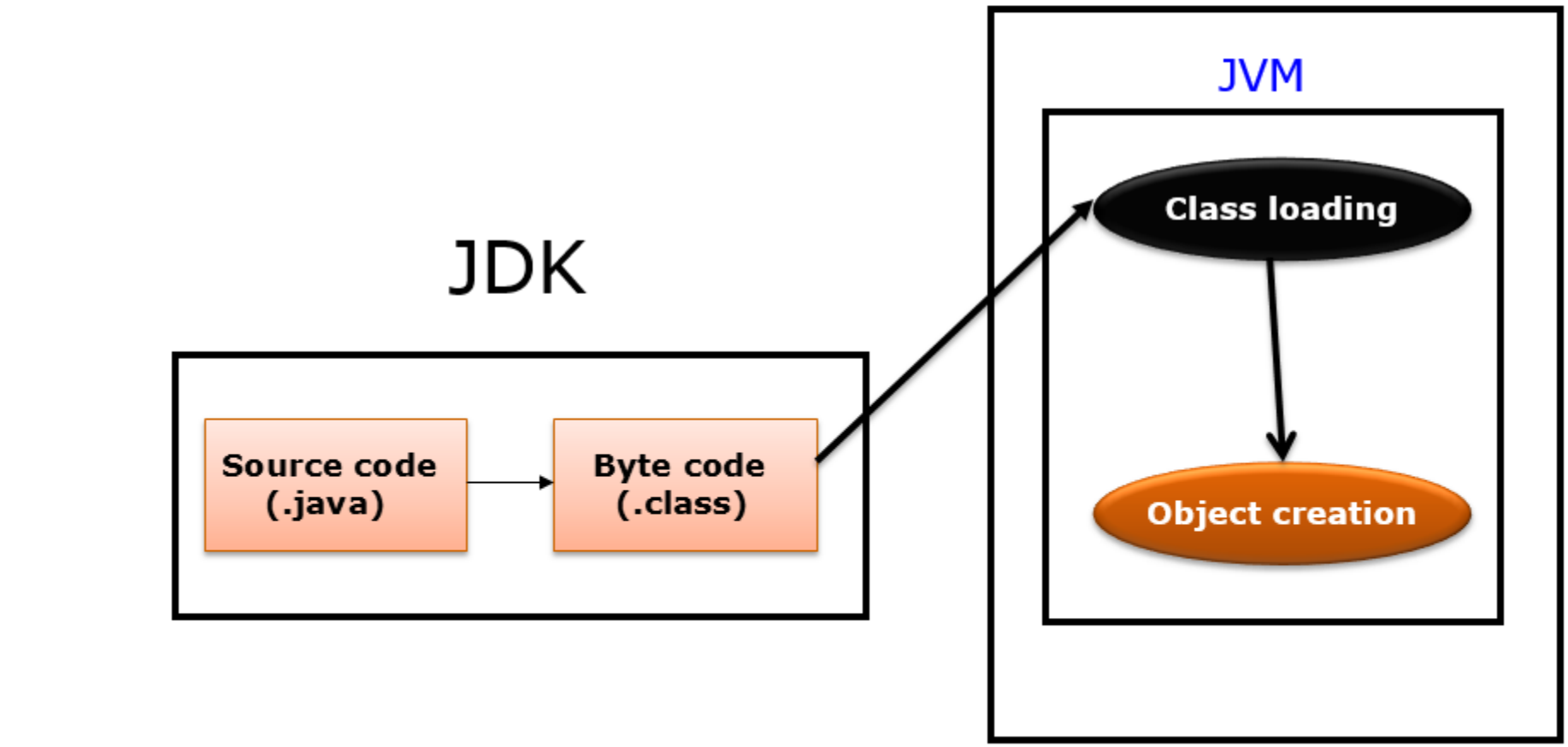
```
class Demo
{
    static int a;
           int b;
}
```



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

Static Member

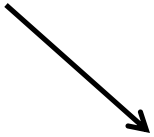
Non-Static Member



```
int        x;
```

Pre-defined data type

```
Student   ob;
```



User-defined data type

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

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

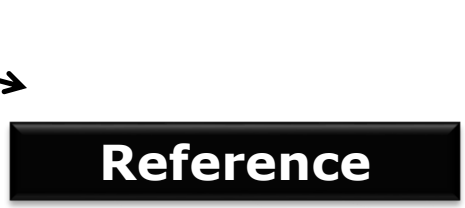
```
class String
{


}
```

```
Student  s1;
```

Object creation of Student class

Pointers

```
int      *x;
```



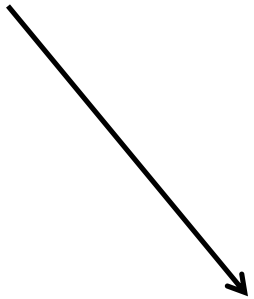

```
Student  *ob;
```

Pointers

```
Student  ob;
```

Object creation

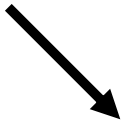
```
new Student()
```



Object creation

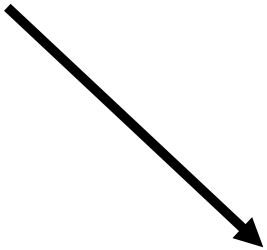
Object Creation in C++

```
Student   ob;
```



Value Type

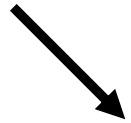
```
Student  *ptr = new Student();
```



Reference Type

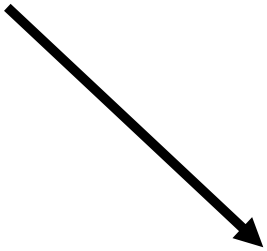
Object Creation in C++

```
Student   ob;
```



Value Type

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

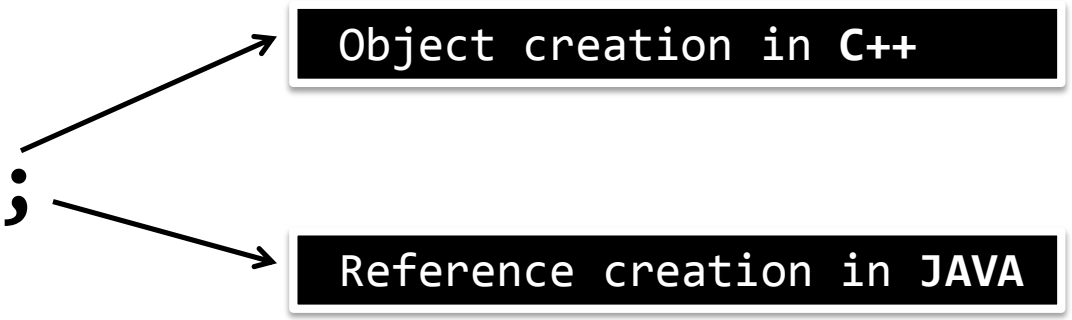


Reference Type

JAVA

```
Student   *ob1;
```


```
Student    ob2;
```



Reference

```
Student   *ob1;
```

Reference in C++

```
Student    ob2;
```

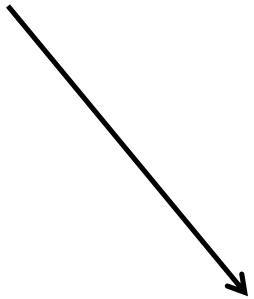
Reference in JAVA

C++

```
Student   *ob;
```

Reference

```
new Student()
```



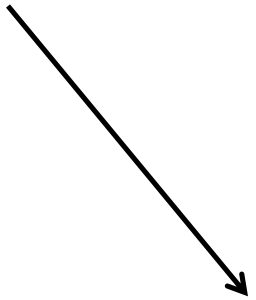
Object creation

Java

```
Student   ob;
```

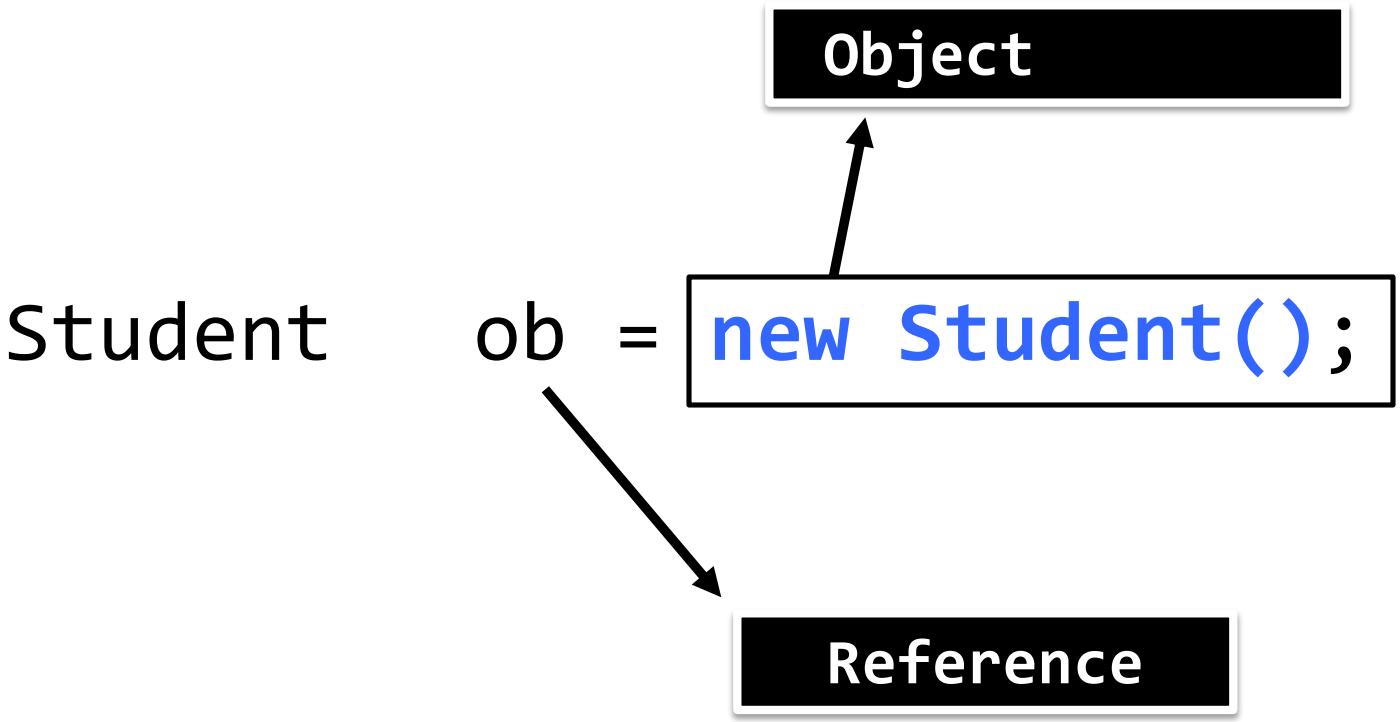
Reference

```
new Student()
```



Object creation

Object Creation in Java



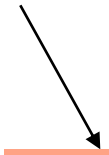
Object Creation

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

Object

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

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



Instance (or) Object

```
Student   ob =  new Student();
          ob.id=200;
          ob.display();
```

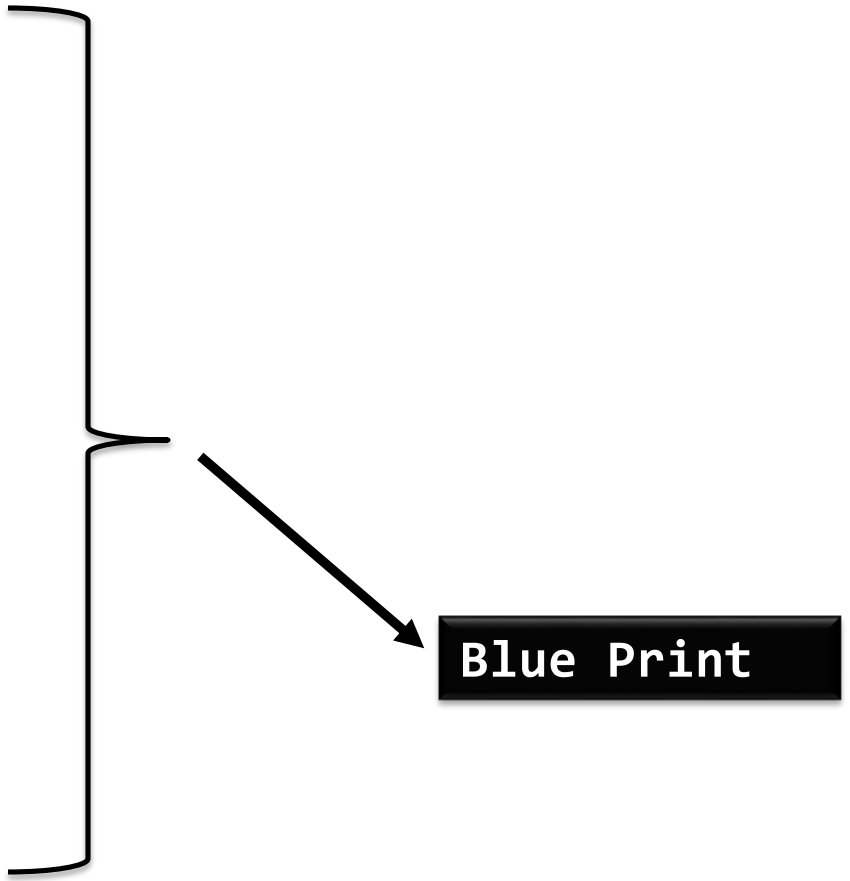
Data Members

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

```
class Address
{
}
```

Class

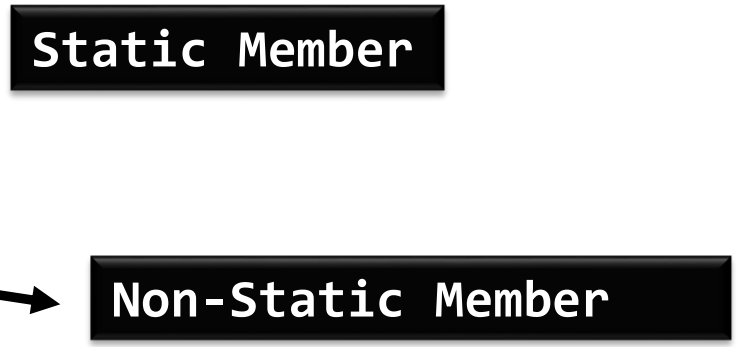
```
class Student
{
     int id;
     void display()
     {
       cout<<id;
     }
}
```



Types of Variables and Methods

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

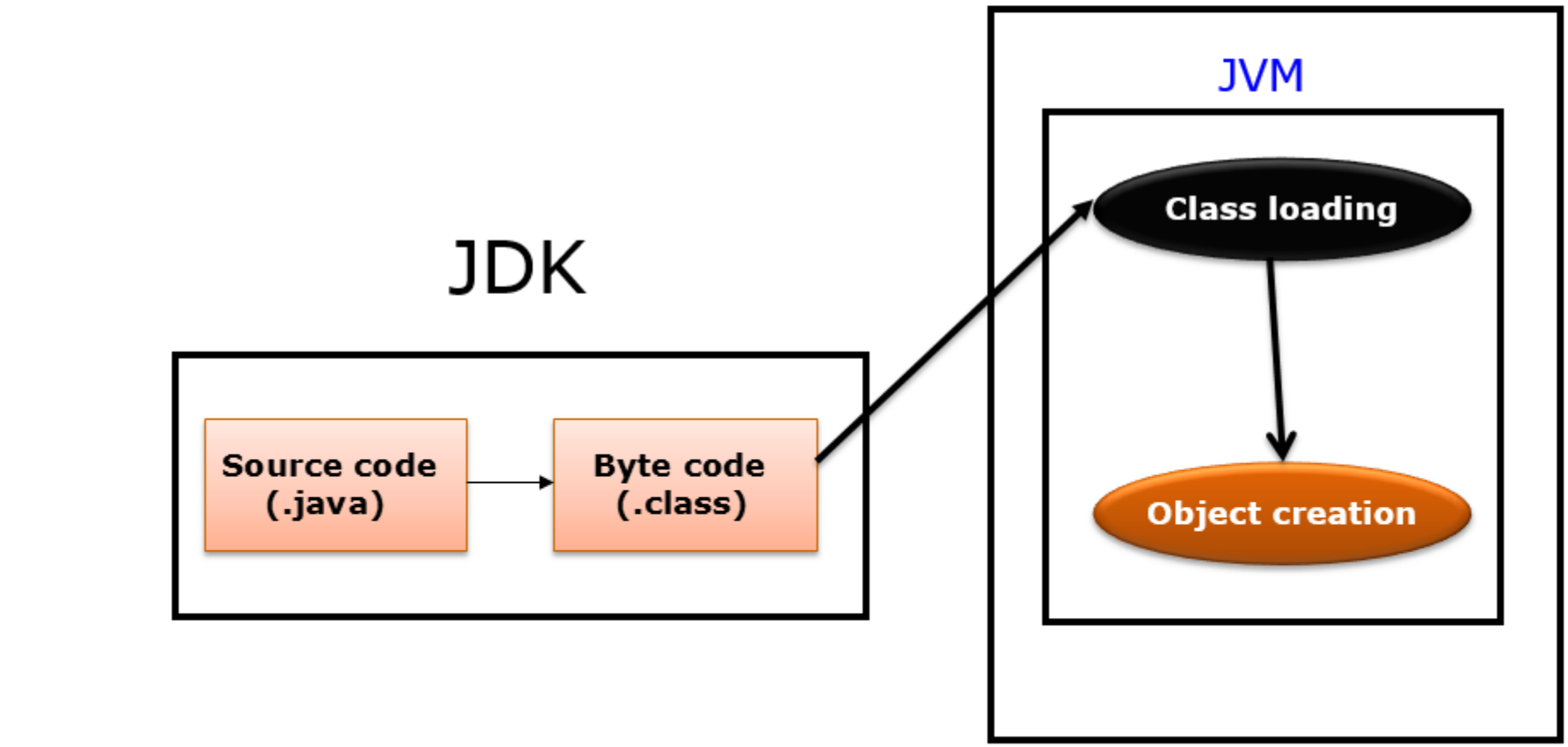
```
class Demo
{
    static int a;
           int b;
}
```



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

Static Member

Non-Static Member



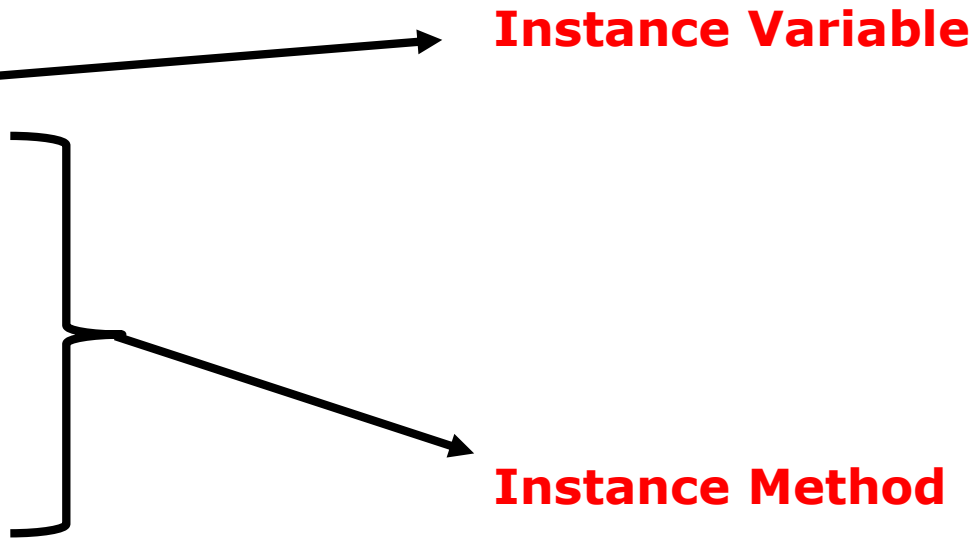
```
class Demo
{
    static int a;
           int b;
}
```

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

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

Instance variable and method

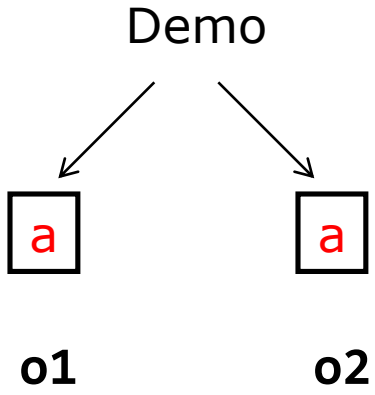
```
class Demo
{
     int a;
     void sum()
     {
       cout<<a;
     }
}
```



Instance variable and method

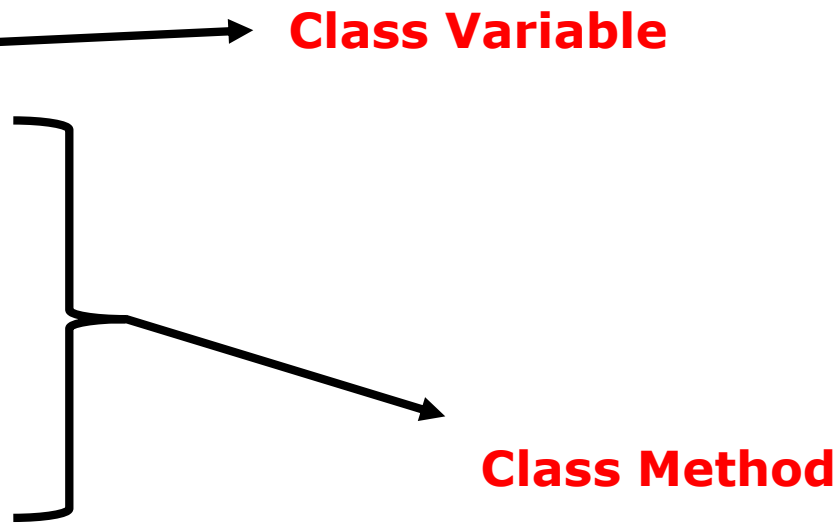
```
class Demo
{
    int a;
    void sum()
    {
     cout<<a;
    }
}
```

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

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



Class variable and method

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



Class variable and method

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

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

Local Variable

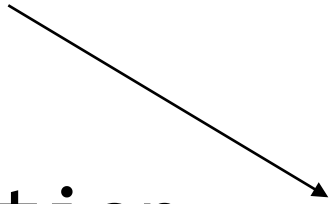
```
class Demo
{
    void sum()
    {
        int a;
        cout<<a;
    }
}
```

Local Variable

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

Packages

```
package yahoo;


class Registration
{
}
```



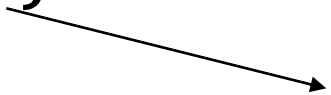
Sub packages

Classes

Interfaces

Java Packages

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



Object

System

Printing Statement

```
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;
}
```

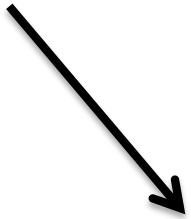
```
System.x = 100;
System.y = 100;
```

PrintStream Class

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

System Class

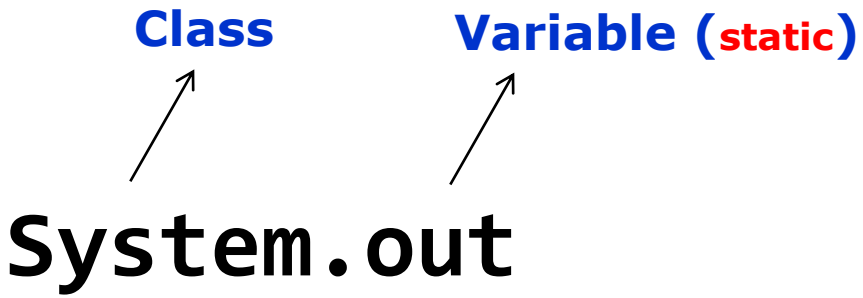
```
PrintStream  out = new PrintStream();
```

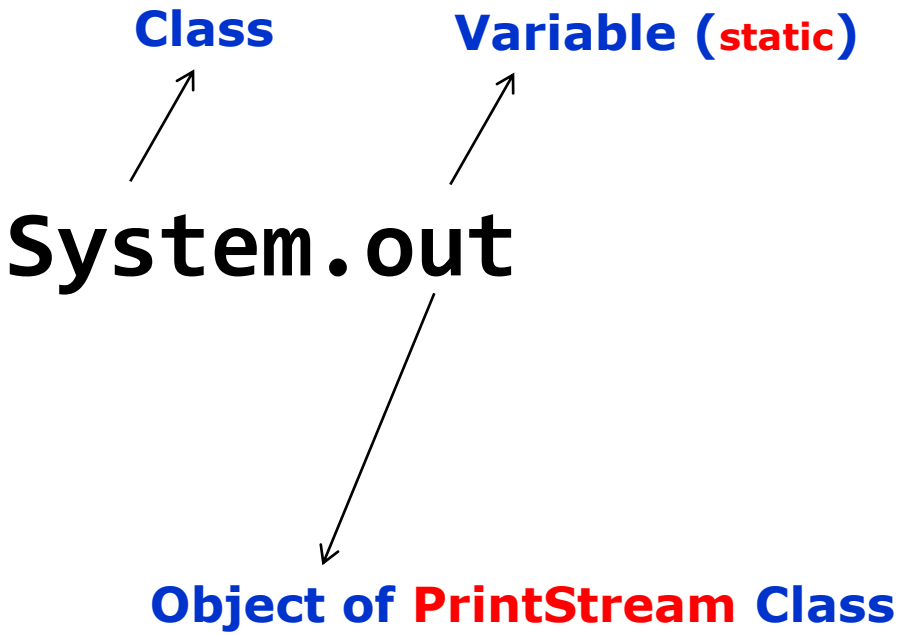


Object of **PrintStream** class

System Class

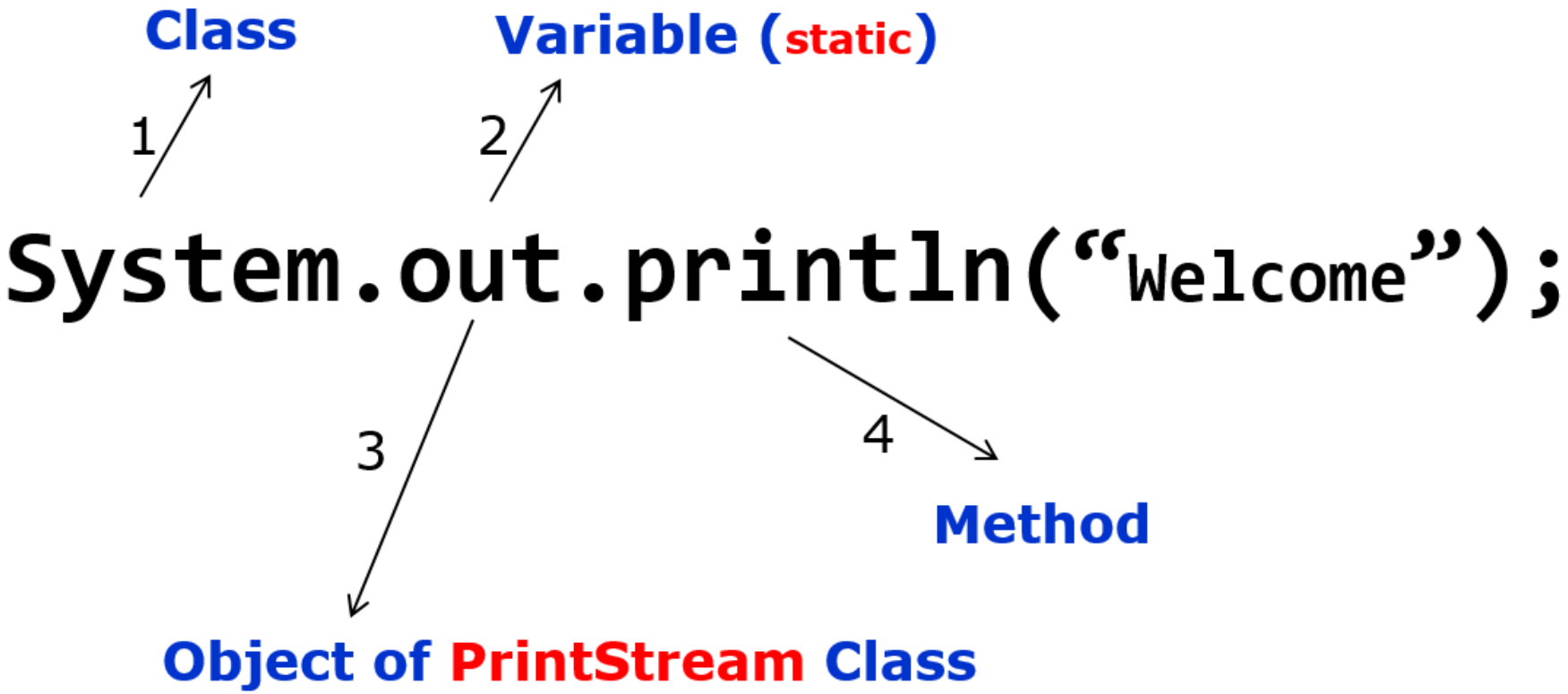
```
class System
{
  static PrintStream  out=new PrintStream();
}
```


Class
Variable (static)
System.out

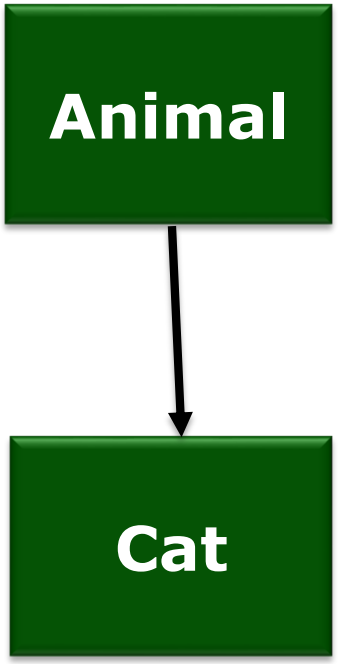

Class
Variable (static)
System.out
Object of PrintStream Class

PrintStream Class

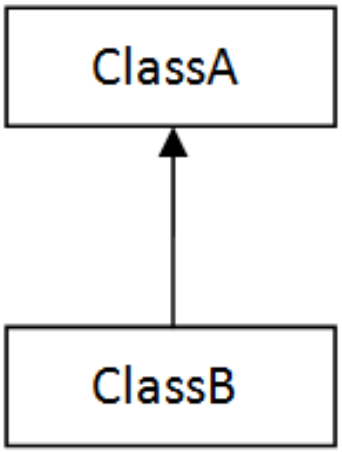
```
class PrintStream
{
        void println(int);
        void println(String);
}
```


Class
Variable (static)
1
2
System.out.println("Welcome");
3
4
Method
Object of PrintStream Class

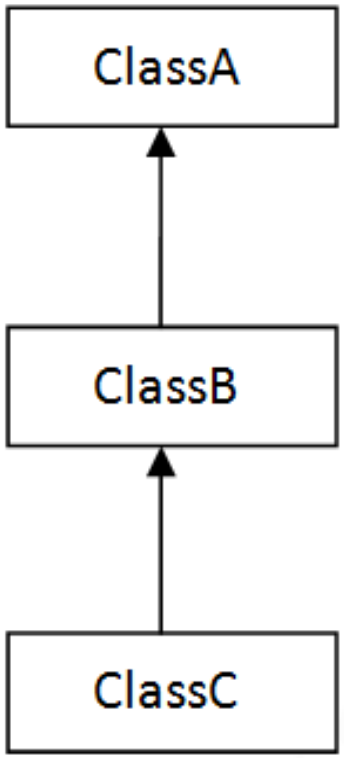
```
class Animal
{

}
class Cat extends Animal
{

}
```



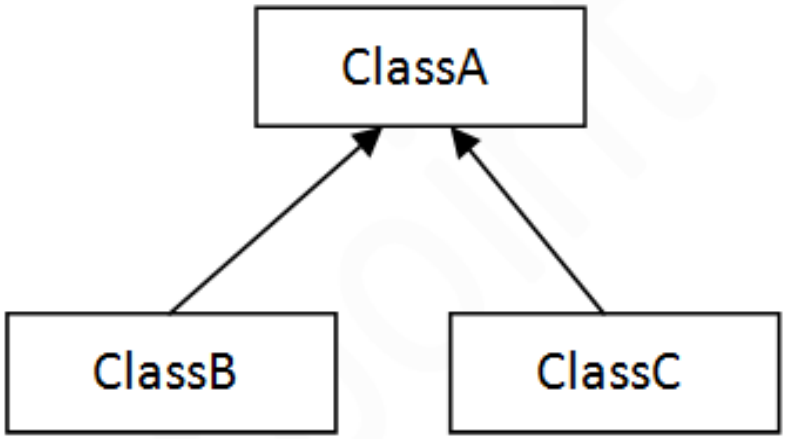
Inheritance



1) Single



2) Multilevel



3) Hierarchical

```
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);
    }

}
```

Relationship

- Inheritance(IS-A)
- Aggregation(HAS-A)

```
class Employee{

        float salary;

}
class Programmer extends Employee{

        int bonus;

}
```

Programmer **IS-A** Employee

```
class Address
{
        String city,state,country;
}

class Employee
{
        int       eid;

        Address   ob1;
}
```

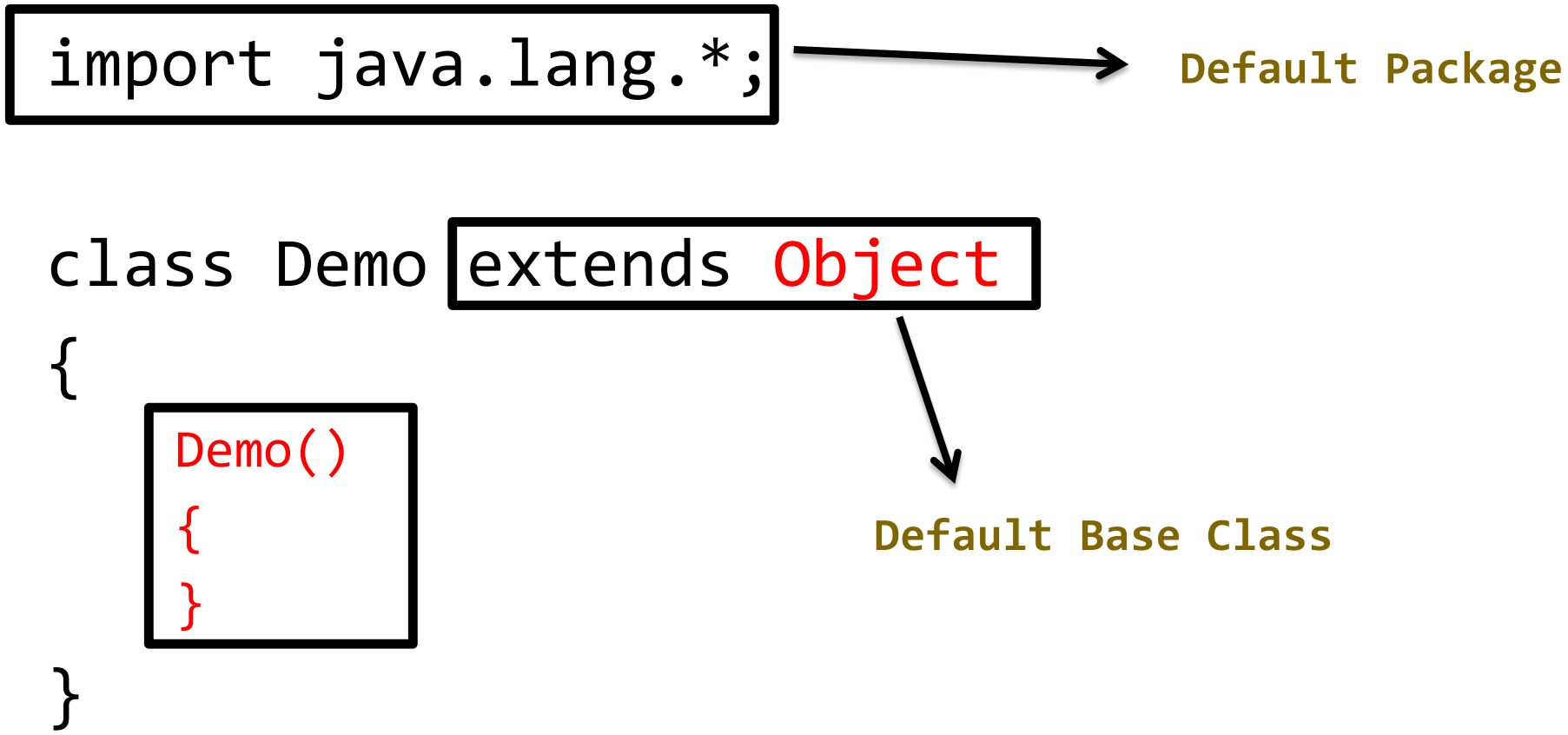
Employee **HAS-A** Address

```
class Address
{
        String city,state,country;
}

class Employee
{
        int      eid;

        Address  ob1 = new Address();
}
```

Employee **HAS-A** Address

```
class Demo
{

}
```

```
class Demo extends Object
{

}
```


import java.lang.*;
Default Package
class Demo extends Object
{
Demo()
{
}
}
Default Base Class

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

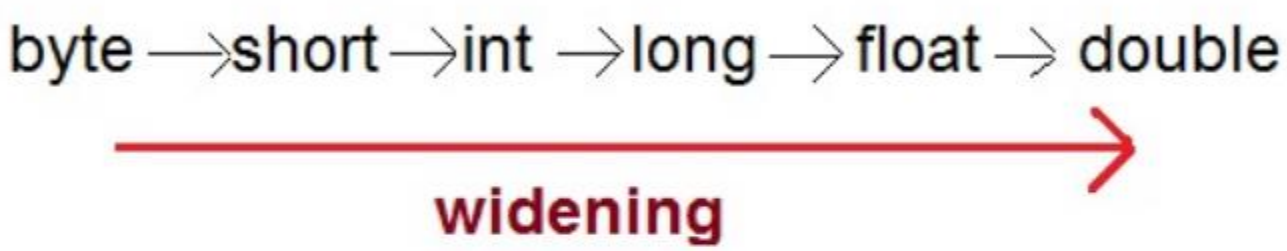
Table 1: List of Java's primitive data types

Type	Size in Bytes	Range
byte	1 byte	-128 to 127
short	2 bytes	-32,768 to 32,767
int	4 bytes	-2,147,483,648 to 2,147,483, 647
long	8 bytes	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4 bytes	approximately ±3.40282347E+38F (6-7 significant decimal digits) *Java implements IEEE 754 standard*
double	8 bytes	approximately ±1.79769313486231570E+308 (15 significant decimal digits)
char	2 byte	0 to 65,536 (unsigned)
boolean	not precisely defined*	true or false

Type Casting

In Java, type casting is classified into two types,

- Widening Casting(Implicit)



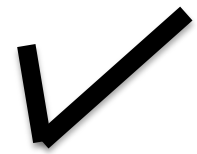
- Narrowing Casting(Explicitly done)

```
int i = 100;

long l = (long) i;

float f = (float) i;
```

```
int i = 100;

long l = i;  //no explicit type casting required

float f = i;  //no explicit type casting required
```

```
int  i  = 100;

long a1  =  i;

long a2  =  (long) i;
```

```
int    a = 100;

char  b = (char)a;
```

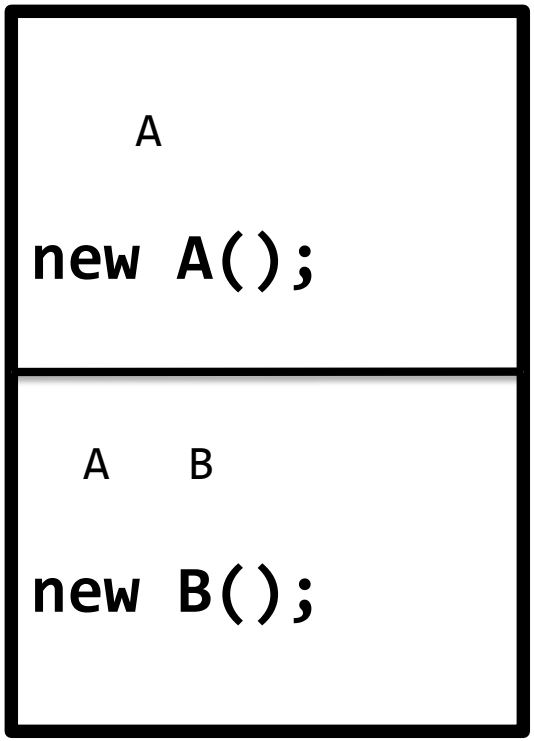


```
int    a = 100;

char  b = a;
```

```
double d = 100.04;

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

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

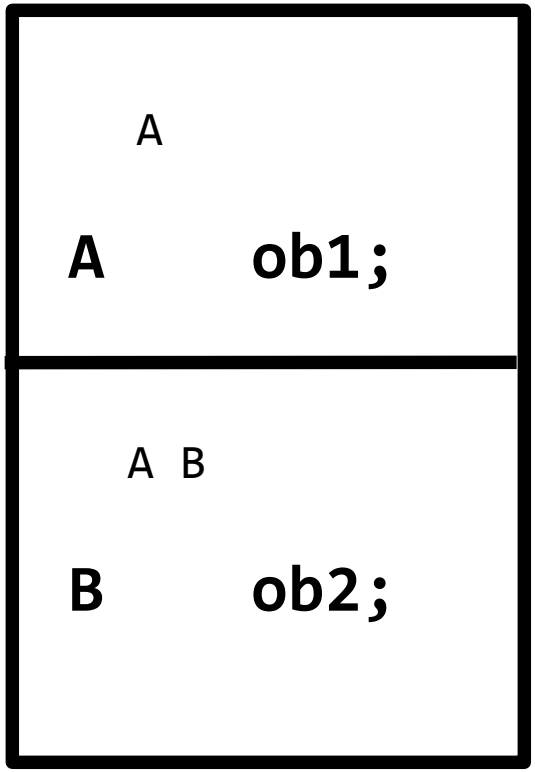
Object Creation

```
class A
{

}
class B extends A
{

}
```



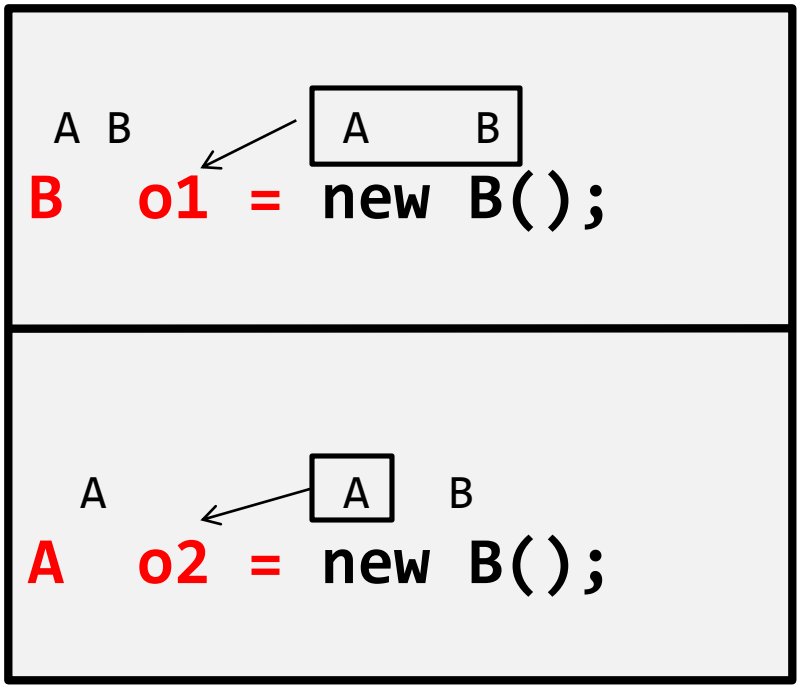
Reference Creation

```
class A
{

}
class B extends A
{

}
```



Class Casting

```
class A
{

}
class B extends A
{

}
```

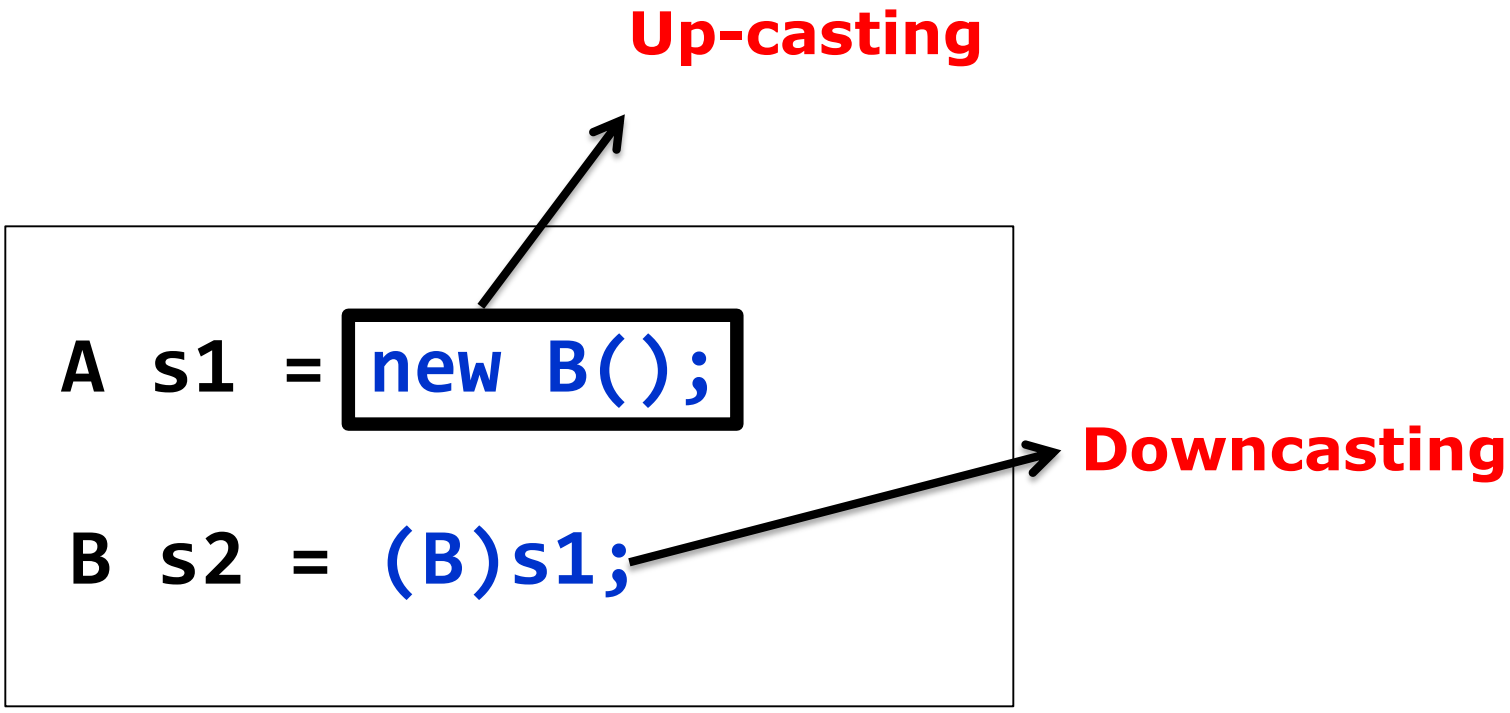


Class Casting

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

```
A s1 = new B();

B s2 = (B)s1;
```

Inheritance

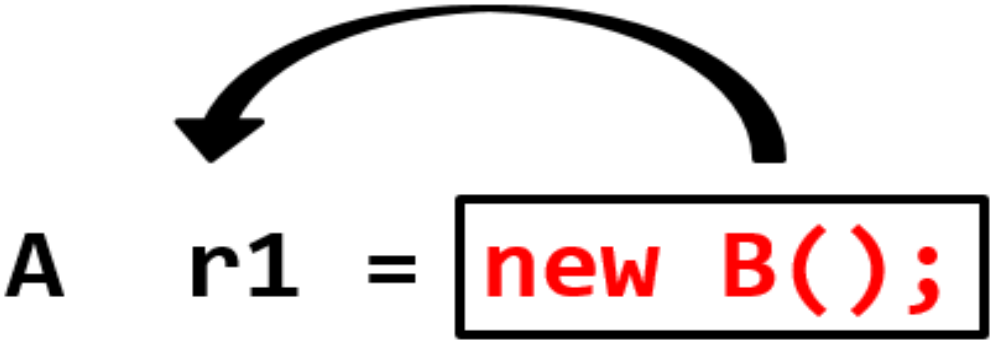


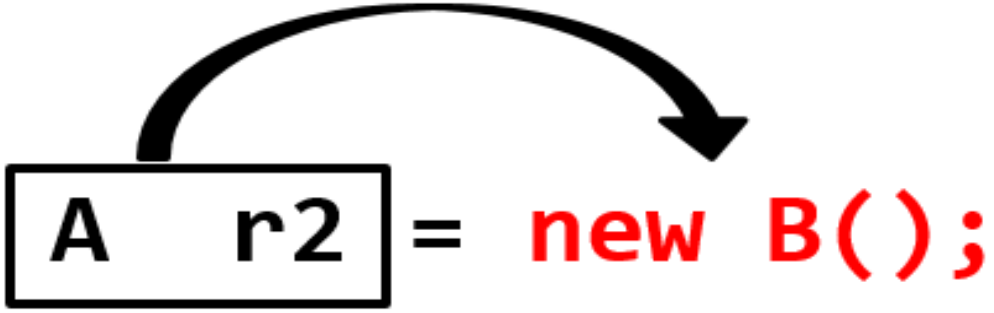
A class **Object** may be under
same-class reference or **base**-class reference

```
B   r1 = new B();

A   r2 = new B();
```

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

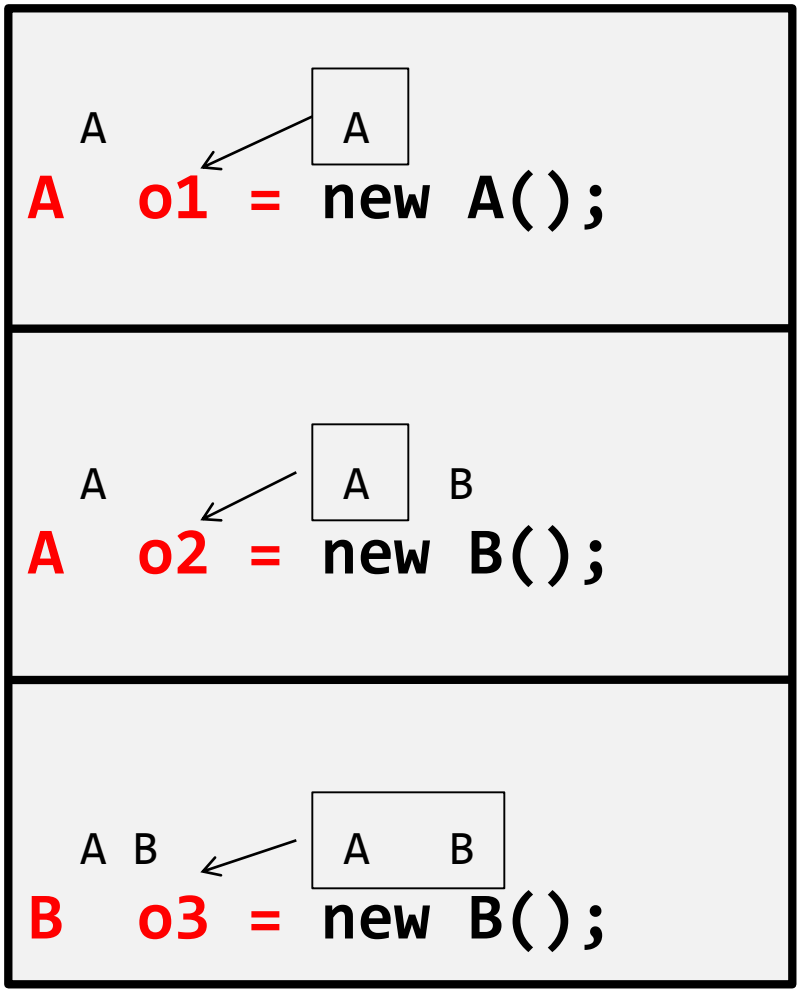
```
A   r1 = new A();

A   r2 = new B();
```


A r1 = new B();


A r2 = new B();

Inheritance

```
class A
{

}
class B extends A
{

}
```



```
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 = new B();

A ob3 = (A)new B();

A ob4 = ob1;

A ob5 = (A)ob1;
```

```
B ob1 = new B();

A ob2 = ob1;

A ob3 = (A)ob1;

B ob4 = ob2;

B ob5 = (B)ob2;
```

```
B ob1 = new B();
```

```
A ob2 = ob1;
```

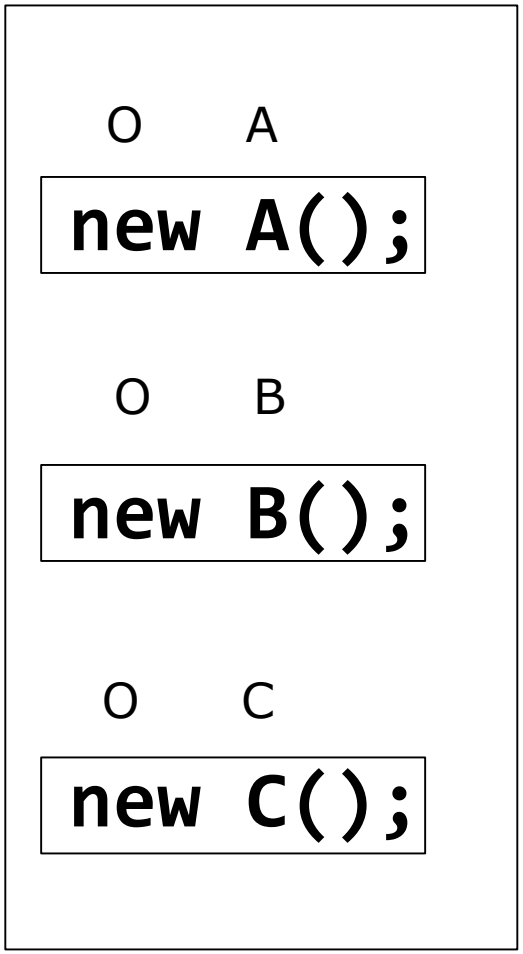
incompatible types: A cannot be converted to B

(Alt-Enter shows hints)

```
B ob4 = ob2;
```

```
B ob5 = (B)ob2;
```

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



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

```
A o1 = new A();

B o2 = new B();

C o3 = 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();
```

Inheritance

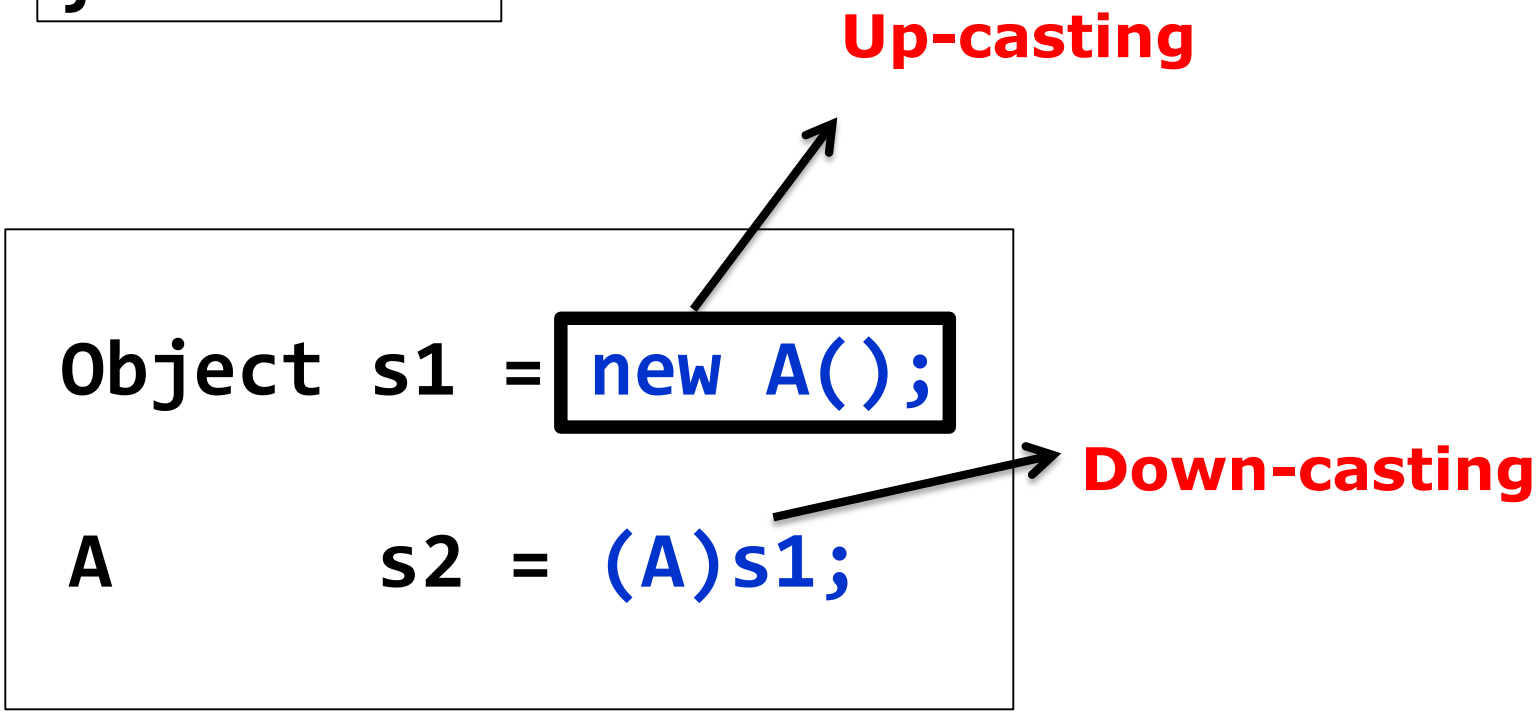
```
class A
{
}
class B
{
}
class C
{
}
```

O A

```
Object s1 = new A();

A      s2 = (A)s1;
```

Inheritance

```
class A
{
}
```



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)
     {
     }
}
```

Methods Overloading

There are two ways to overload the method in java

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

Methods Overloading

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

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

Methods

```
void calculation();
```

Method Declaration

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

Method Declaration

Method Definition

Methods

```
void calculation();
```

→ **Abstract Method**

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

→ **Non-Abstract Method**

or

Concrete Method

Method

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

Methods Overriding

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

}
```

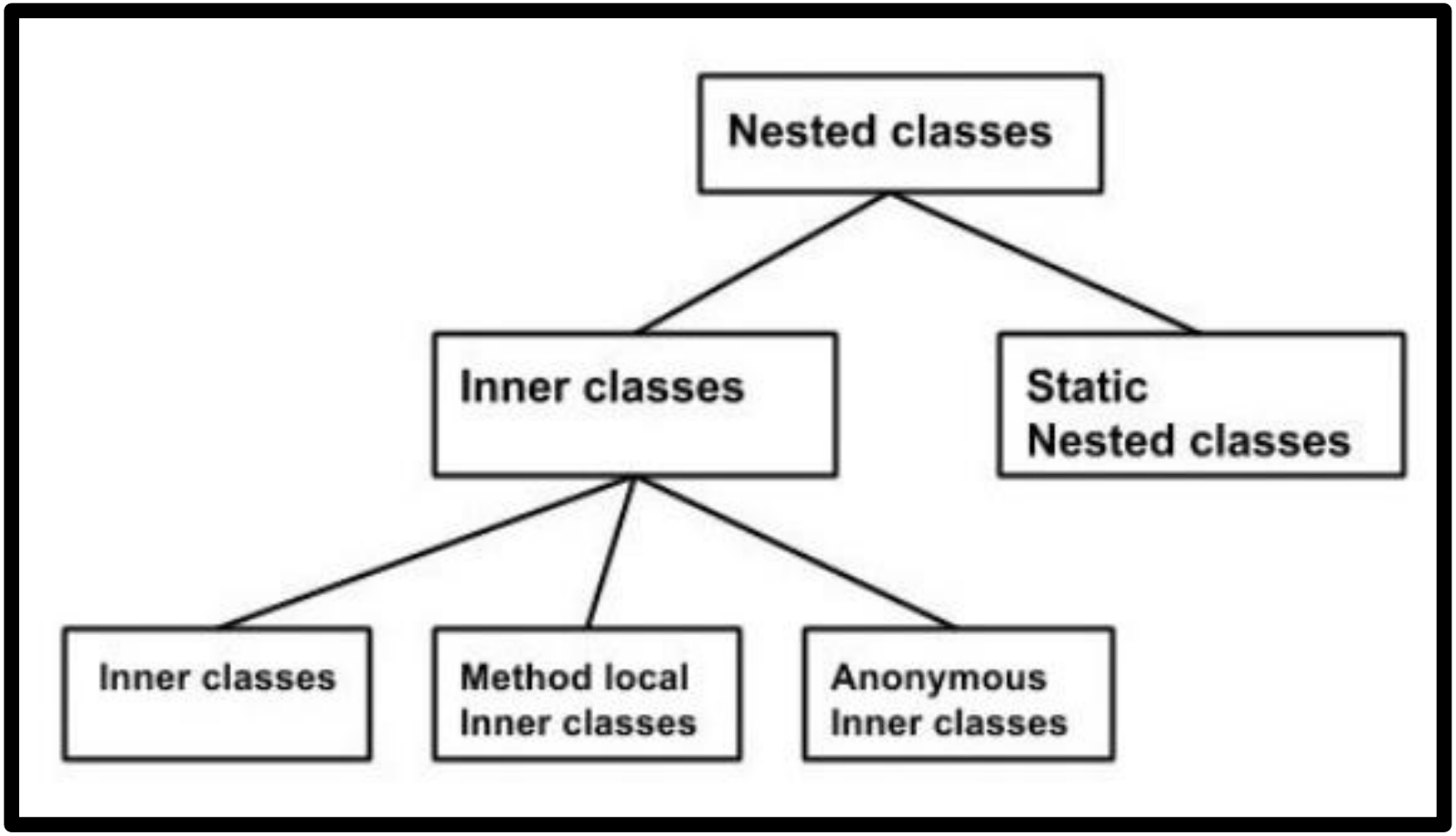
Method Overriding

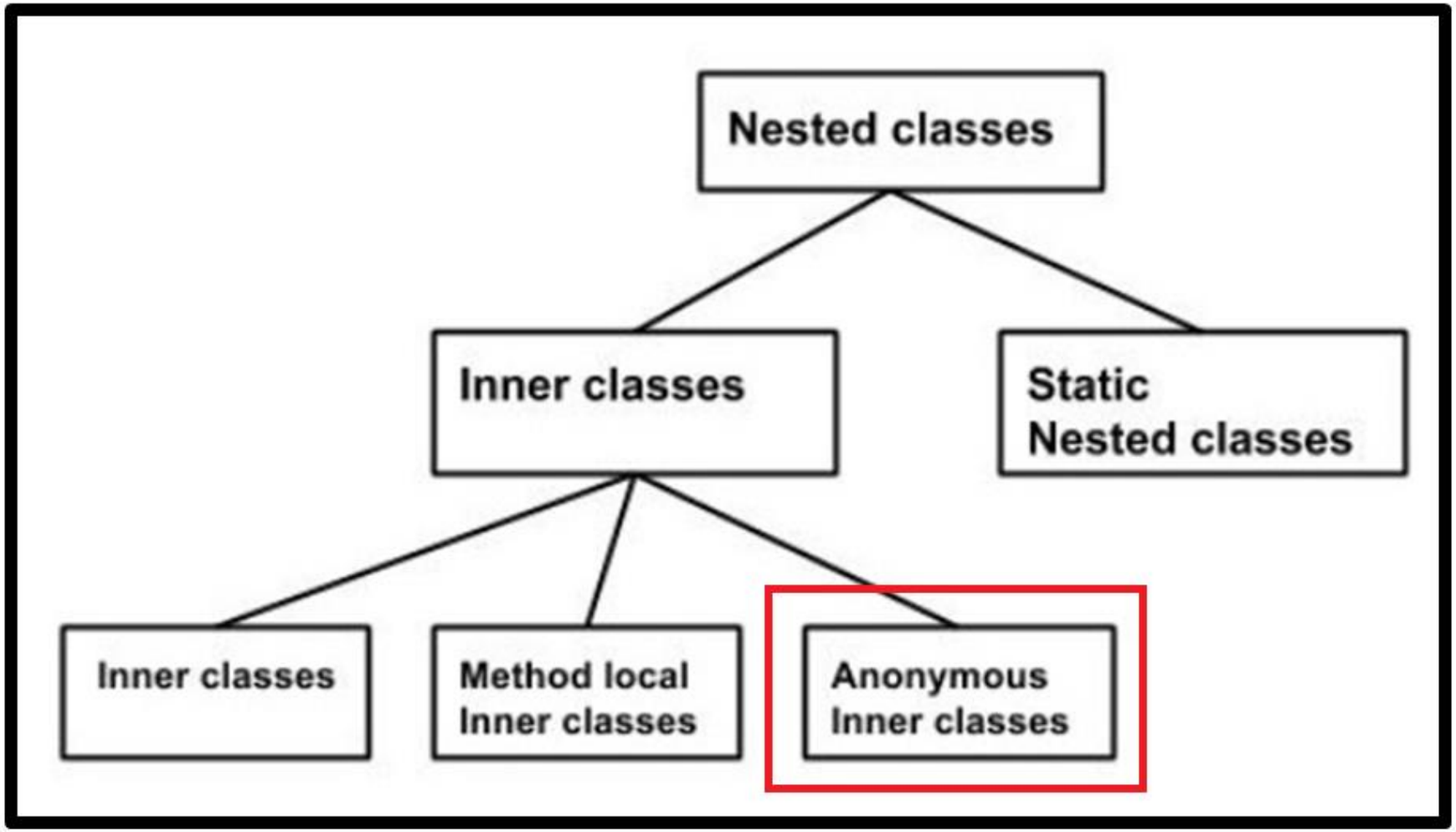
```
class A
{
     void add()
     {
       System.out.println(“Hai”);
     }
}
class B extends A
{
     void add()
     {
        System.out.println(“Welcome”);
     }
}
```

Method Overriding

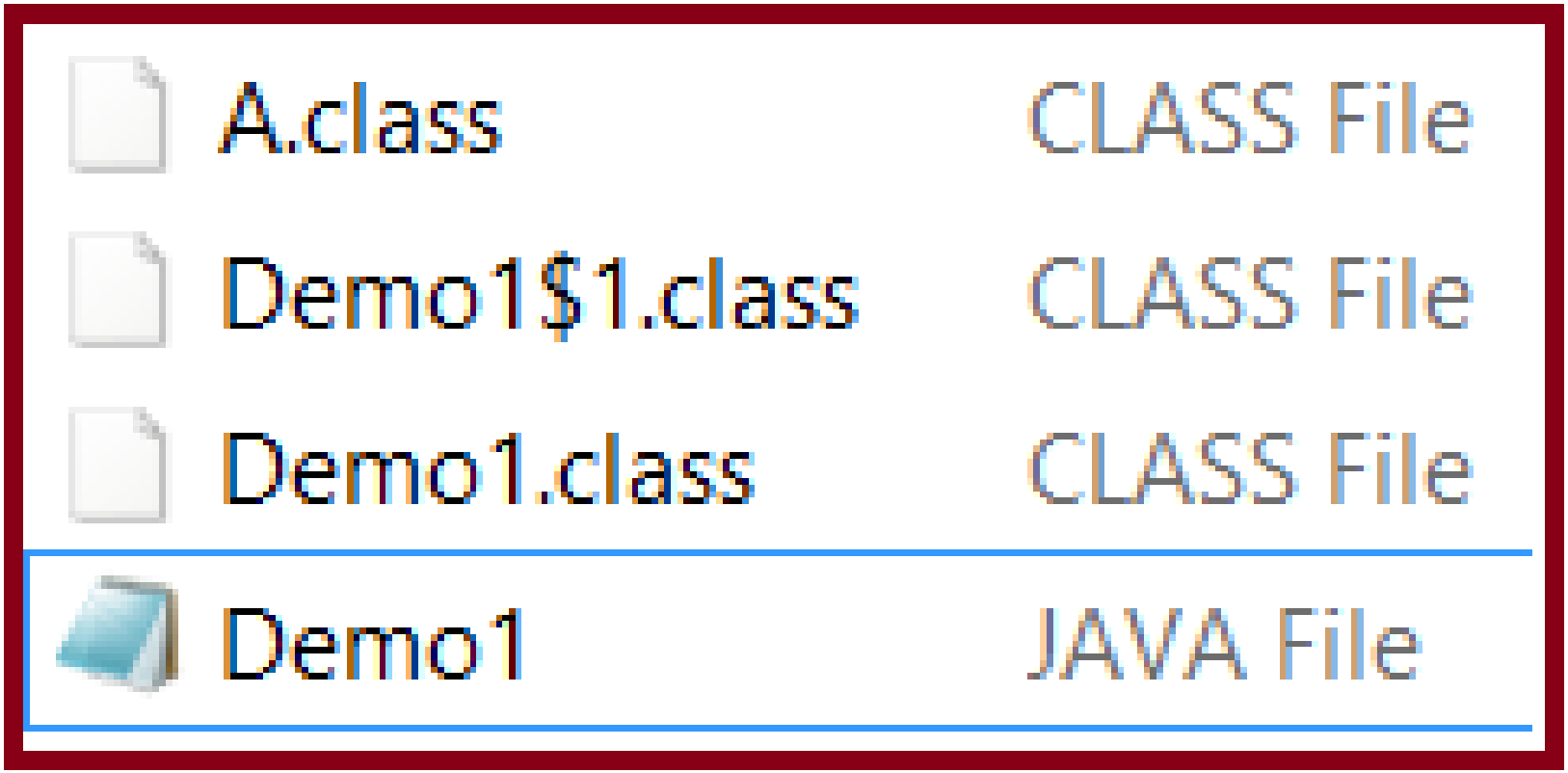
```
class A
{
      abstract void add();
}
class B extends A
{

}
```

Java - Inner classes




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

```
class A{

}

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

                new A() {  };

        }
}
```


A.class
CLASS File
Demo1$1.class
CLASS File
Demo1.class
CLASS File
Demo1
JAVA File

```
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 ");
        }
};
```

Method Overriding

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

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

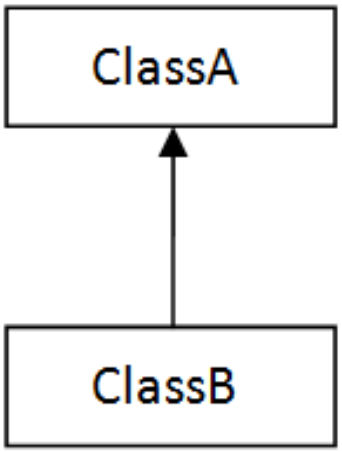
Usage of Java super Keyword

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.

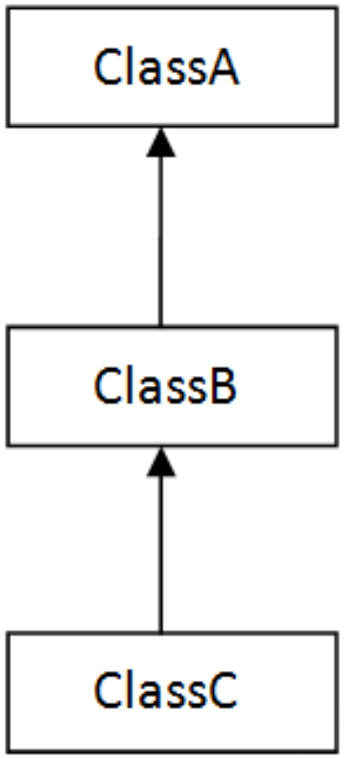
Inheritance

- Single
- Multilevel
- Hierarchal
- Multiple
- Hybrid

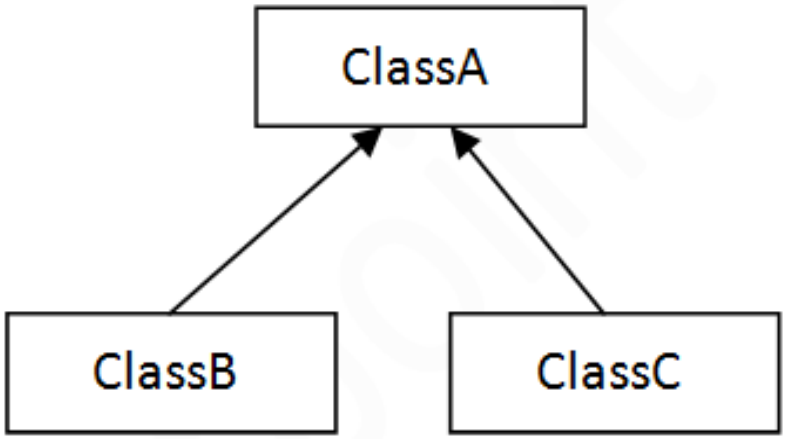
Inheritance



1) Single

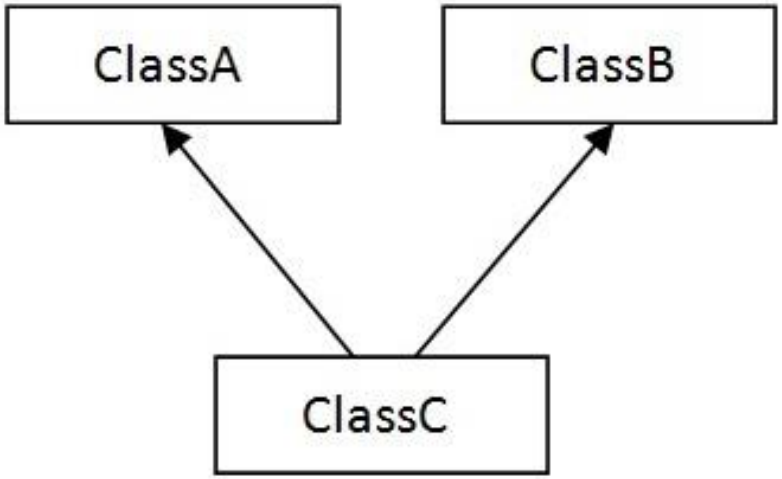


2) Multilevel

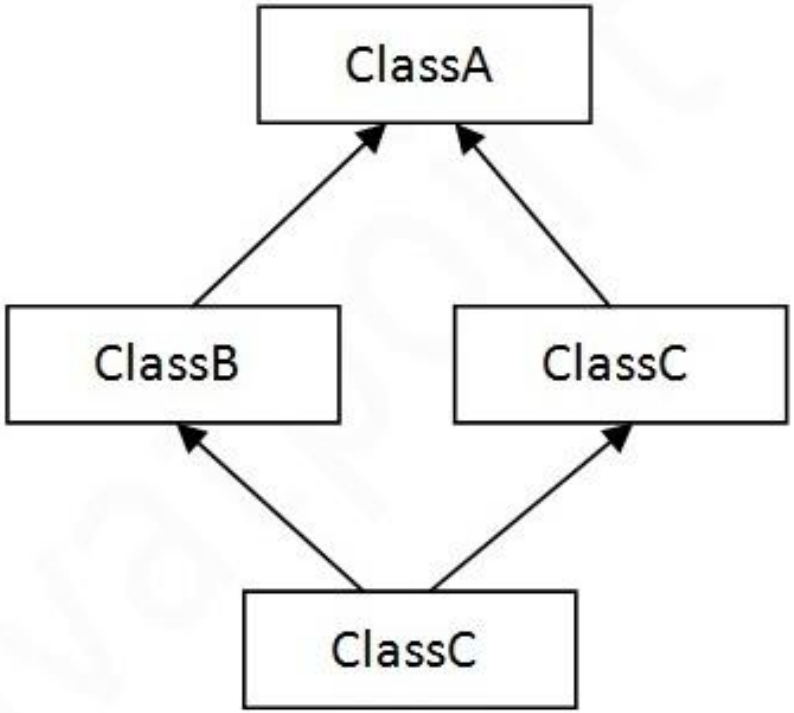


3) Hierarchical

Inheritance



4) Multiple

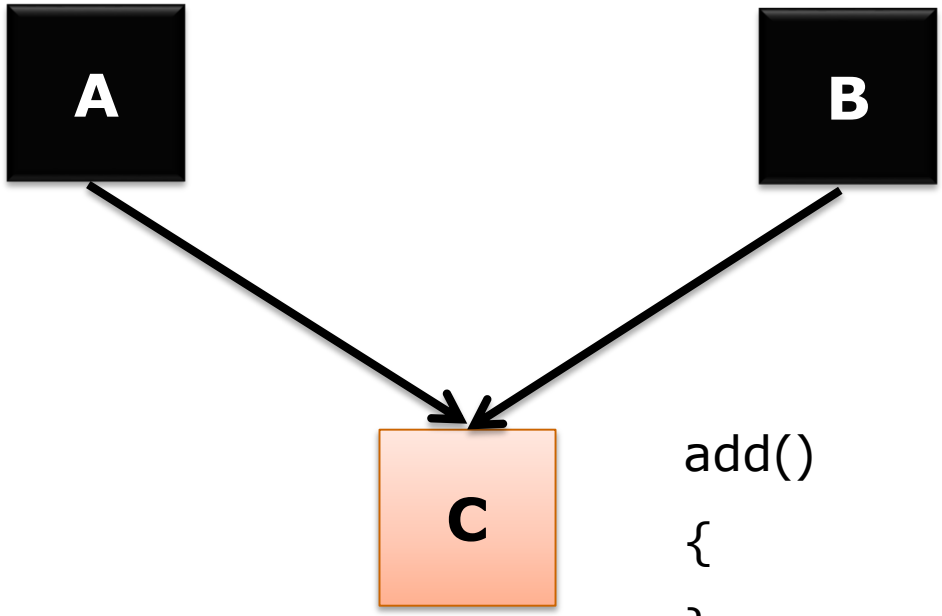


5) Hybrid

Multiple Inheritance

```
void add()
{
    Hai
}
```

```
void add()
{
    Welcome
}
```



Class

```
class Demo1
{

}
```

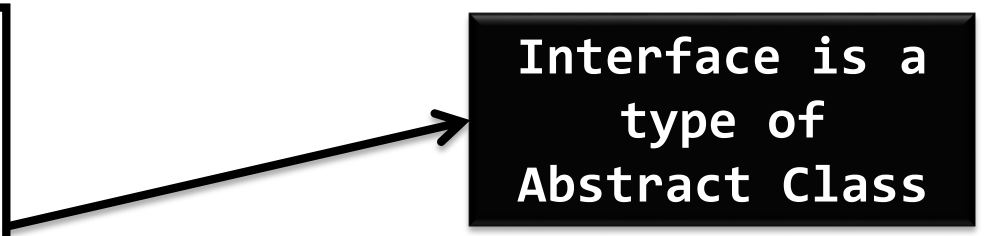
Concrete Class

```
abstract class Demo1
{

}
```

Abstract Class

Interface

```
interface Car
{

}
```



Methods

```
void calculation();
```

→ **Abstract Method**

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

→ **Non-Abstract Method**

or

Concrete Method

Method

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

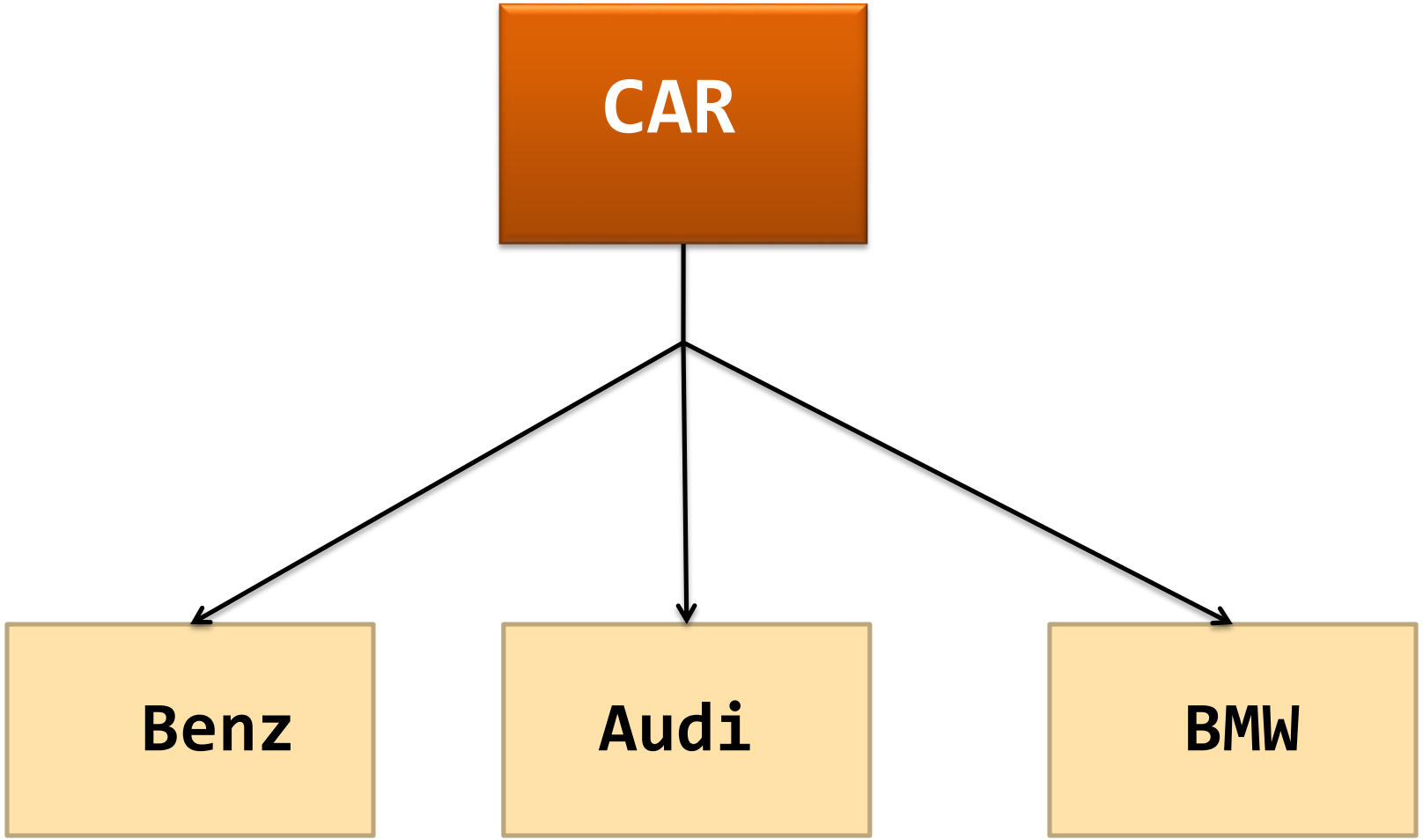
Abstract Class

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

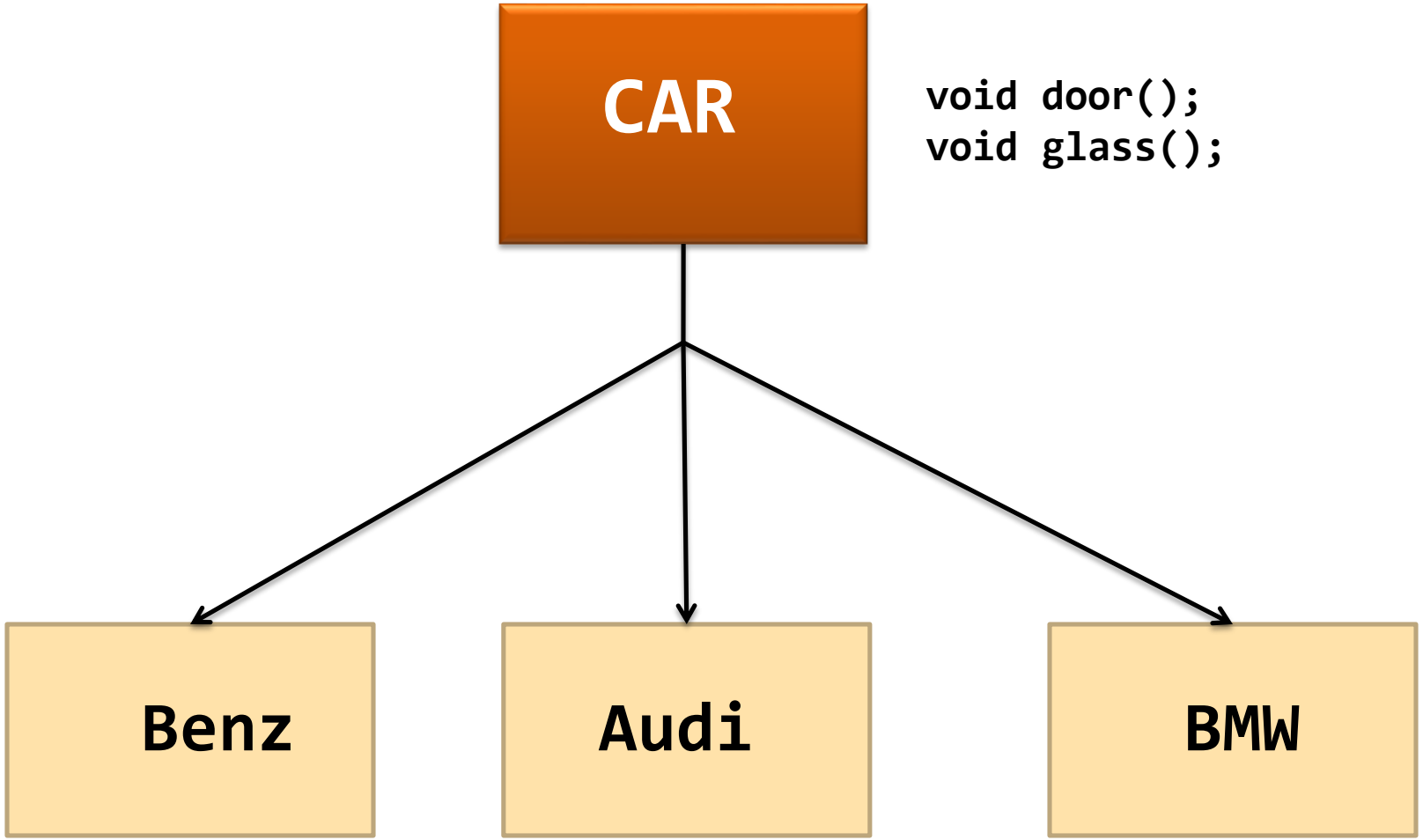
Abstract Class

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

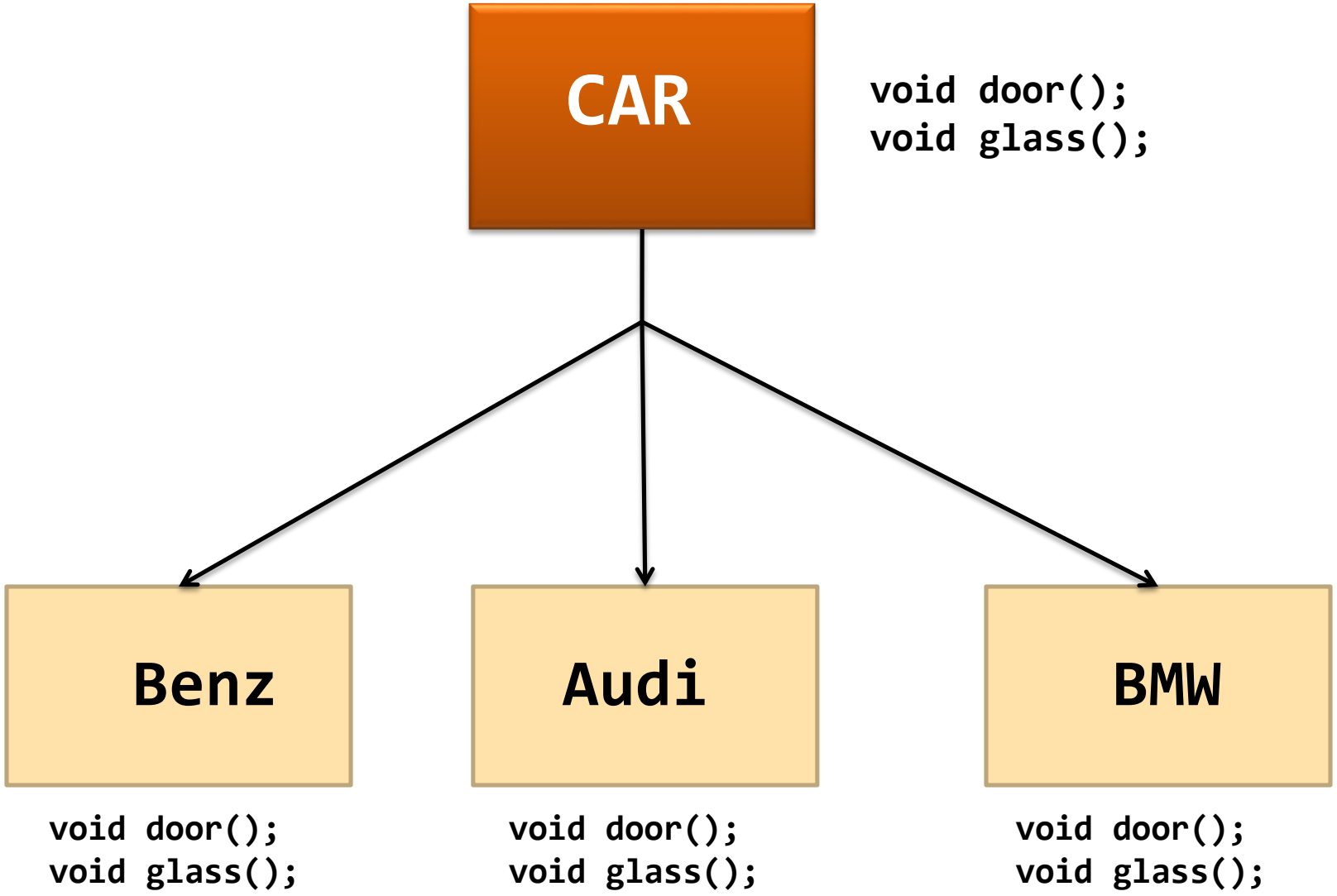
Abstract Class



Abstract Class



Abstract Class



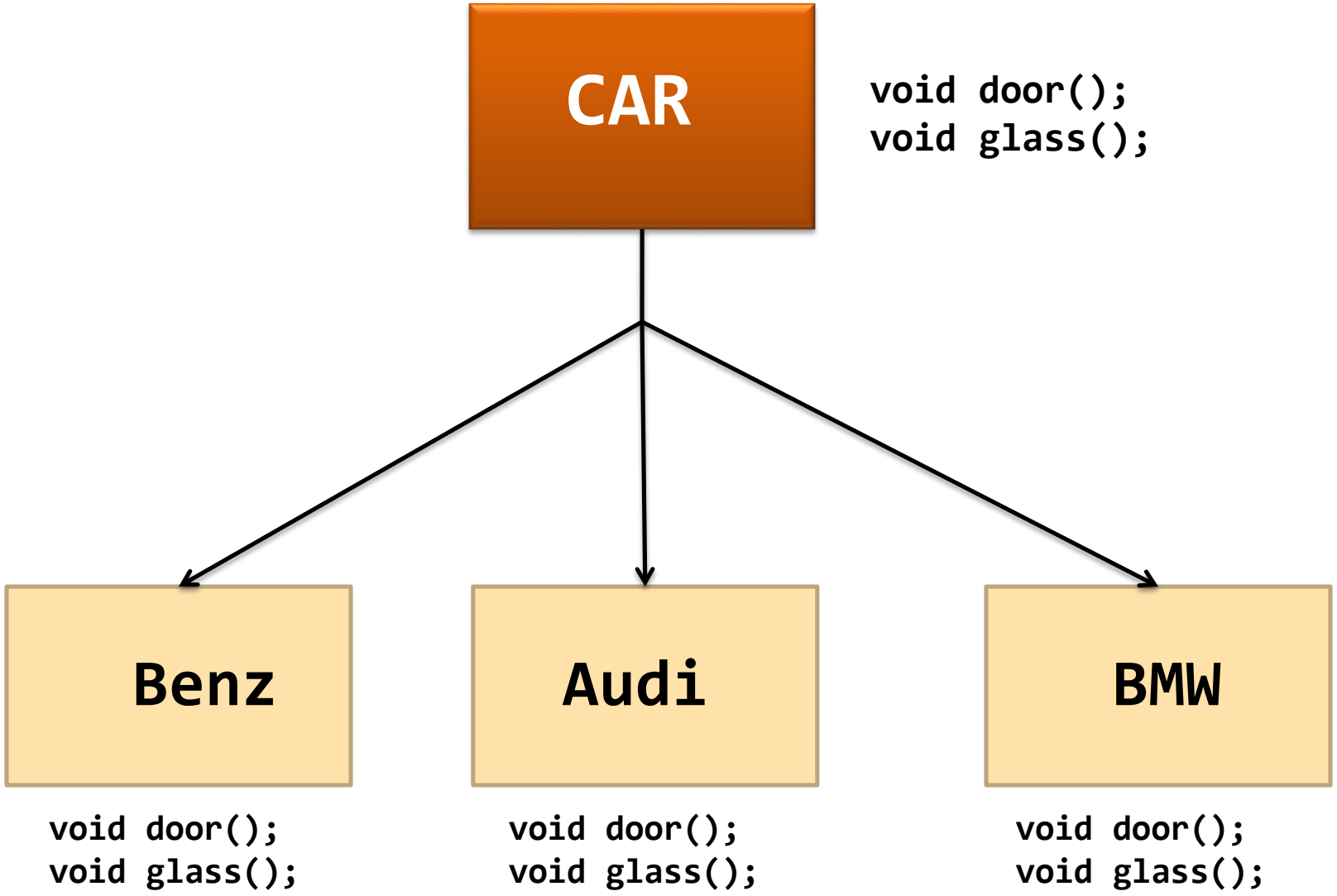
Abstract Class

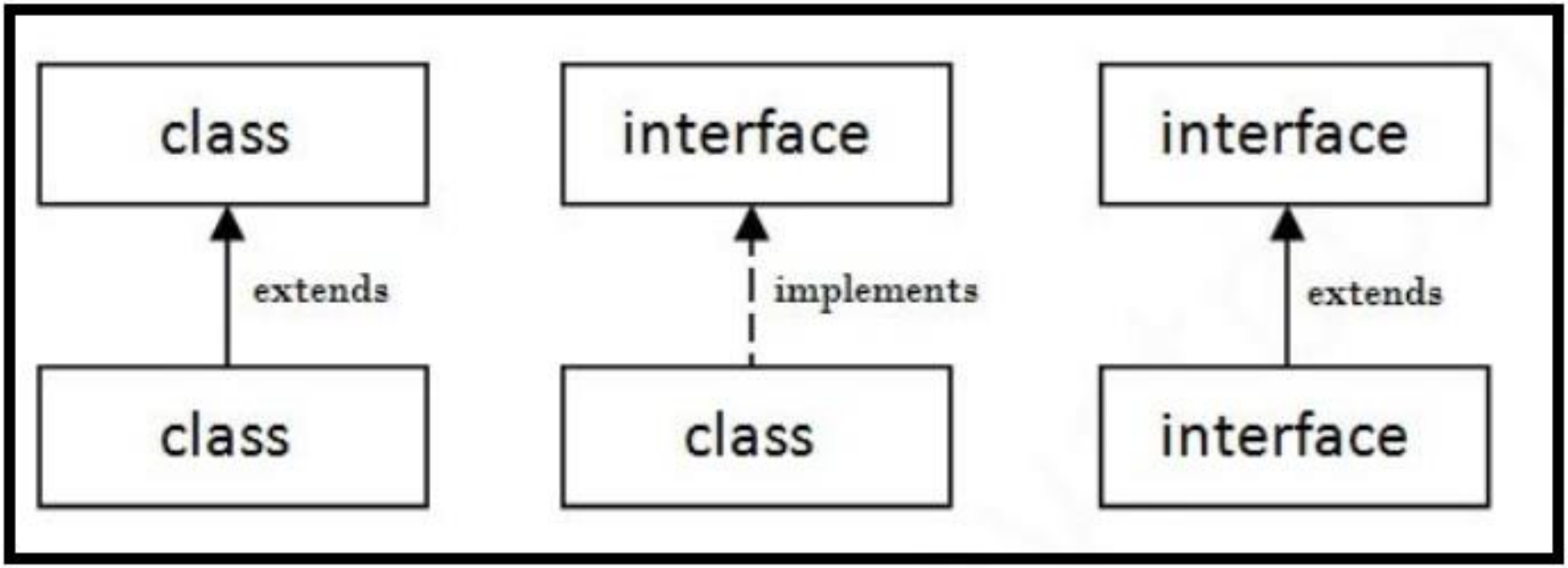
```
class Lancer  extends  Car
{
        void door()
        {
             System.out.println(“Lancer door”);
        }
        void glass()
        {
             System.out.println(“Lancer glass”);
        }
}
```

Abstract Class vs. Interface

Abstract Class	Interface
▫ Contains abstract and concrete methods.	▫ Contains only abstract methods.

Interface


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

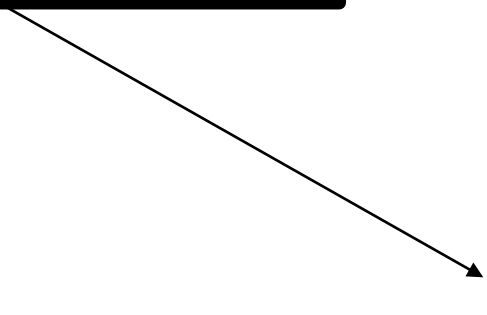

class
interface
interface
extends
implements
extends
class
class
interface

Interface

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

Interface

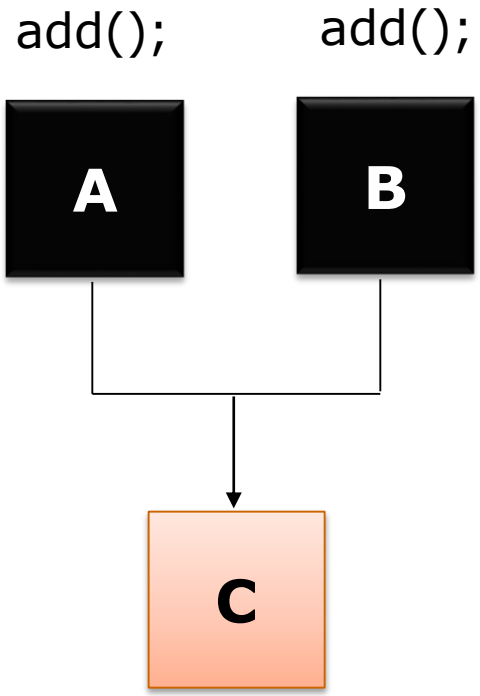
```
interface Car
{
    public abstract   void door();
    public abstract   void glass();
}
```



Interface

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

Multiple Inheritance



Object Creation

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

new Mail();

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

new Car();

Object Creation

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

new Yahoo();

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

new Benz();

Object Creation

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

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

Object Creation

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

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

```
interface A{

}
class B extends C  implements A{

}
class C {

}
```

```
B ob1 = new B();

C ob2 = ob1;

A ob3 = ob1;

B ob4 = ob2;

B ob5 = (B) ob2;
```