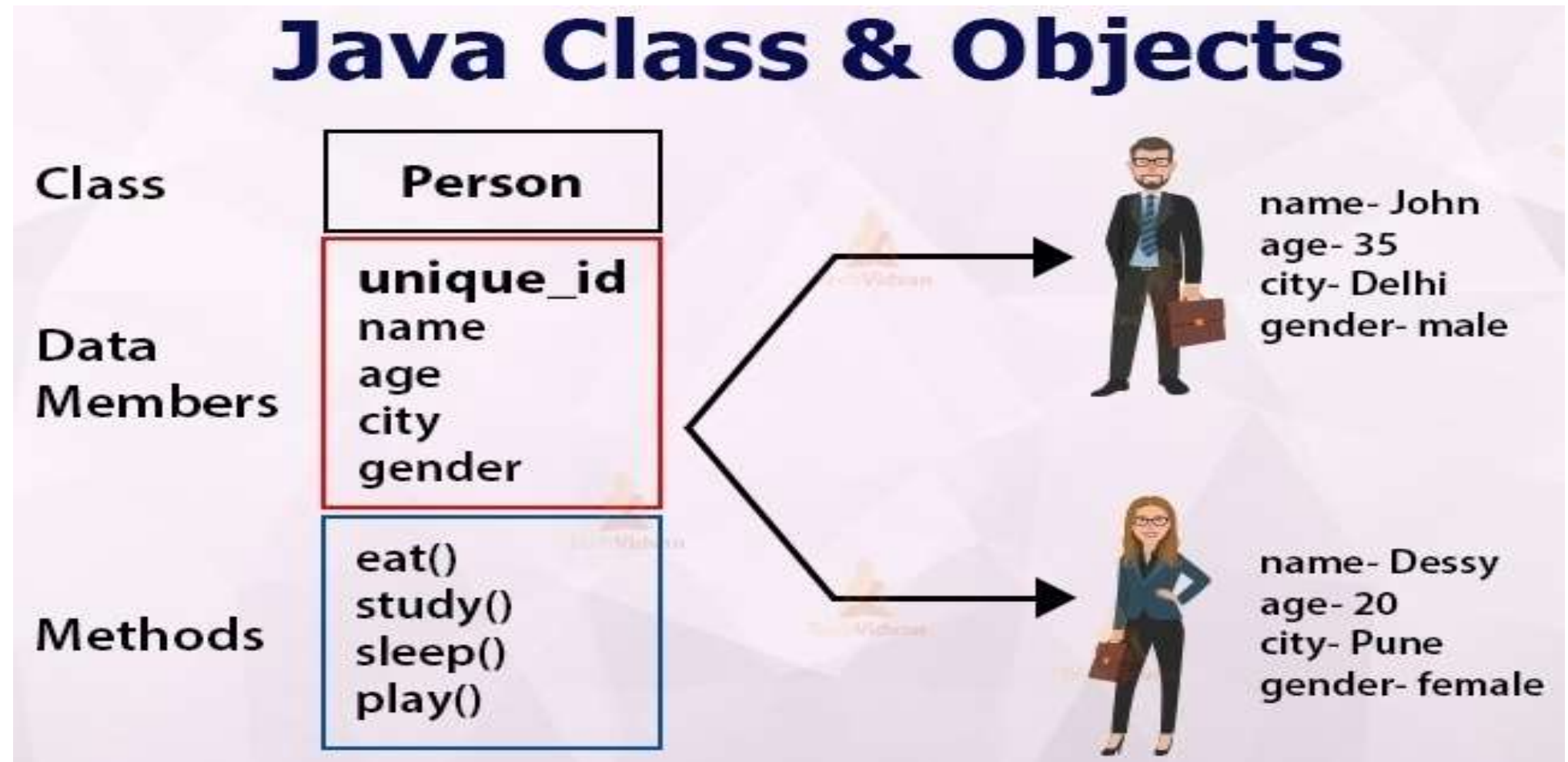


# Java Classes/Objects

- Everything in Java is associated with classes and objects, along with its attributes and methods.
- Class is a keyword in java it is used to defines a new data type. This new type can be used to create objects of that type.
- Thus, a class is a template for an object, and an object is an instance of a class. object and instance used interchangeably.
- A Class is like an object constructor, or a "blueprint" for creating objects.
- For example: in real life, a car is an object. The car has **attributes**, such as weight and color, and **methods**, such as drive and brake.

# Java Classes/Objects



# General Form of a Class

```
class classname
{
type instance-variableN;
type methodname1(parameter-list)
{
    // body of method
}
type methodnameN(parameter-list)
{
    // body of method
}
}
```

```
class Box {
double width;
double height;
double depth;

void volume() {
System.out.println("Volume is ");
System.out.println(width * height * depth);
}
}
```

# Object Creation

- To create a **Box** object,

**Box mybox = new Box();**


- Mybox is consider as Instance/object of class Type Box

```
class C1
{
.....
.....
}
```

**Has-A Relation**

```
class C2
{
.....
.....
}

C1 obj = new C1();
```



# Constructor

- **Constructor in java** is a special type of method that is used to initialize the object.
- Java constructor is invoked at the time of object creation.

## Rules for creating java constructor:

- Constructor name must be same as its class name
- Constructor must have no explicit return type. implicit return type of a class constructor is the **class type itself**.

# Types of constructor

**1. Default constructor**

**2. Parameterized constructor**

# Default constructor

- Java compiler automatically creates a default constructor (Constructor with no arguments) in case no constructor is present in the java class.

**Following are the motive behind a default constructor**

- Create the Object
- Call the super class constructor()
- Initialize all the instance variables of the class object.

# Default constructor

```
class Test {  
    int i;  
    Test t;  
    boolean b;  
    float ft;  
    public static void main(String args[]) {  
        Test obj = new Test(); // default constructor  
        System.out.println(obj.i);  
        System.out.println(obj.t);  
        System.out.println(obj.b);  
        System.out.println(obj.ft);  
    }  
}
```



# Parameterized constructor

- A constructor which has a specific number of parameters is called a parameterized constructor.
- The parameterized constructor is used to provide different values to distinct objects

# Parameterized constructor

```
class Student4
{
    int id;
    String name;
    Student4(int i, String n)
    {
        id = i;
        name = n;
    }
    void display() //simple method
    {
        System.out.println(id+" "+name);
    }
}
```

```
public static void main(String args[])
{
    Student4 s1 = new Student4(111,"karan");
    Student4 s2 = new Student4(222,"Aryan");
    s1.display();
    s2.display();
}
}
```

# 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.

# Constructor Overloading

```
class Student5{  
    int id;  
    String name;  
    int age;  
    Student5(int i, String n){  
        id = i;  
        name = n;  
    }  
    Student5(int i, String n,int a){  
        id = i;  
        name = n;  
        age=a; }  
}
```

```
void display1(){System.out.println(id+" "+name);}  
void display2(){System.out.println(id+" "+name+" "+age);  
    }  
  
    public static void main(String args[])  
    {  
        Student5 s1 = new Student5(111,"Karan");  
        Student5 s2 = new Student5(222,"Aryan",25);  
        s1.display1();  
        s2.display2();  
    }  
}
```

# Constructor Chaining

- Constructor chaining is the process of calling one constructor from another constructor with respect to current object.
- One of the main use of constructor chaining is to avoid duplicate codes while having multiple constructor (by means of constructor overloading) and make code more readable.

## Constructor chaining can be done in two ways:

- **Within same class:** It can be done using **this()** keyword for constructors in the same class
- **From base class:** by using **super()** keyword to call the constructor from the base class.
- **Why do we need constructor chaining?**
- This process is used when we want to perform multiple tasks in a single constructor rather than creating a code for each task in a single constructor we create a separate constructor for each task and make their chain which makes the program more readable.

# Constructor Chaining using this keyword

```
class Temp
{
    Temp()
    {
        this(5);
        System.out.println("The Default constructor");
    }
}
```

```
    Temp(int x)
    {
        this(5, 15);
        System.out.println(x);
    }
    Temp(int x, int y)
    {
        System.out.println(x * y);
    }
}
```

```
public static void main(String args[])
{
    // invokes default constructor first
    new Temp();
}
```

**Out put**  
75  
5  
The Default constructor

# Constructor Chaining using this keyword

```
class Temp
{
    Temp()
    {
        System.out.println("default");
    }

    Temp(int x)
    {
        this();
        System.out.println(x);
    }

    Temp(int x, int y)
    {
        this(5);
        System.out.println(x * y);
    }
}
```

```
public static void main(String args[])
{
    new Temp(8, 10);
}
```

**Out put**  
default  
5  
80

# Constructor Chaining using super() keyword

class Base

```
{
    String name;
    Base()
    {
        this(" ");
        System.out.println("base default constructor");
    }
    Base(String name)
    {
        this.name = name;
        System.out.println("Calling base parameterized
        constructor ");
    }
}
```

class Derived extends Base

```
{
    Derived()
    {
        System.out.println("default constructor derived");
    }
    Derived(String name)
    {
        super(name);
        System.out.println("Calling derived parameterized
        constructor ");
    }
    public static void main(String args[])
    {
        Derived obj = new Derived("test");
    }
}
```

**Output:** Calling base parameterized constructor  
Calling derived parameterized constructor



## Using init block

- When we want certain common resources to be executed with every constructor we can put the code in the init block.
- Init block is always executed before any constructor, whenever a constructor is used for creating a new object.

# Using init block

```
class Temp
{
    {
        System.out.println("init block");
    }

    Temp()
    {
        System.out.println("default");
    }

    Temp(int x)
    {
        System.out.println(x);
    }

    public static void main(String[] args)
    {
        new Temp();
        new Temp(10);
    }
}
```

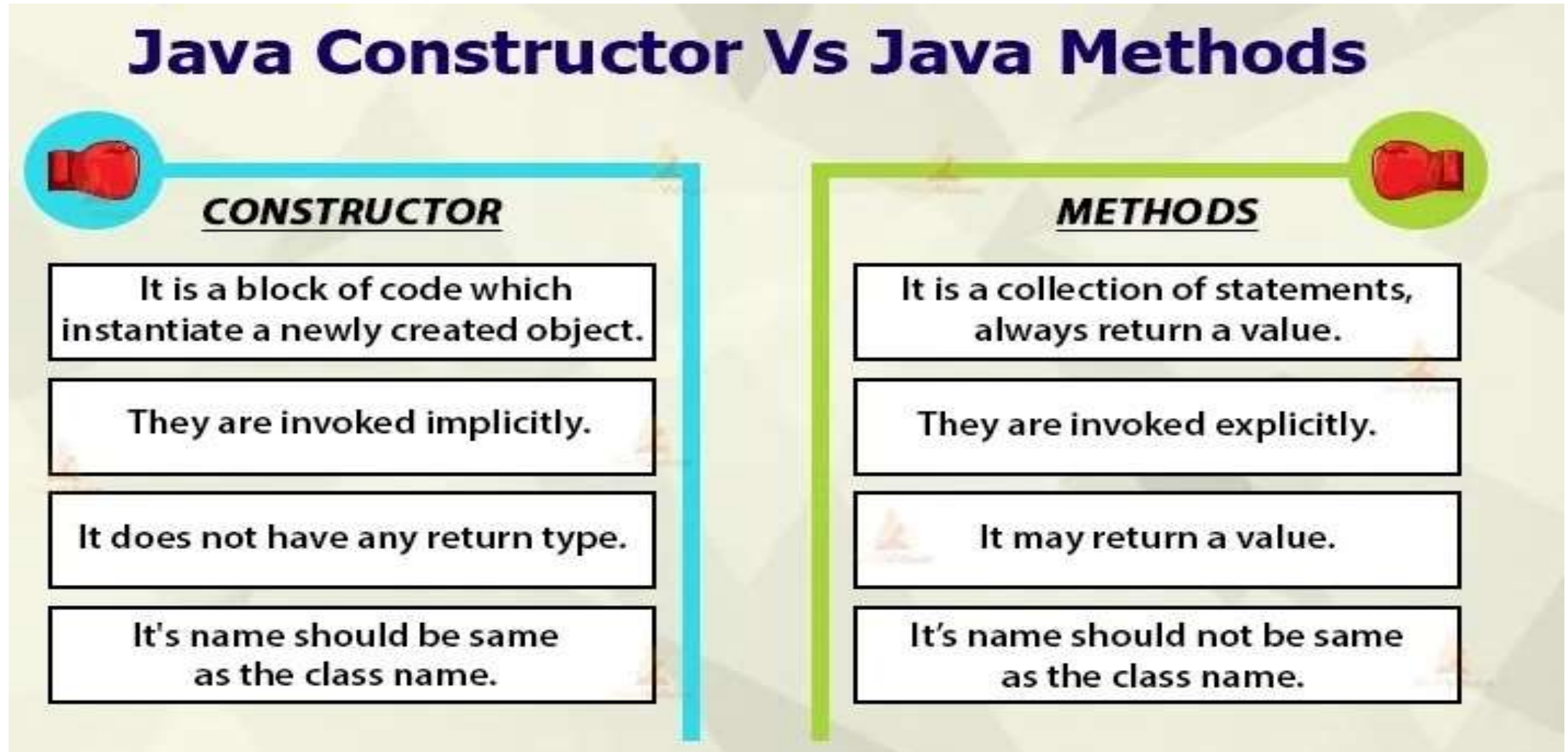
## Output:

```
init block
default
init block
10
```

## Few more points about constructors

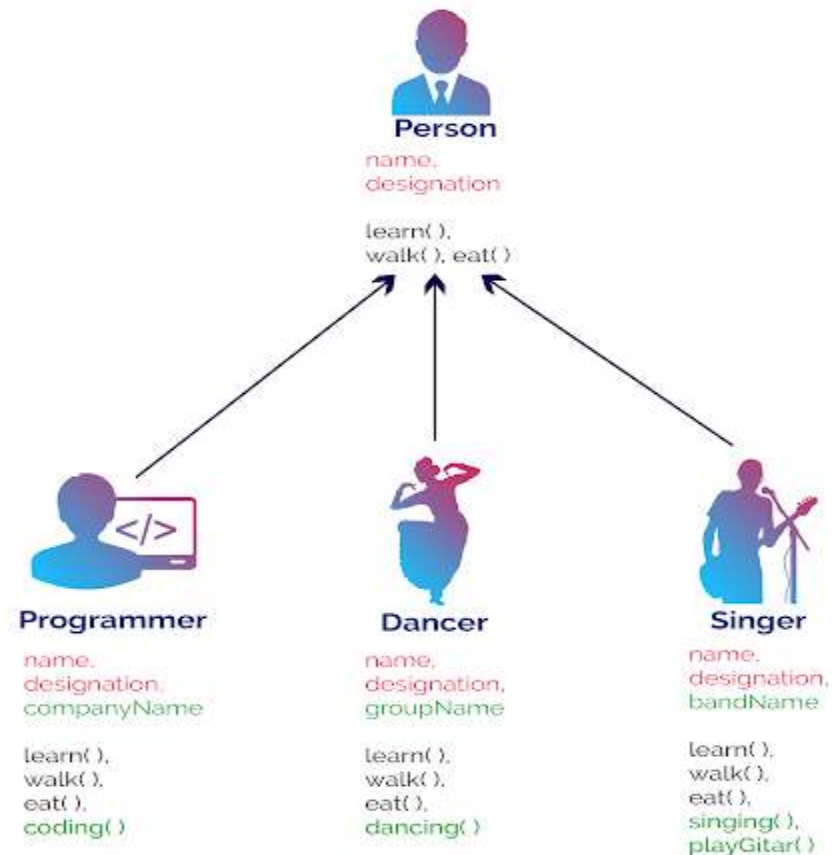
- Constructor can use any access specifier, they can be declared as private also.
- **Private constructors** are possible in java but their scope is within the class only.
- If you don't define any constructor within the class, compiler will do it for you and it will create a no-arg (default) constructor for you.
- **this() and super()** should be the first statement in the constructor code.

# Constructor Vs Methods

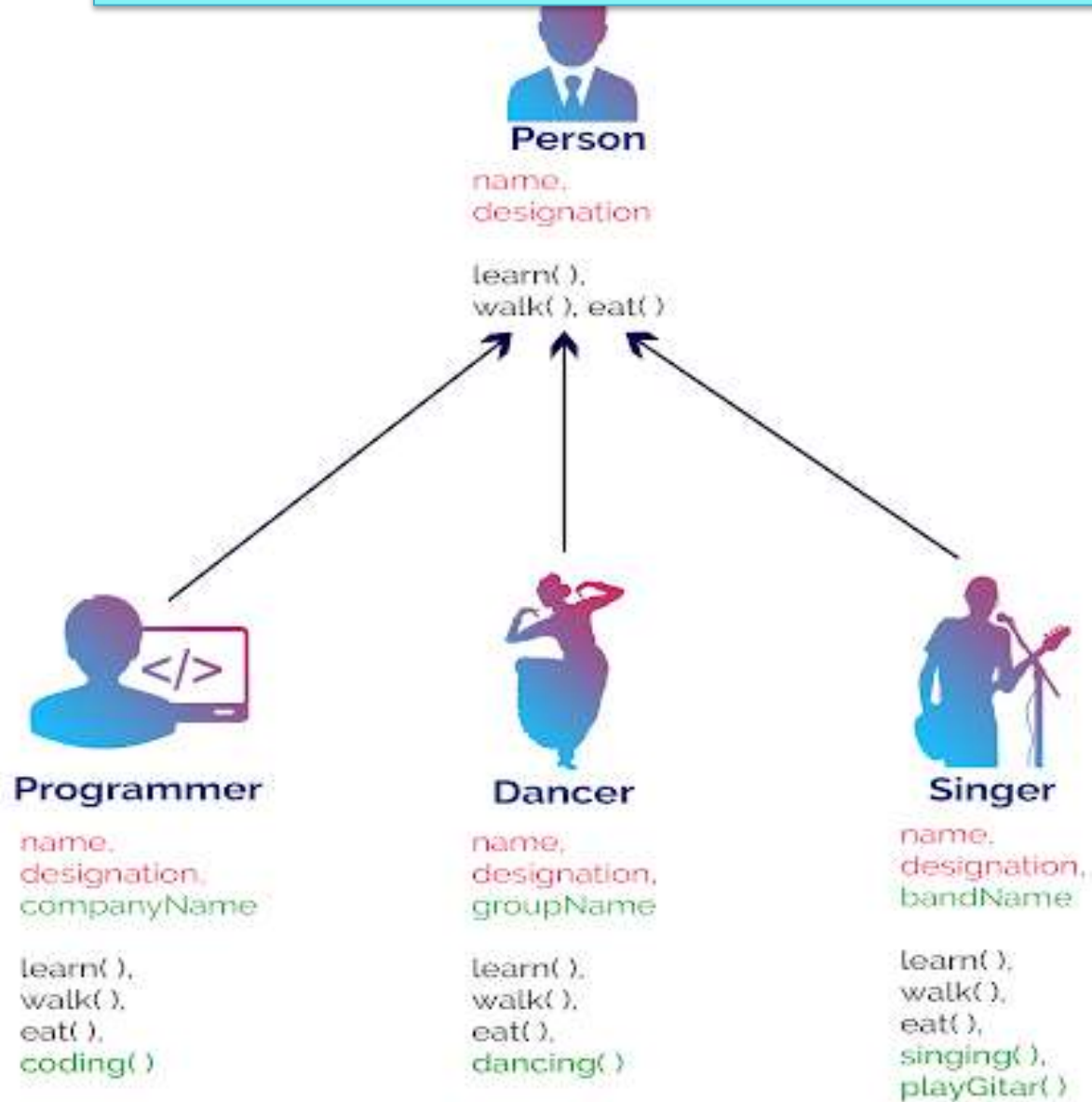


# Inheritance

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

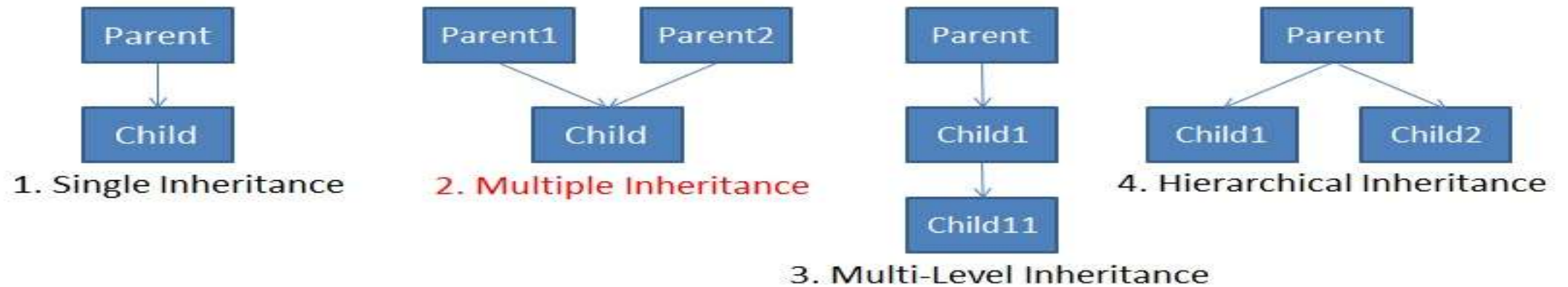


# Inheritance

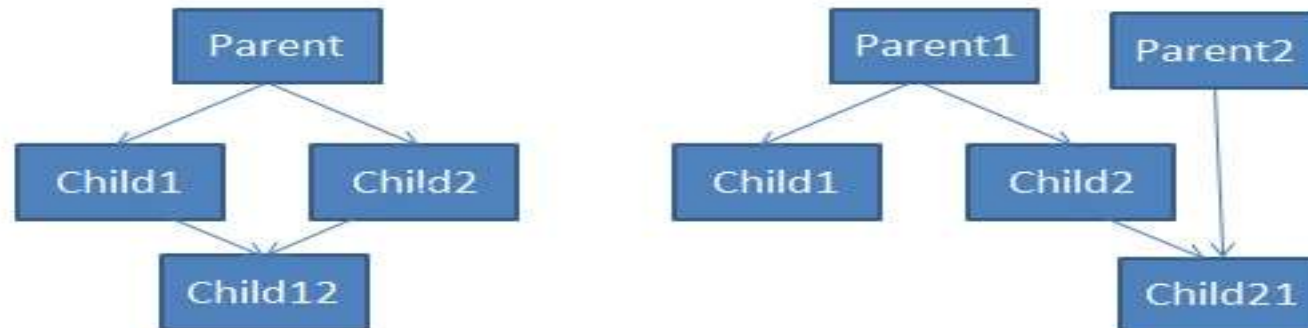


# Types of Inheritance

## Types of Inheritance



## 5. Hybrid Inheritance Variations (Mix of Single & Multiple Inheritance)



Note: The ones marked in red are not supported by Java



# Single level Inheritance

```
class A{  
    void e()  
    {  
        System.out.println("eing...");  
    }  
    class B extends A{  
        void y()  
        {  
            System.out.println("ying...");  
        }  
    }  
    class TestInheritance{  
        public static void main(String args[])  
        {  
            B d=new B();  
            d.y();  
            d.e();  
        }  
    }  
}
```

```
class C1
```

```
{  
    .....  
    .....  
}
```

**Is-A Relation**



```
class C2 extends C1
```

```
{  
    .....  
    .....  
}
```



# Multilevel Inheritance

```
class Animal{  
    void eat(){System.out.println("eating...");}  
}  
  
class Dog extends Animal{  
    void bark(){System.out.println("barking...");}  
}  
  
class BabyDog extends Dog{  
    void weep(){System.out.println("weeping...");}  
}  
  
class TestInheritance2 {  
    public static void main(String args[]) {  
        BabyDog d=new BabyDog();  
        d.weep();  
        d.bark();  
        d.eat();  
    }  
}
```

# Polymorphism

We will cover three concepts in polymorphism

*1.Method overloading*

*2.Method overriding*

*3.Constructor overloading*

Polymorphism in java is a concept by which we can perform a *single action by different ways*.

**Real life example of polymorphism**

# Abstract class

- **Abstraction** is a process of hiding the implementation details and showing only functionality to the user.
- A class which is declared with the abstract keyword is known as an **abstract class** in Java
- It can have abstract and non-abstract methods (method with the body).
- An abstract class can contain constructors in Java. And a constructor of abstract class is called when an instance of an inherited class is created.

# Abstract class

## Rules for Java Abstract class



**1**

An abstract class must be declared with an abstract keyword.

**2**

It can have abstract and non-abstract methods.

**3**

It cannot be instantiated.

**4**

It can have final methods

**5**

It can have constructors and static methods also.

# Abstract class Example

```
abstract class Bike{  
    abstract void run();  
}  
  
class Honda4 extends Bike{  
    void run(){System.out.println("running safely");}  
    public static void main(String args[]){  
        Bike obj = new Honda4();  
        obj.run();  
    }  
}
```

## Abstract class Example (with constructor)

```
abstract class Base {  
Base() {  
System.out.println("Base Constructor Called");  
}  
abstract void fun();  
}  
class Derived extends Base {  
Derived() {  
System.out.println("Derived Constructor Called");  
}  
}
```

```
void fun() {  
System.out.println("Derived fun() called");  
}  
}  
class Main {  
public static void main(String args[])  
{  
Derived d = new Derived();  
}  
}
```

# Interface

- The **interface** is a mechanism to achieve fully abstraction in java.
- All the data members of interface are **implicitly public static final**.

```
interface Person
{
    int age=19;
    void run();
}
```

Person.java

Compiler

```
interface Person
{
    public static final int age=19;
    public abstract void run();
}
```

Person.class

**There are mainly three reasons to use interface. They are given below.**

- It is used to achieve abstraction.
- It can support the functionality of multiple inheritance.
- It can be used to achieve loose coupling.
- Java **final keyword** is a non-access specifier that is used to restrict a class, variable, and method. If we initialize a variable with the final keyword, then we cannot modify its value. If we declare a method as final, then it cannot be overridden by any subclasses.



# Interface Example

```
interface hello
{
void print();
}

class Hi implements hello{

public void print()
{
    System.out.println("Hello");
}
```

```
public static void main(String args[])
{
    Hi obj = new Hi();
    obj.print();
}
}
```

## Interface Example(Multiple Inheritance)

```
interface AnimalEat {  
    void eat();  
}  
  
interface AnimalTravel {  
    void travel();  
}  
  
class Animal implements AnimalEat,  
AnimalTravel {  
    public void eat() {  
        System.out.println("Animal is  
eating"); }  
}
```

```
public void travel() {  
    System.out.println("Animal is travelling");  
} }  
  
public class Demo {  
    public static void main(String args[]) {  
        Animal a = new Animal();  
        a.eat();  
        a.travel(); } }
```

# Difference between Interface and Abstract class

Abstract class	Interface
1) Abstract class can <b>have abstract and non-abstract</b> methods.	Interface can have <b>only abstract</b> methods. Since Java 8, it can have <b>default and static methods</b> also.
2) Abstract class <b>doesn't support multiple inheritance</b> .	Interface <b>supports multiple inheritance</b> .
3) Abstract class <b>can have final, non-final, static and non-static variables</b> .	Interface has <b>only static and final variables</b> .
4) Abstract class <b>can provide the implementation of interface</b> .	Interface <b>can't provide the implementation of abstract class</b> .
5) The <b>abstract keyword</b> is used to declare abstract class.	The <b>interface keyword</b> is used to declare interface.
6) An <b>abstract class</b> can be extended using keyword "extends".	An <b>interface</b> can be implemented using keyword "implements".
7) A Java <b>abstract class</b> can have class members like private, protected, etc.	Members of a Java interface are public by default.
8) <b>Example:</b> <pre>public abstract class Shape{ public abstract void draw(); }</pre>	<b>Example:</b> <pre>public interface Drawable{ void draw(); }</pre>

# Super()

- Super keyword in java is a reference variable that is used to refer parent class object.
- Used to refer immediate parent class instance variable.
- Used to invoke immediate parent class methods.
- Used to invoke immediate parent class constructor

# Super can be use to refer immediate parent class instance variable

```
public class Vehicle {  
    int maxSpeed = 120;  
}  
  
public class Car extends Vehicle {  
    int maxSpeed = 180;  
    void display() {  
        System.out.println("Maximum Speed: "+  
            super.maxSpeed);  
    }  
}
```

```
}  
public class GFG {  
    public static void main(String[] args)  
    {  
        Car small = new Car();  
        small.display();  
    }  
}
```

# Super can be use to refer immediate parent class instance variable

```
class Emp
{
    float salary=10000;
}
class HR extends Emp
{
    float salary=20000;
    void display()
    {
        System.out.println("Salary: "+salary);//print current class salary
        System.out.println("Salary: "+super.salary);//print base class salary
    }
}
```

```

}
}
class Demo
{
    public static void main(String[] args)
    {
        HR obj=new HR();
        obj.display();
    }
}
```

# Super can be use to invoke immediate parent class constructor

```
class Person {  
  
    Person() {  
  
        System.out.println("Person class Constructor");  
  
    } }  
  
class Student extends Person {  
  
    Student(){  
  
        super();  
  
        System.out.println("Student class Constructor");  
  
    } }
```

```
class Stu extends Student {  
  
    Stu()  
  
    {  
  
        System.out.println("Stu class  
Constructor");  
  
    } }  
  
class GFG1 {  
  
    public static void main(String[] args)  
  
        {  
  
            Stu s = new Stu();  
  
        } }
```

# Super can be use to invoke immediate parent class method

```
class Person{  
    void message(){  
        System.out.println("This is person class");    }  
    }  
}  
  
class Student extends Person  
{  
    void message()  
    {  
        System.out.println("This is student class");  
    }  
}
```

```
void display()  
    {  
        message();  
        super.message();  
    }  
}  
  
class Test  
{  
    public static void main(String  
args[])  
    {  
        Student s = new Student();  
  
        // calling display() of Student  
        s.display();  
    }  
}
```



## Super can be use to invoke immediate parent class method

```
class A {  
    void msg() {  
        System.out.println("A is called here");  
    }  
}  
  
class B extends A {  
    void msg() {  
        super.msg();  
  
        System.out.println("B is called here");  
    }  
}
```

```
class Main extends B {  
    void msg() {  
        super.msg();  
  
        System.out.println("C is called here");  
    }  
  
    public static void main(String args[]) {  
        Main cc = new Main();  
  
        cc.msg();    }  
}
```

## This Keyword

- This is a reference variable that refers to the current object.
- Used to refer current class instance variable.
- Used to invoke current class method( implicitly ).
- Can be passed as an argument in the method call.
- Can be passed as argument in the constructor call.
- Can be used to return the current class instance from the method.

## To refer current class instance variable

```
class Student{  
    int rollno;  
    String name;  
    float fee;  
  
    Student(int rollno,String name,float fee){  
        this.rollno=rollno;  
        this.name=name;  
        this.fee=fee;  
    }  
}
```

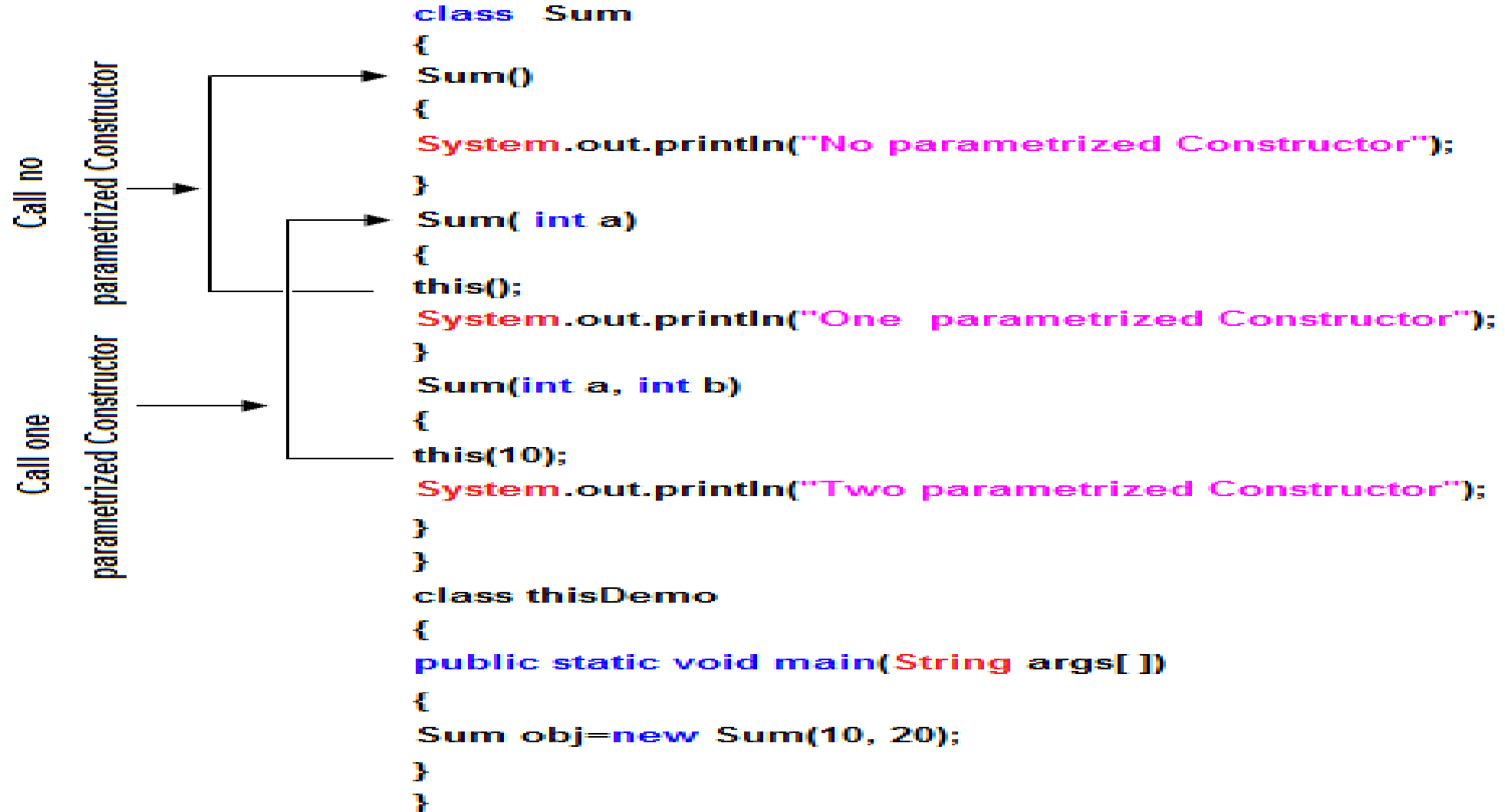
```
void display(){System.out.println(rollno+" "+  
name+" "+fee); }  
  
}  
  
class TestThis2{  
  
    public static void main(String args[]){  
        Student s1=new Student(111,"ankit",5000f);  
        Student s2=new Student(112,"sumit",6000f);  
        s1.display();  
        s2.display();  
    }  
}
```

## To invoke current class method

```
class A{  
  
void m(){System.out.println("hello m");}  
  
void n(){  
System.out.println("hello n");  
  
this.m();  
  
}  
  
}
```

```
class TestThis4{  
  
public static void main(String args[]){  
A a=new A();  
a.n();  
  
}}
```

# One constructor can call only other single constructor at a time by using this().



# Difference between This and Super

Sr. No.	Key	this	super
1	Represent and Reference	this keyword mainly represents the current instance of a class.	On other hand super keyword represents the current instance of a parent class.
2	Interaction with class constructor	this keyword used to call default constructor of the same class.	super keyword used to call default constructor of the parent class.
3	Method accessibility	this keyword used to access methods of the current class as it has reference of current class.	One can access the method of parent class with the help of super keyword.
4	Static context	this keyword can be referred from static context i.e can be invoked from static instance. For instance we can write <code>System.out.println(this.x)</code> which will print value of x without any compilation or runtime error.	On other hand super keyword can't be referred from static context i.e can't be invoked from static instance. For instance we cannot write <code>System.out.println(super.x)</code> this will leads to compile time error.

# Difference between This and Super

```
class Person{  
    int id;  
    String name;  
    Person(int id,String name){  
        this.id=id;  
        this.name=name;  
    } }  
  
class Emp extends Person{  
    float salary;
```

```
    Emp(int id,String name,float salary){  
        super(id,name);//reusing parent constructor  
        this.salary=salary; }  
    void display(){ System.out.println(id+" "+name+" "+salary);} }  
class TestSuper5{  
    public static void main(String[] args){  
        Emp e1=new Emp(1,"ankit",45000f);  
        e1.display(); } }
```

# Garbage Collection in Java

- In JAVA, programmer is responsible for both creation and destruction of objects.
- But in Java, the programmer need not to care for all those objects which are no longer in use. Garbage collector destroys these objects.

## **Objet eligible for garbage collection:**

- 1. By nulling a reference**
- 2. By assigning a reference to another**
- 3. By anonymous object**



# Garbage Collection in Java

## 1. By nulling a reference:

```
Employee e=new Employee();  
e=null;
```

## 2. By assigning a reference to another:

```
Employee e1=new Employee();  
Employee e2=new Employee();  
e1=e2;           //now the first object referred by e1 is available for garbage collection
```

## 3. By anonymous object:

```
new Employee();
```

# Finalization

- Finalize() is the method of Object class. This method is called just before an object is garbage collected. finalize() method overrides to dispose system resources, perform clean-up activities and minimize memory leaks.
- Just before destroying an object, Garbage Collector calls *finalize()* method on the object to perform cleanup activities.
- Based on our requirement, we can override finalize() method for perform our cleanup activities like closing connection from database.
- finalize() is not public because it shouldn't be invoked by anyone other than the JVM. However, it must be protected so that it can be overridden by subclasses who need to define behavior for it

## finalize()

```
public class Example1 {  
  
    public static void main(String[] args)  
    {  
  
        Example1 obj = new Example1();  
  
        System.out.println(obj.hashCode());  
  
        obj = null;  
  
        // calling garbage collector  
  
        System.gc();  
    }  
}
```

```
System.out.println("end of garbage collection");  
}  
  
protected void finalize()  
{  
  
    System.out.println("finalize method called");  
  
} }
```

# Polymorphism



In Shopping malls behave like Customer

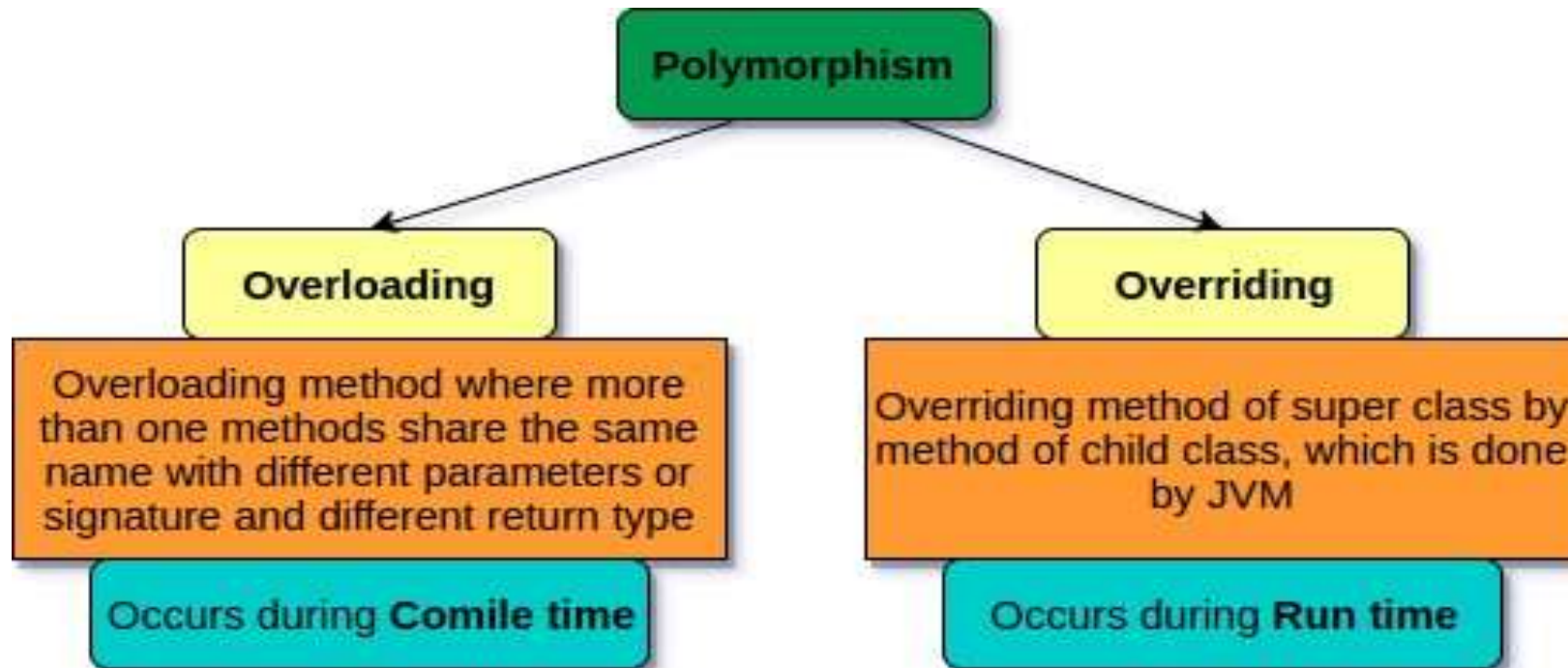
In Bus behave like Passenger

In School behave like Student

At Home behave like Son

# Types of Polymorphism

- Static or Compile time polymorphism
- Dynamic or Runtime polymorphism (Dynamic Binding)



# Overloading in Java

- Overloading allows different methods to have the same name, but different signatures where the signature can differ by the number of input parameters or type of input parameters or both.
- Overloading is related to compile-time (or static) polymorphism.

# Overloading in Java

```
public class Sum {  
  
    public int sum(int x, int y) {  
  
        return (x + y);  
  
    }  
  
    public int sum(int x, int y, int z) {  
  
        return (x + y + z);  
  
    }  
  
    public double sum(double x, double y) {
```

```
        return (x + y);  
    }  
  
    public static void main(String args[]) {  
  
        Sum s = new Sum();  
  
        System.out.println(s.sum(10, 20));  
        System.out.println(s.sum(10, 20, 30));  
        System.out.println(s.sum(10.5, 20.5));  
  
    }  
  
}
```

# Overriding in Java

- A method can only be written in the child class and not in same class.
- Argument list should be the same as that of the overridden method of that class.
- Instance methods can also be overridden if they are inherited by the child class.
- A constructor cannot be overridden.
- Any method that is static cannot be used to override.
- The return type must have to be the same, or a subtype of the return type declared in the original overridden method in the parent class.
- If a method cannot be inherited, then it cannot be overridden.



# Overriding in Java

- A child class within the same package as the instance's parent class can override any parent class method that is not declared private or final.
- A child class in a different package can only override the non-final methods declared as public or protected.

# Overriding in Java

```
class ABC{  
  
public void disp() {  
  
System.out.println("disp() method of parent class");  
  
}    }  
  
class Demo extends ABC{  
  
public void disp(){  
  
System.out.println("disp() method of Child class");  
  
}
```

# Overriding in Java

```
public void newMethod(){
```

```
    System.out.println("new method of child class"); }
```

```
public static void main( String args[]) {
```

**/\* When Parent class reference refers to the parent class object then in this case overridden method (the method of parent /\*class) is called. \*/**

```
ABC obj = new ABC( );
```

# Overriding in Java

```
obj.disp();
```

**/\* When parent class reference refers to the child class object then the overriding method (method of child class) is called.**

**/\* This is called dynamic method dispatch and runtime polymorphism \*/**

```
ABC obj2 = new Demo();
```

```
obj2.disp();
```

```
}
```

```
}
```

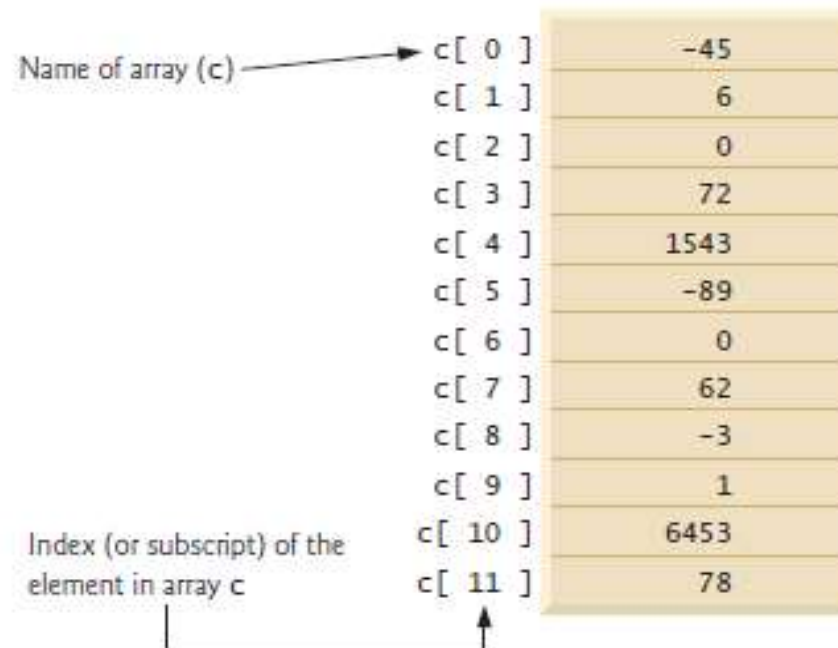
# Overriding in Java

- In the above example the call to the disp() method using second object (obj2) is runtime polymorphism (or dynamic method dispatch).
- **Note:** In dynamic method dispatch the object can call the overriding methods of child class and all the non-overridden methods of base class but it cannot call the methods which are newly declared in the child class.
- In the above example the object obj2 is calling the disp(). However if you try to call the newMethod() method (which has been newly declared in Demo class) using obj2 then you would give compilation error with the following message:

Exception in thread "main" java.lang.Error: Unresolved compilation

# Array

- An array is a group of variables (called **elements or components**) containing values **that all** have the same type.
- Arrays are objects, so they're considered reference types.



The diagram illustrates an array structure. On the left, the text "Name of array (c)" has an arrow pointing to the first element's index "c[ 0 ]". Below this, the text "Index (or subscript) of the element in array c" has an arrow pointing to the index "11" of the last element "c[ 11 ]". The array elements are listed in a table with their corresponding values.

c[ 0 ]	-45
c[ 1 ]	6
c[ 2 ]	0
c[ 3 ]	72
c[ 4 ]	1543
c[ 5 ]	-89
c[ 6 ]	0
c[ 7 ]	62
c[ 8 ]	-3
c[ 9 ]	1
c[ 10 ]	6453
c[ 11 ]	78

# Declaration/Defining Array

## 1. Declare an integer array variable

```
int[] primes;
```

```
int []primes;
```

```
int  primes[] ;
```

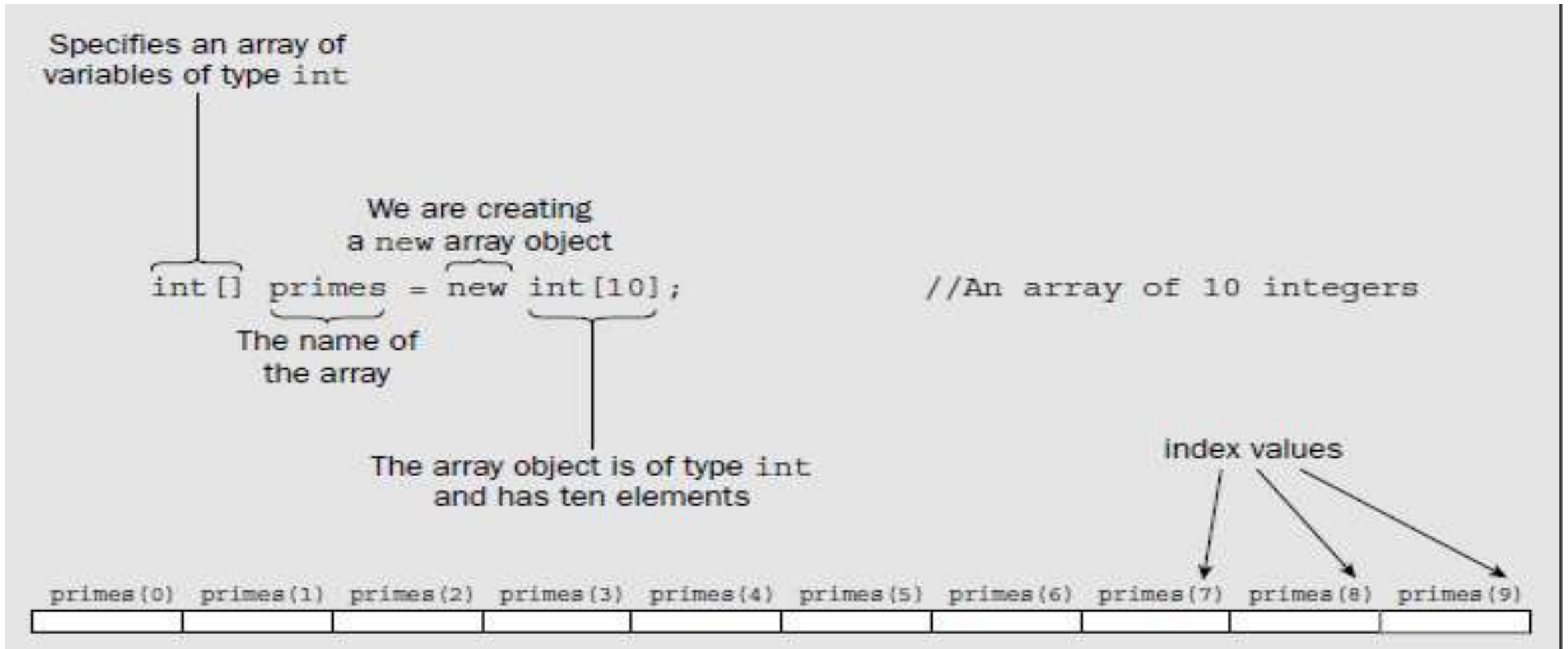
Note: At the time of declaration size is not required other wise we will get CTE

## 2. Defining an Array/Array Creation

Every array in java is an object only. Hence we can create array by using **new operator**

```
primes = new int[10]; // Define an array of 10 integers
```

# You can declare and create Array in single line





# Array in Java

```
class GFG
{
    public static void main (String[] args)
    {
        int[] arr;
        // allocating memory for 5 integers.
        arr = new int[5];
        arr[0] = 10;
        arr[1] = 20;
        arr[2] = 30;
        arr[3] = 40;
        arr[4] = 50;
```

```
        for (int i = 0; i < arr.length; i++)
        {
            System.out.println("Element at
index " + i + " : " + arr[i]);
        }
    }
```

# Types of Array in java

There are two types of array.

- Single Dimensional Array
- Multidimensional Array

# Single Dimensional Array in Java

## Syntax to Declare an Array in Java

- `dataType[] arr; (or)`
- `dataType []arr; (or)`
- `dataType arr[];`

# Multidimensional Array in Java

In such case, data is stored in row and column based index (also known as matrix form).

- `dataType[][] arrayRefVar; (or)`
- `dataType [][]arrayRefVar; (or)`
- `dataType arrayRefVar[][]; (or)`
- `dataType []arrayRefVar[];`

## Example

```
int[][] arr=new int[3][3];//3 row and 3 column
```

# Multidimensional Array in Java

**//Java Program to illustrate the use of multidimensional array**

**class Testarray3{**

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

**int arr[][]={{ 1,2,3},{2,4,5},{4,4,5}};**

**//printing 2D array**

**for(int i=0;i<3;i++){**

**for(int j=0;j<3;j++){**

**System.out.print(arr[i][j]+" ");**

**}**

**System.out.println();**

**}**

**}}**

# Java Lambda Expressions

- It provides a clear and concise way to represent one method interface using an expression.
- A lambda expression is a short block of code which takes in parameters and returns a value.
- Lambda expression provides implementation of *functional interface*. An interface which has only one abstract method is called functional interface.

# Java Lambda Expressions Syntax

(argument-list) -> {body}

Java lambda expression is consisted of three components.

- 1) **Argument-list:** It can be empty or non-empty as well.
- 2) **Arrow-token:** It is used to link arguments-list and body of expression.
- 3) **Body:** It contains expressions and statements for lambda expression.

# Java Lambda Expressions Syntax

## 1) No Parameter Syntax

() -> {

//Body of no parameter lambda

}

## 2) One Parameter Syntax

(p1) -> {

//Body of single parameter lambda

}

## 3) Two Parameter Syntax

(p1,p2) -> {

//Body of multiple parameter

lambda

}



# Without Lambda Expression

```
interface Drawable{  
  
    public void draw( );  
  
}  
  
public class LambdaExpressionExample {  
  
    public static void main(String[] args) {  
  
        int width=10;  
  
        //without lambda, Drawable implementation using  
        anonymous class
```

```
        Drawable d=new Drawable( ){  
  
            public void draw(){  
  
                System.out.println("Drawing "+width);  
  
            }  
  
            };  
  
            d.draw();  
  
        }  
  
    }
```

# Java Lambda Expression Example

```
interface Drawable{  
    public void draw();  
}  
  
public class LambdaExpressionExample2 {  
    public static void main(String[] args) {  
        int width=10;  
        //with lambda  
        Drawable d2=( )->{  
            System.out.println("Drawing "+width);  
        };  
    }  
}
```

```
d2.draw();  
    }  
}
```

# Java Lambda Expression Example: No Parameter

```
interface Sayable{  
    public String say();  
}  
  
public class LambdaExpressionExample3{  
    public static void main(String[] args) {  
        Sayable s=()->{  
            return "I have nothing to say.";  
        };  
        System.out.println(s.say());  
    }  
}
```

# Java Lambda Expression Example: Single Parameter

```
interface Sayable{  
    public String say(String name);  
}  
  
public class LambdaExpressionExample4{  
    public static void main(String[] args) {  
  
        // Lambda expression with single parameter.  
        Sayable s1=(name)->{  
            return "Hello, "+name;  
        };  
    }  
}
```

```
System.out.println(s1.say("Sonoo"));  
  
    // You can omit function parentheses  
  
    Sayable s2= name ->{  
        return "Hello, "+name;  
    };  
    System.out.println(s2.say("Sonoo"));  
  
    }  
}
```

# Justify array in java is object

**Consider following code**

```
class Test
{
    psvm(String [] args)
    {
        int []x=new int[3]
        s.o.p(x.getClass().getName());
    }
}
```

**For every array type corresponding classes are available and these classes are part of java language and not available to the programmer level**



## Print Array In Java

```
Public class ArrayPrintInJava
{
Public static void main (string args[])
{
String [] country = new string [4];
Country[0] = "India";
Country[1] = "Nepal";
Country[2] = "Bhutan";
Country[3] = "Japan";
For (string str : country )
{
Sysytem.out.println(str); }
}
}
```

### Output

India
Nepal
Bhutan
Japan

1. What is false about constructor?[CO2]
  - a) Constructors cannot be synchronized in Java
  - b) Java does not provide default copy constructor
  - c) Constructor can have a return type**
  - d) “this” and “super” can be used in a constructor
2. Abstract class cannot have a constructor.[CO2]
  - a) True
  - b) False**
3. What is not the use of “this” keyword in Java?[CO2]
  - a) Passing itself to another method
  - b) Calling another constructor in constructor chaining
  - c) Referring to the instance variable when local variable has the same name
  - d) Passing itself to method of the same class**

4. What would be behaviour if the constructor has a return type?[CO2]
- a) **Compilation error**
  - b) Runtime error
  - c) Compilation and runs successfully
  - d) Only String return type is allowed
5. What is the syntax of abstract class in java?[CO2]
- a. abstract A{ }
  - b.abstract class A
  - c abstract class A{ }**
  - d. abstract class A[ ]
6. A method which is declared as abstract and does not have implementation is known as an \_\_\_\_\_?[CO2]
- A. Abstract Interface
  - B. Abstract Thread
  - C. Abstract List
  - D. abstract Method**



## Weekly Assignment

- Q1. Explain Classes and objects in java with examples.[CO2]
- Q2. Explain Object Creation in java with suitable example.[CO2]
- Q3. Explain Constructor with examples.[CO2]
- Q4. Explain the concept of Abstract Classes with program.[CO2]
- Q5. Explain Interface, how interface can be used to achieve multiple inheritance?[CO2]
- Q6. Write the similarities and differences between abstract classes and interfaces.[CO2]
- Q7. Differentiate between Constructor and Method.[CO2]
- Q8. Explain Garbage Collection in Java.[CO2]
- Q9. Explain This keyword with suitable example.[CO2]
- Q10. Explain Super keyword with example.[CO2]

- <https://www.youtube.com/watch?v=ZHLdVRXIuC8&list=PLS1QulWo1RIbfTjQvTdj8Y6yyq4R7g-Al&index=19>
- <https://www.youtube.com/watch?v=2aQ9Y7bumts>
- <https://www.youtube.com/watch?v=XQ5NRKg8lXI>
- <https://www.youtube.com/watch?v=jg4MpYr1TBc>
- <https://www.youtube.com/watch?v=nixQyPIAnOQ>
- <https://www.youtube.com/watch?v=5X0Y--92pMI>

# MCQs

1. Employee emp = \_\_\_\_ Employee (); Pick a suitable word from the list so that an object of the class Employee is created.[CO2]

- A) object
- B) class
- C) run
- D) new**

2. \_\_\_\_ is a Java run-time system that chooses to execute the JAVA Bytecode:[CO2]

- A) SDK
- B) JDK
- C) JVM**
- D) None of the above

3. Super keyword in java is used to [CO2]

- a) Refer immediate parent class instance variables.
- b) Invoke immediate parent class methods.
- c) Invoke immediate parent class constructor.
- d) All**

4. Which of these keywords are used to define an abstract class? [CO2]

- a) abst
- b) **abstract**
- c) Abstract
- d) abstract class

5. Which of these can be overloaded?[CO2]

- a) Methods
- b) Constructors
- c) All of the mentioned**
- d) None of the mentioned

6. What would be behaviour if the constructor has a return type?[CO2]

- a) **Compilation error**
- b) Runtime error
- c) Compilation and runs successfully
- d) Only String return type is allowed

7. What is the syntax of abstract class in java?[CO2]

- a. abstract A{}
- b. abstract class A
- c abstract class A{}**
- d. abstract class A[]

8. A method which is declared as abstract and does not have implementation is known as an \_\_\_\_\_?[CO2]

- A. Abstract Interface
- B. Abstract Thread
- C. Abstract List
- D. abstract Method**

9. Which of this keyword can be used in a subclass to call the constructor of superclass?[CO2]

- a. super**
- b.this
- c. extent
- d. Extends

10. Which of these is correct way of calling a constructor having no parameters, of superclass A by subclass B?[CO2]

- a. super(void);
- b. superclass.();
- c. super.A();
- d. super();**

11. What is false about constructor?[CO2]

- a) Constructors cannot be synchronized in Java
- b) Java does not provide default copy constructor
- c) Constructor can have a return type**
- d) “this” and “super” can be used in a constructor

12. Abstract class cannot have a constructor.[CO2]

- a) True
- b) False**

13. What is not the use of “this” keyword in Java?[CO2]

- a) Passing itself to another method
- b) Calling another constructor in constructor chaining
- c) Referring to the instance variable when local variable has the same name
- d) Passing itself to method of the same class**

# Glossary Questions

1. Attempt all the parts. Pick the correct option from glossary. [CO2]

i) Constructor                      ii) Super                      iii) Base                      iv) Data members

- a) \_\_\_\_\_ is called first, automatically, whenever an object is created.
- b) \_\_\_\_\_ is the term used to indicate the variable and constants of a class.
- c) A base class is also known as \_\_\_\_\_ class.
- d) An abstract class is always a \_\_\_\_\_ class.

2. Attempt all the parts. Pick the correct option from glossary. [CO 2]

i) super                      ii) constructor                      iii) Method overloading                      iv) Polymorphism

- a) \_\_\_\_\_ supports method overriding in Java.
- b) \_\_\_\_\_ name must be same as the class name.
- c) \_\_\_\_ keyword is a reference variable that is used to refer to the immediate parent class object.
- d) \_\_\_\_\_ increases the readability of the program.

# Sessional Paper-1

Printed page: 2

Subject Code: ACSE0302

Roll No:

**NOIDA INSTITUTE OF ENGINEERING AND TECHNOLOGY, GREATER NOIDA**

(An Autonomous Institute Affiliated to AKTU, Lucknow)

B. Tech (AI / AIML / IOT / DS)

(SEM:III SESSIONAL EXAMINATION – I)(2021-2022)

Subject Name: OBJECT ORIENTED TECHNIQUES USING JAVA

Time: 1.15 Hours

Max. Marks:30

**General Instructions:**

- > All questions are compulsory. Answers should be brief and to the point.
- > This Question paper consists of 2 pages & 5 questions.
- > It comprises of three Sections, A, B, and C. You are to attempt all the sections.
- > **Section A** - Question No- 1 is objective type questions carrying 1 mark each, Question No- 2 is very short answer type carrying 2 mark each. You are expected to answer them as directed.
- > **Section B** - Question No-3 is short answer type questions carrying 5 marks each. You need to attempt any two out of three questions given.
- > **Section C** - Question No. 4 & 5 are Long answer type (within unit choice) questions carrying 6marks each. You need to attempt any one part a or b.
- > Students are instructed to cross the blank sheets before handing over the answer sheet to the invigilator.
- > No sheet should be left blank. Any written material after a blank sheet will not be evaluated/checked.

		<b>SECTION – A</b>	<b>[8]</b>	
<b>1.</b>	<b>Attempt all parts.</b>		<b>(4×1=4)</b>	<b>CO</b>
	a.	The type of Arguments the main method accepts is ____ a) integer[ ] b) String c) float[ ] d) String[ ]	<b>(1)</b>	<b>CO2</b>
	b.	The default value for data field of a boolean type, numeric type is ____ , ____ respectively. a) false, 1 b) true, 0 c) false, 0 d) true, 1	<b>(1)</b>	<b>CO1</b>
	c.	Using _____, we can force immediate termination of a loop. a) break b) continue c) return d) goto	<b>(1)</b>	<b>CO1</b>
	d.	The _____ statement is used to explicitly return from a method. a) break b) continue c) return d) goto	<b>(1)</b>	<b>CO2</b>
<b>2.</b>	<b>Attempt all parts.</b>		<b>(2×2=4)</b>	<b>CO</b>
	a.	Write a JAVA program to check whether the number entered by user is even or odd by using if-else. (Use the Scanner class to enter the integer	<b>(2)</b>	<b>CO1</b>



		number)		
	b.	What is the output of the following Java code snippet? <pre>int i=0; for(i=1; i &lt;= 6; i++) {     if ( i % 3 == 0 )         continue;     System.out.print (i+","); }</pre>	(2)	CO1
<b>SECTION – B</b>				
3.	Answer any <u>two</u> of the following-		[2×5=10]	CO
	a.	Draw a class diagram of Student class and explain how the access modifiers are represented in the class diagrams?	(5)	CO1
	b.	Discuss different levels of access specifiers available in JAVA.	(5)	CO1
	c.	What is the purpose of constructors? Explain all the types of constructors in JAVA with the help of example of your choice.	(5)	CO2
<b>SECTION – C</b>				
4	Answer any <u>one</u> of the following-(Any one can be applicative if applicable)		[2×6=12]	CO
	a.	Explain the four pillars of Object-Oriented Programming with the help of examples.	(6)	CO1
	b.	Write a JAVA program to display all even numbers from 100 to 50 using all the loops statements you have studied.	(6)	CO1
5.	Answer any <u>one</u> of the following-			
	a.	Write a program in JAVA that takes arguments name, department and marks of 4 subjects from the user and then print total and average marks obtained. (Use command line arguments for giving input).	(6)	CO1
	b.	Write a JAVA program to display the reverse of an input number. (Use Scanner class to input the positive integer)	(6)	CO1

# Sessional Paper-2

Printed page: 2

Subject Code: ACSE0302

Roll No:

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

**NOIDA INSTITUTE OF ENGINEERING AND TECHNOLOGY, GREATER NOIDA**  
**(An Autonomous Institute)**

**Affiliated to Dr. A.P. J. Abdul Kalam Technical University, Uttar Pradesh, Lucknow**

**Course ...B.Tech.....Branch...IT....**

**Semester.....III...Sessional Examination.....II.....Year- (2021 - 2022)**

**Subject Name: OBJECTS ORIENTED TECHNIQUES USING JAVA**

**Time: 1.15Hours**

**[ SET- A]**

**Max. Marks:30**

---

**General Instructions:**

- This Question paper consists of 2 pages & 5 questions. It comprises of three Sections, A, B, and C
- Section A - Question No- 1 is objective type questions carrying 1 mark each, Question No- 2 is very short answer type carrying 2 mark each. You are expected to answer them as directed.
- Section B - Question No-3 is Short answer type questions carrying 5 marks each. Attempt any two out of three questions given.
- Section C - Question No. 4 & 5 are Long answer type (within unit choice) questions carrying 6 marks each. Attempt any one part a or b.

		<u>SECTION – A</u>	[08Marks]	
<b>1.</b>		<b>All questions are compulsory</b>	<b>(4×1=4)</b>	
	<b>a.</b>	<b>Identify the correct way to call the constructor of class “Super” that is inherited by the class “Child”.</b> (i) class Child extends Super{ Child(){ Super();}} (ii) class Child extends Super{ Child(){ super();}} (iii) class Child extends Super{ Child() { super.Super();}} (iv) All the ways are correct.	<b>(1)</b>	<b>CO2</b>
	<b>b.</b>	<b>Total abstraction can be achieved using?</b> (i) abstract class (ii) interface (iii) both (iv) total abstraction cannot be achieved	<b>(1)</b>	<b>CO2</b>
	<b>c.</b>	<b>The ..... class inherits all the properties of the ..... class?</b> (i) base, derived (ii) derived base (iii) base, initial (iv) base, final	<b>(1)</b>	<b>CO2</b>
	<b>d.</b>	<b>Runtime polymorphism is also known as:</b> (i) Dynamic binding (ii) Static binding (iii) Early binding (iv) None of the above	<b>(1)</b>	<b>CO2</b>
<b>2.</b>		<b>All questions are compulsory</b>	<b>(2×2=4)</b>	
	<b>a.</b>	<b>Explain a simple program showing garbage collection.</b>	<b>(2)</b>	<b>CO2</b>
	<b>b.</b>	<b>Explain the concept of interface using suitable example.</b>	<b>(2)</b>	<b>CO2</b>

<b>SECTION – B</b>			<b>[10Marks]</b>	
<b>3.</b>	<b>Answer any <u>two</u> of the following-</b>		<b>(2×5=10)</b>	
	<b>a.</b>	<b>Explain the types of polymorphism in java using suitable examples for each type.</b>	<b>(5)</b>	<b>CO2</b>
	<b>b.</b>	<b>Explain the difference between interface and abstract class in java using suitable example.</b>	<b>(5)</b>	<b>CO2</b>
	<b>c.</b>	<b>Explain inheritance in java. Explain all types of inheritance supported by java using suitable examples.</b>	<b>(5)</b>	<b>CO2</b>
<b>SECTION – C</b>			<b>[12Marks]</b>	
<b>4</b>	<b>Answer any <u>one</u> of the following-</b>		<b>(1×6=6)</b>	
	<b>a.</b>	<b>Explain the working of “this” and “super” keyword in java. Illustrate each using suitable example.</b>	<b>(6)</b>	<b>CO2</b>
	<b>b.</b>	<b>Explain abstract class. State the use of abstract class with suitable example. Also write the names of OOPs concepts that is used to implement abstract class and that is implemented using abstract class.</b>	<b>(6)</b>	<b>CO2</b>
<b>5.</b>	<b>Answer any <u>one</u> of the following-</b>		<b>(1×6=6)</b>	
	<b>a.</b>	<b>Explain the concept of access modifiers (public, private, protected) using packages.</b>	<b>(6)</b>	<b>CO3</b>
	<b>b.</b>	<b>Explain overloading and overriding of methods? Illustrate overloading and overriding of methods in Java with suitable examples.</b>	<b>(6)</b>	<b>CO2</b>

**NOTE: No example should be repeated.**

# Old University Question Paper

Printed Page:-

Subject Code:- ACSE0302

Roll. No:

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

NOIDA INSTITUTE OF ENGINEERING AND TECHNOLOGY, GREATER NOIDA

(An Autonomous Institute Affiliated to AKTU, Lucknow)

B.Tech.

SEM: III - THEORY EXAMINATION (2021 - 2022)

Subject: Object Oriented Techniques using Java

Time: 03:00 Hours

Max. Marks: 100

## General Instructions:

1. All questions are compulsory. It comprises of three Sections A, B and C.
  - Section A - Question No- 1 is objective type question carrying 1 mark each & Question No- 2 is very short type questions carrying 2 marks each.
  - Section B - Question No- 3 is Long answer type - I questions carrying 6 marks each.
  - Section C - Question No- 4 to 8 are Long answer type - II questions carrying 10 marks each.
  - No sheet should be left blank. Any written material after a Blank sheet will not be evaluated/checked



# Old University Question Paper

## SECTION A

20

1. Attempt all parts:-

1-a. In JAVA main method returns value of type \_\_\_\_\_ [CO1]

1

1. float
2. int
3. void
4. String

1-b. What is bytecode in Java? [CO1]

1

1. Code generated by a Java compiler
2. Code generated by a Java Virtual Machine
3. Name of Java source code file
4. Block of code written inside a class

1-c. If same message is passed to objects of several different classes and all of those can respond in a different way, what is this feature called? [CO2]

1

1. Inheritance
2. Overloading
3. Polymorphism
4. Overriding

1-d. Java \_\_\_\_\_ is invoked at the time of object creation. [CO2]

1

1. constructor
2. class
3. method

4. array

1-e. Which of these keywords must be used to monitor for exceptions? [CO3]

1

1. try
2. catch
3. throw
4. finally

1-f. An \_\_\_\_\_ statement can be used to access the classes and interface of a different package from the current package. [CO3]

1

1. instanceof
2. import
3. extends
4. implement

# Old University Question Paper

1-e. Which of these keywords must be used to monitor for exceptions? [CO3] 1

1. try
2. catch
3. throw
4. finally

1-f. An \_\_\_\_\_ statement can be used to access the classes and interface of a different package from the current package. [CO3] 1

1. instanceof
2. import
3. extends
4. implement

1-g. The keyword that is used to protect the methods from simultaneous access in Threads is \_\_\_\_\_ [CO4] 1

1. save
2. synchronized
3. Both
4. This task is not possible in threads

1-h. Which of these classes are used by Byte streams for input and output operation? [CO4] 1

1. Input Stream
2. InputStream
3. Reader
4. All of the mentioned

1-i. A \_\_\_\_\_ dictates the style of arranging the components in a container. [CO5] 1

1. border layout
2. grid layout
3. panel
4. layout manager

1-j. \_\_\_\_\_ interface provides the capability to store objects using a key-value pair. [CO5] 1

1. Java.util.Map
2. Java.util.Set
3. Java.util.List
4. Java.util.Collection

# Old University Question Paper

2. Attempt all parts:-

- |       |                                                                                    |   |
|-------|------------------------------------------------------------------------------------|---|
| 2-a.  | What is JVM? [CO1]                                                                 | 2 |
| 2-b.  | What is the use of final keyword in JAVA? [CO2]                                    | 2 |
| 2-c.  | What is an assertion in Java? How is it different from if - else conditions? [CO3] | 2 |
| <hr/> |                                                                                    |   |
| 2-d.  | Describe any two Annotations from the Java Standard Library. [CO4]                 | 2 |
| 2-e.  | What Are Wrapper Classes? Why do we need wrapper classes in JAVA? [CO5]            | 2 |



# Old University Question Paper

## SECTION B

30

3. Answer any five of the following:-

- |      |                                                                                                                                          |   |
|------|------------------------------------------------------------------------------------------------------------------------------------------|---|
| 3-a. | What is the difference between object diagrams and class diagrams? Draw a class diagram of order management system. [CO1]                | 6 |
| 3-b. | How to take an input from a user with the help of scanner class in JAVA? Explain using JAVA code. [CO1]                                  | 6 |
| 3-c. | Explain Abstract class concept with an example program. [CO2]                                                                            | 6 |
| 3-d. | Compare overloading and overriding of methods in java using proper examples. [CO2]                                                       | 6 |
| 3-e. | Write a method to check if input string is Palindrome? [CO3]                                                                             | 6 |
| 3-f. | Explain the concept of multithreading in java and explain how even and odd numbers can be printed by using multithreading concept. (CO4) | 6 |
| 3-g. | Examine ArrayList with Example. [CO5]                                                                                                    | 6 |

# Old University Question Paper

## SECTION C

50

4. Answer any one of the following:-

- 4-a. What are command line arguments? How are they useful? Write a program to compute the sum of the digits of an input number (Using command line arguments) eg if 4523 is an integer then the sum of digits displayed will be 14. [CO1] 10
- 4-b. Write a JAVA program that takes values of name, age, department and marks of 4 subjects from the user. Display the name, total and average of marks computed. [CO1] 10

5. Answer any one of the following:-

- 5 Explain the following with respect to JAVA: [CO2] 10
- a) super keyword
  - b) Garbage collection
  - c) Interface
  - d) Static data members
  - e) final keyword
- 5 What is the lambda expression in Java and what are the features of a lambda expression? Briefly explain its use with the help of suitable example. [CO2] 10



6. Answer any one of the following:-

6 Write the differences between String, StringBuffer and StringBuilder classes. With proper syntax, explain the following methods. [CO3] 10

1. Method to extract a particular character of a string.
2. Reverse a String.

6 What is the difference between an error and exception? Write the following Java program for illustrating the use of throw keyword. Write a class ThrowExample contains a method checkEligibility(int age, int weight) which throw an 10

---

ArithmeticException with a message "Student is not eligible for registration" when age < 12 and weight < 40, otherwise it prints "Student Entry is Valid!!". [CO3]

7. Answer any one of the following:-

7-a. What is the difference between thread and a process? Explain the concept of Inter Thread Communication and describe the role of wait(), notify(), and notifyAll() methods in inter thread communication. [CO4] 10

# Old University Question Paper

- 7-b. While reading a file, how would you check whether you have reached to the end of file? Write a JAVA program to copy the content of "file1.txt" to "file2.txt". [CO4] 10
8. Answer any one of the following:-
- 8-a. Discuss some general rules for using layout managers. Describe the various layout managers available in AWT. [CO5] 10
- 8-b. Differentiate between List and ArrayList. Create a class TestArrayList having main method. Perform following functionality. [CO5] 10
1. Create an ArrayList having fruits name of type String.
  2. Store different fruit names. (Try to add duplicate fruit names).
  3. Print all fruit names.
  4. Print the first and last fruit names.
  5. Print the size of ArrayList.
  6. Remove a particular fruit from ArrayList.

## Expected Questions

1. Can we override the private methods?
2. What is the purpose of a default constructor?
3. What is the use of super keyword in java?
4. What are objects? How are they created?
5. Which concept allows you to reuse the written code?
6. Which function in Java program is necessary for running the program?
7. Which type of inheritance results in the diamond problem?

# Old Question Papers

- [https://www.iare.ac.in/sites/default/files/IARE\\_JAVA\\_MODEL\\_QP.pdf](https://www.iare.ac.in/sites/default/files/IARE_JAVA_MODEL_QP.pdf)
- <https://www.manareresults.co.in/jntuh/download.php?subcode=133BM>

## Recap of Unit

- A class which is declared with the abstract keyword is known as an abstract class in Java.
- Inheritance in java is a mechanism in which one object acquires all the properties and behaviors of parent object.
- **Constructor in java** is a special type of method that is used to initialize the object.
- Polymorphism in Java is **the ability of an object to take many forms**.
- Everything in Java is associated with classes and objects, along with its attributes and methods.

## Text Books:

**(1) Herbert Schildt," Java - The Complete Reference", McGraw Hill Education 12<sup>th</sup> edition**

**(2) Herbert Schildt," Java: A Beginner's Guide", McGraw-Hill Education 2<sup>nd</sup> edition**

**(3) James Rumbaugh et. al, "Object Oriented Modeling and Design", PHI 2<sup>nd</sup> Edition**

## Reference Books:

**(4) Cay S. Horstmann, "Core Java Volume I – Fundamentals", Prentice Hall**

**(5) Joshua Bloch," Effective Java", Addison Wesley**

**(6) E Balagurusamy, "Programming with Java A Primer", TMH, 4th edition.**



# Thank You