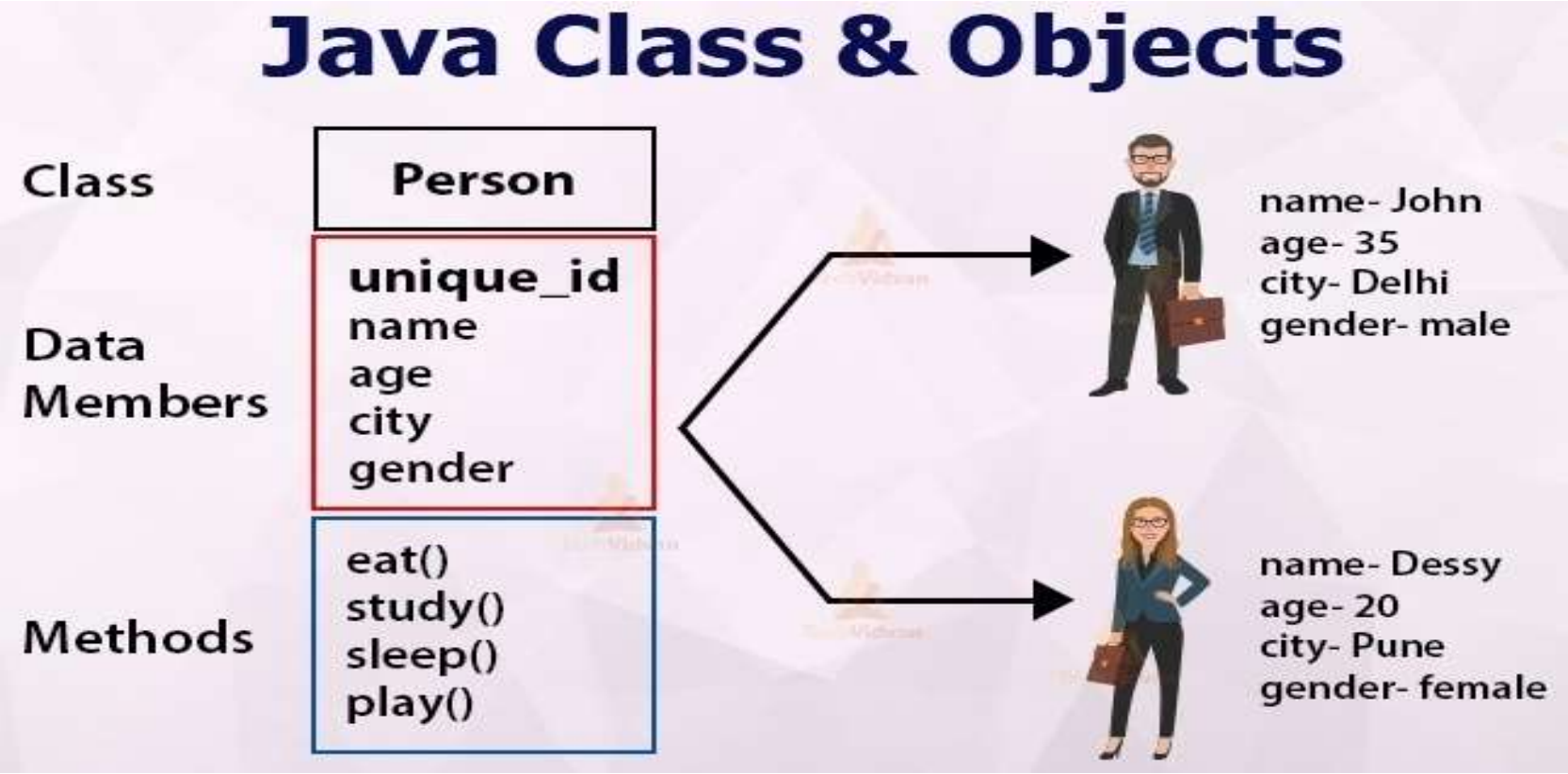
Java Classes/Objects

- 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 menthod
{
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

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 salar
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();
}}
```

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 System.out.println(this.x) 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 System.out.println(super.x) 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();  }  }
```

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

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);
   }
}
```

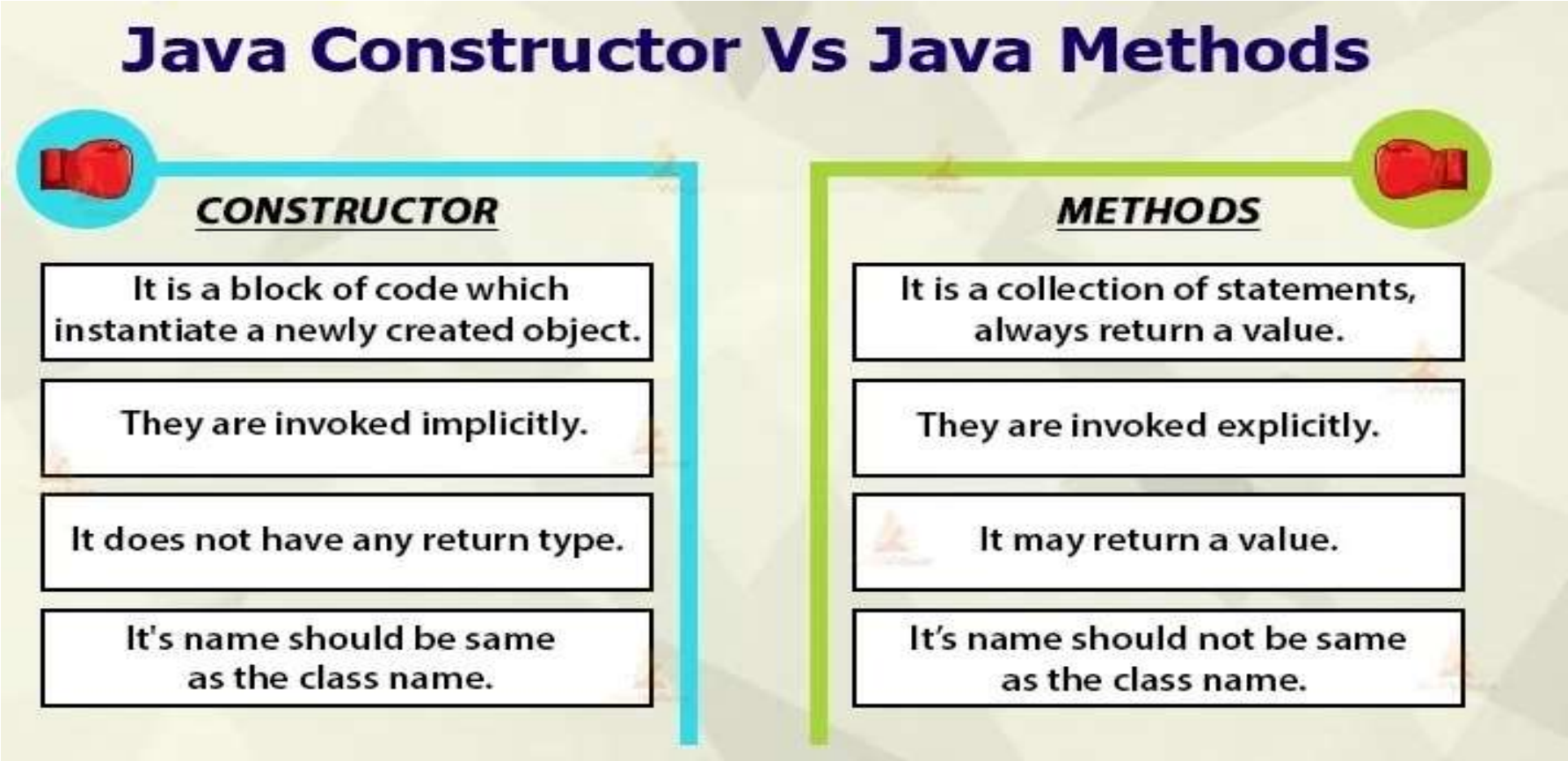
Output:

```
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 there 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



Java Access Modifier

It is used to restrict the scope of a class, constructor, variable, method, or data member. There are four types of access modifiers available in java:

1.Default – No keyword required

2.Private

3.Protected

4.Public

Java Access Modifier

Default: No access modifier is specified for a class, method, or data member.

- The data members, class or methods which are accessible **only within the same package**.

Private: The methods or data members declared as private are accessible only **within the class** in which they are declared. Any other **class of** the **same package will not be able to access** these members.

Protected: Accessible within the same package or subclasses in different packages.

Public : Classes, methods, or data members that are declared as public are **accessible from everywhere** in the program. There is no restriction on the scope of public data members.

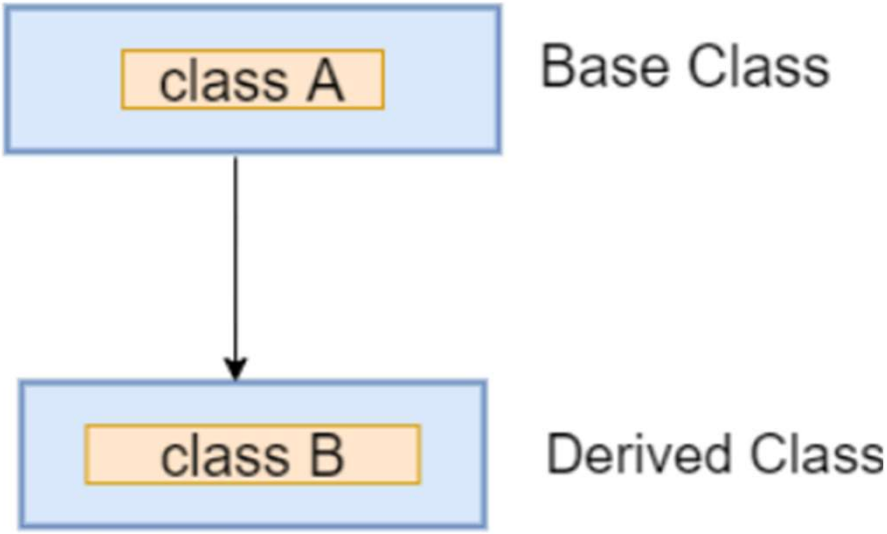
Java Access Modifier

Java Access Modifiers Table

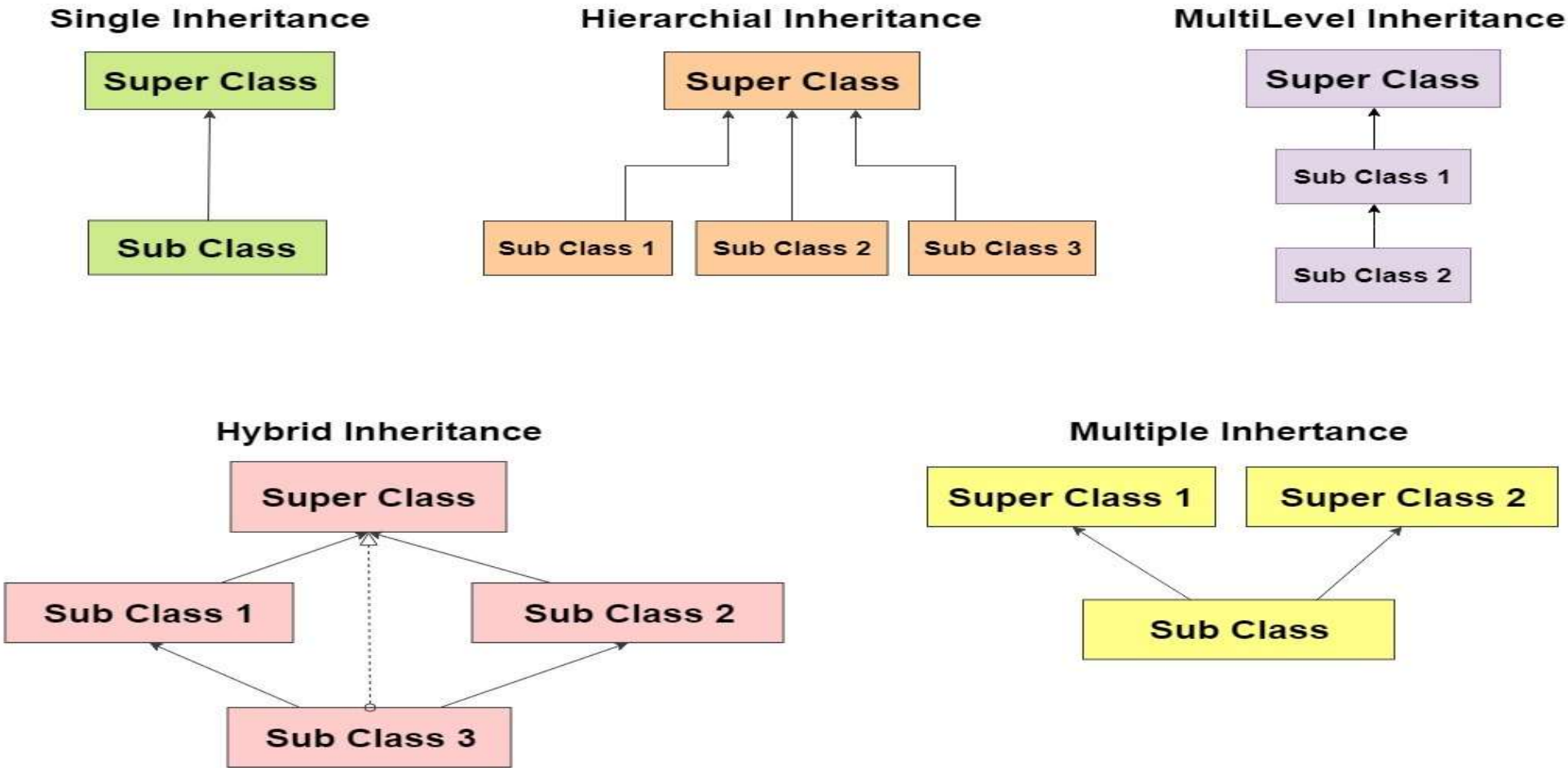
	private	default	protected	public
Same Package – NonSubclass	NO	YES	YES	YES
Same Package – Subclass	NO	YES	YES	YES
Diff Package – Nonsubclass	NO	NO	NO	YES
Diff Package - Subclass	NO	NO	YES	YES

Inheritance

- Inheritance in java is a mechanism in which one object acquires all the properties and behaviors of parent object.
- It shows the parent child relationship between two classes.
- Parent class : Super class : Base class
- Child class: Sub class : Derived class



Types of Inheritance



Types of Inheritance

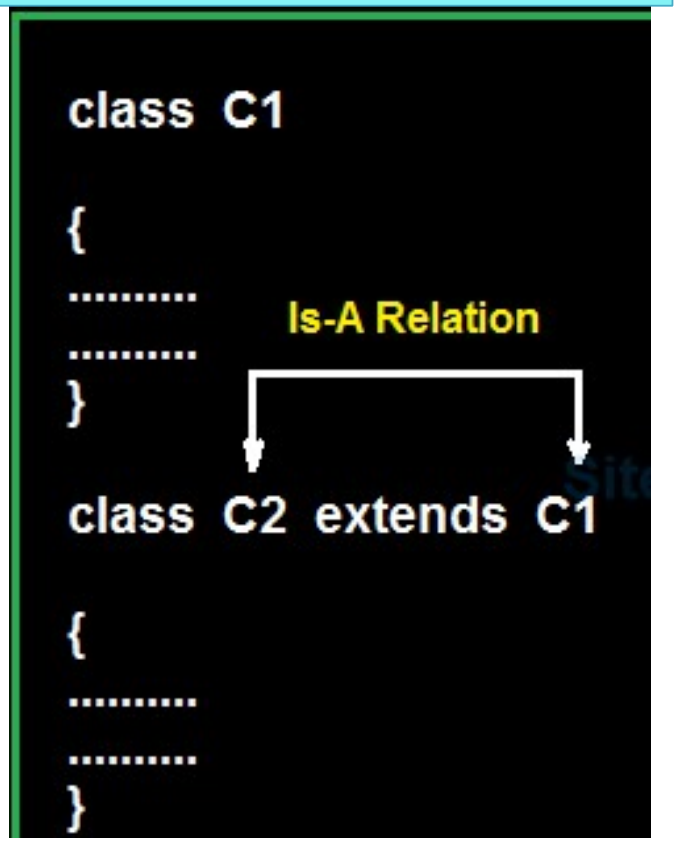
Single Inheritance
Class A → Class B
	public class A {
	.....
	}
	public class B **extends** A {
	.....
	}

Multi Level Inheritance Class A → Class B → Class C	public class A { .....} public class B **extends** A {.....} public class C **extends** B {..... }
Hierarchical Inheritance Class A ← Class B, Class A ← Class C	public class A { .....} public class B **extends** A {.....} public class C **extends** A {..... }

Multiple Inheritance
Class A, Class B → Class C
	public class A { .....}
	public class B {.....}
	public class C **extends** A,B {
	.....
	} // Java does not support mutiple Inheritance

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();
}
}
```



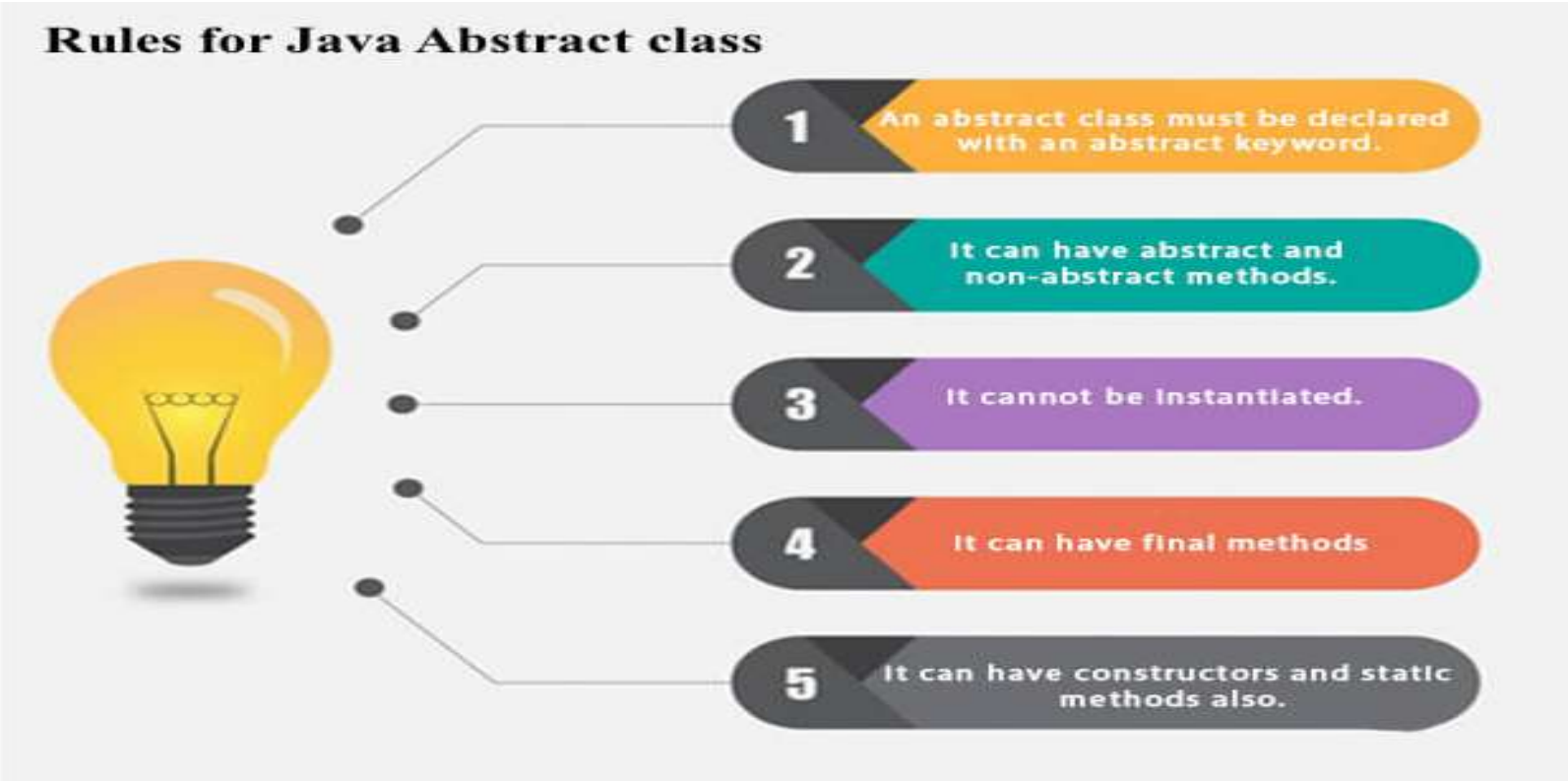
Multilevel Inheritance

```
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();
}}
```

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 clas**s 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



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 {

   // declare constant fields
   // declare methods that abstract  by default.
}
```

Static keyword

- It is mainly used for memory management.
- It is used to share the same variable or method of a given class.
- The static keyword belongs to the class than an instance of the class.

In Java it is applicable for the following:

1. Blocks
2. Variables
3. Methods
4. Classes

Static keyword

When a member is declared static, it can be accessed before any objects of its class are created, and without reference to any object(e.g. obj.var).

```
class Test{
    static void m1() {
    System.out.println("from m1");
   }
   public static void main(String[] args){
            m1();
   }
}
```

Output

From m1

Static keyword in Block

Static blocks

If you need to do the computation in order to initialize your **static variables**, you can declare a static block that gets executed exactly once, when the class is first loaded.

```
class Test
{   static int a = 10;
    static {
      System.out.println("Static block initialized.");
      a = a * 4;    }
    public static void main(String[] args){
        System.out.println("from main");
        System.out.println("Value of a : "+a);}}
```

Output

```
Static
block initliaized
From main
Vlaue of a : 40
```

Static keyword in variable

Static variables

When a variable is declared as static, then a single copy of the variable is created and shared among all objects at the class level. Static variables are, essentially, global variables. All instances of the class share the same static variable.

Important points for static variables:

•We can create static variables at the class level only.

•static block and static variables are executed in the order they are present in a program.

Static keyword in variable

```
class Test
{       static int a = m1();
        static {
        System.out.println("Inside static
block");
      }
        static int m1() {
        System.out.println("from m1");
        return 20;
    }
      public static void main(String[] args)
    {
      System.out.println("Value of a : "+a);
      System.out.println("from main");
    }
}
```

Output

```
from m1
Inside static block
Value of a : 20
from main
```

Static keyword in methods

Static methods

When a method is declared with the *static* keyword, it is known as the static method. The most common example of a static method is the *main( )* method. Methods declared as **static have several restrictions:**

•They can only directly call other static methods.

•They can only directly access static data.

•They cannot refer to this or super in any way.

Static keyword in methods

```
class Calculate{
        static int cube(int x){
        return x*x*x;
        }
   public static void main(String args[]){
   int result=Calculate.cube(5);
   System.out.println(result);
   }
}
```

Output
125

Static keyword in Class

Static Class

- A class can be made **static** only if it is a nested class.
- We cannot declare a top-level class with a static modifier but can declare nested classes as static.
- Such types of classes are called Nested static classes.
- Nested static class doesn't need a reference of Outer class.
- A static class cannot access non-static members of the Outer class.

Static keyword in Class

```
class Outer {
     static String str = "Hi Students";
                int x=10; //non static
      static class Nested{

                public void disp(){
                        System.out.println(str);
                         //System.out.println(x); //error
       }
   }
   public static void main(String args[])
   {
       Outer.Nested obj  = new Outer.Nested();
                        obj.disp();
   }
}
```

Output
Hi Students

Final keyword

It is applicable only to a variable, a method, or a class.

Final Variable	→	To Create constant variable
Final Methods	→	Prevent Mehod Overriding
Final Classes	→	Prevent Inheritance

Final keyword in class

Final classes

When a class is declared with *final* keyword, it is called a final class. A final class cannot be extended(inherited).

There are two uses of a final class:

Usage 1: One is definitely to prevent inheritance

Usage 2: To create an immutable class like the predefined String class. We can not make a class immutable without making it final.

```
final class A {       // methods and fields  }
class B extends A {    // COMPILE-ERROR! Can't  subclass  }
```

Final keyword in Methods

Final Methods

- A final method cannot be overridden.
- It is used when we required to follow the same implementation throughout all the derived classes.

```
class A
{
    final void m1()
    {
        System.out.println("This is a
final method.");
    }
}
class B extends A
{
    void m1()
    {
        // Compile-error! We can not override
        System.out.println("Illegal!");
    }
}
```

Final keyword in variables

Final Variables

- It's value can't be modified, essentially, it behave like a constant.
- final variable must be initialized otherwise, the compiler will throw error.
- A blank final variable(uninitialized final variable) can be initialized inside an constructor. If you have more than one constructor in a class then it must be initialized in all of them, otherwise, a compile-time error will be thrown. It is good practice to represent final variables in all **uppercase**, using underscore to separate words.
- A blank final static variable can be initialized inside a static block.

Final keyword in variables

```
class Test_Final {
   final int THRESHOLD = 5;
   final int CAPACITY;
   static final double PI = 3.1415;
   static final double INTEREST_RATE;

   {
      CAPACITY = 25;
   }
  static{
        INTEREST_RATE = 12.3;
   }
```

Final keyword in variables

```
Class Test {

   // Main driver method
   public static void main(String args[])
   {
      // Declaring local final variable
      final int i;

      // Now initializing it with integer value
      i = 20;

      // Printing the value on console
      System.out.println(i);
   }
}
```

Output

20

Interface

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.

Defining and Implements Interface

Defining an Interface

```
access interface name {
    return-type method-name1(parameter-list);
    return-type method-name2(parameter-list);
    type final-varname1 = value;
    type final-varname2 = value;
    // ...
    return-type method-nameN(parameter-list);
    type final-varnameN = value;
}
```

Implementing Interfaces

```
access class classname [extends superclass]
                [implements interface [,interface...]] {
    // class-body
}
```

Extending Interface

Interfaces Can Be Extended

One interface can inherit another by use of the keyword extends. The syntax is the same as for inheriting classes.

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();
  }
}
```

Output:
Hello

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();   }  }
```

Output:

Animal is eating
Animal is travelling

Difference between Interface and Abstract class

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

8)**Example:**
public abstract class Shape{
public abstract void draw();
}
	Example:
	public interface Drawable{
	void draw();
	}

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 Example {
    public static void main(String[] args)
   {
      Example obj = new Example();
      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");
   }  }
```

Output:

314265080
end of garbage collection
finalize method called

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

Polymorphism

In Shopping malls behave like Customer

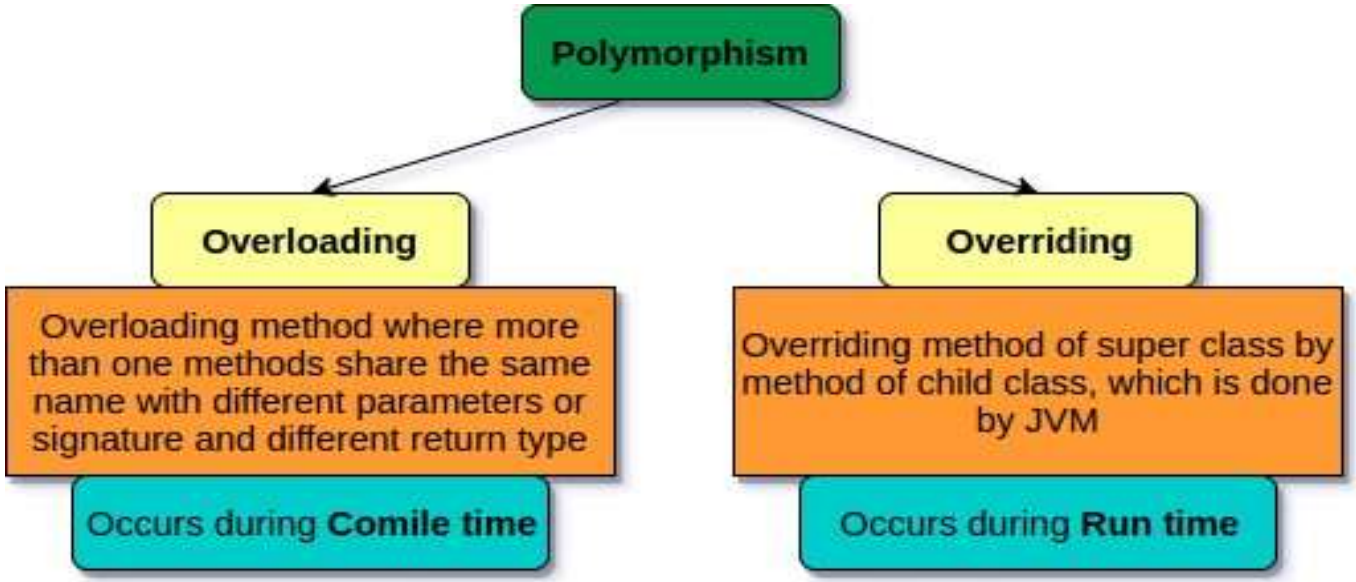
In Bus behave like Passenger

In School behave like Student

At Home behave like Son

Types of Polymorphism

- Static or Compile time polymorphism (Static / Early Binding)
- Dynamic or Runtime polymorphism (Dynamic/Late 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));
}}
```

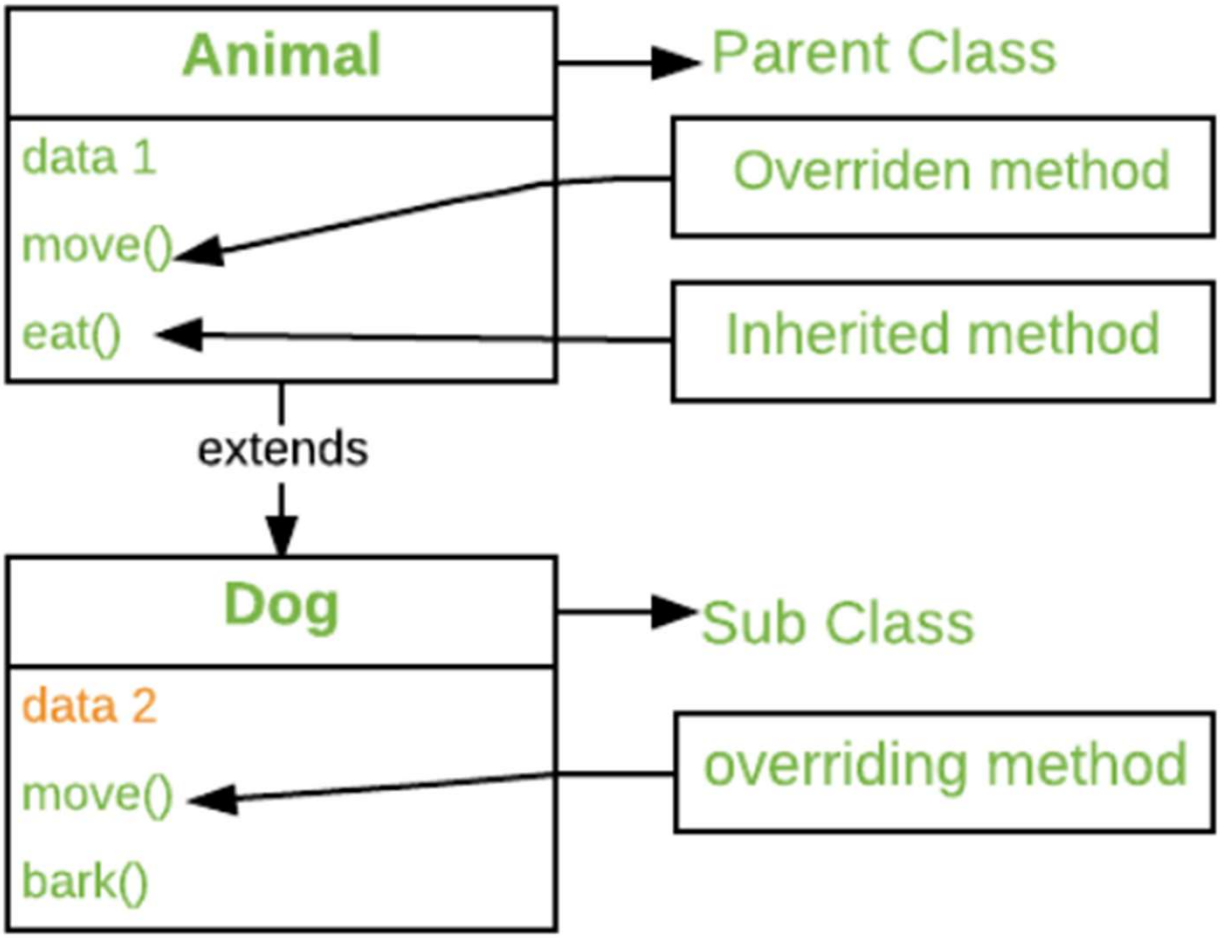
Output:

30
60
31.00

Overriding in Java

- A method(overridden method) written in a class and method with same name (overriding method) , same signature and same return type is re- written its derived class than during method invocation binding of method by the reference of parent class is resolved at runtime so it is called runtime binding.
- Argument list should be the same as that of the overridden method of that class.
- A constructor cannot be overridden.
- Any method that is static cannot be used to override.

Overriding in Java



Overriding in Java

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

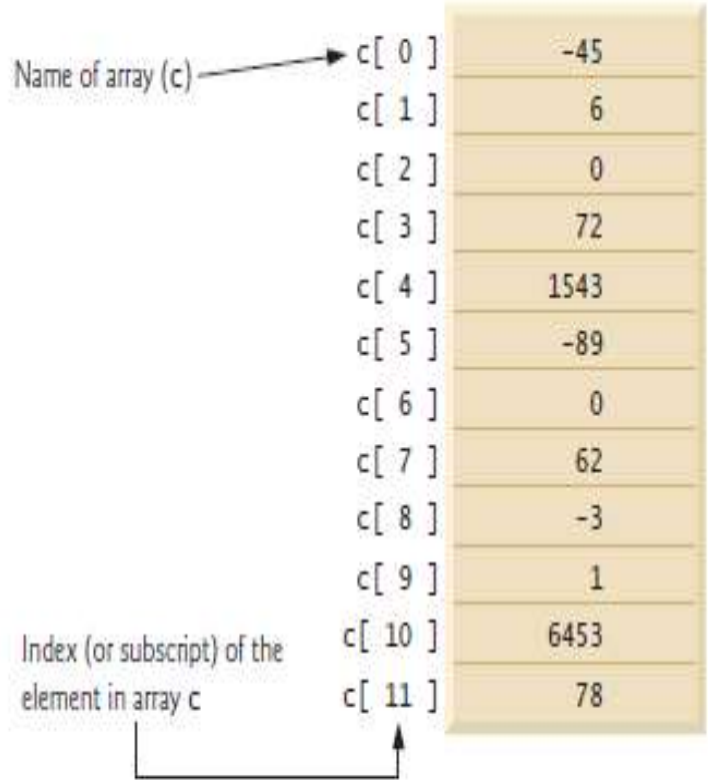
```
public static void main( String args[]) {
        Parent obj2 = new Child();//upcast
        obj2.disp();
   }
}
```

Output:

disp() method of Child class

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.



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

Specifies an array of variables of type `int`

We are creating a `new` array object

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

The name of the array

The array object is of type `int` and has ten elements

index values

primes[0]	primes[1]	primes[2]	primes[3]	primes[4]	primes[5]	primes[6]	primes[7]	primes[8]	primes[9]

Array in Java

```
class MyArray
{
        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]);
        }}
```

Output:

Element at index 0: 10
Element at index 1: 20
Element at index 2: 30
Element at index 3: 40
Element at index 4: 50

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
```

Types of Array in Java

There are two types of array.

- **Single Dimensional Array**
- **Multidimensional Array**

Declare an 1D Array

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

Multidimensional Array

-- 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();
}
}}
```

Output:

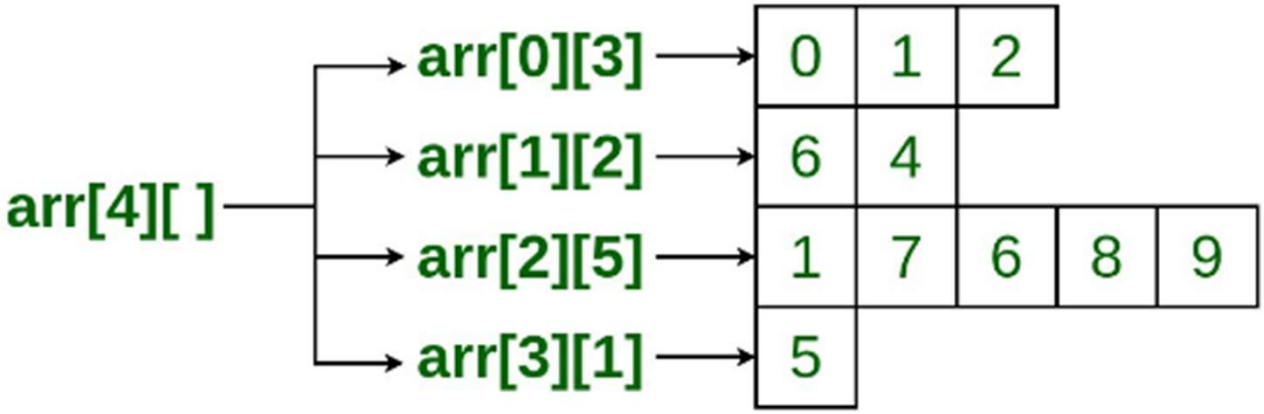
1,2,3

2,4,5

4,4,5

Jacked Array

A jagged array is an array of arrays such that member arrays can be of different sizes, i.e., we can create a 2-D array but with a variable number of columns in each row. These types of arrays are also known as Jagged arrays.



Jacked Array

```
class JackedArray
{
  public static void main(String[] a)
  {
   int x[][]={{2,4},
              {33,11,66,22},
              {10,30,90}};
        //or
  //int x[][]={new int[]{2,4},
             //  new int[]{33,11,66,22},
              // new int[]{10,30,90}};
for(int i=0;i<x.length;i++){
      for(int j=0;j<x[i].length;j++){
         System.out.print(x[i][j]+"  ");
         }
     System.out.println();
    }
  }
}
```

Output:

```
2  4
33  11  66  22
10  30  90
```

Java Lambda Expressions

- It is a new and important feature of Java which was included in Java SE 8.
- It provides a clear and concise way to represent **one method interface** using an expression
- It is a short block of code which takes in parameters and returns a value. Lambda expressions are similar to methods, but they do not need a name and they can be implemented right side in the body of a method.
- It is very useful in collection library. It helps to iterate, filter and extract data from collection.
- In case of lambda expression, we don't need to define the method again for providing the implementation. It saves a lot of code.

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 LambdaExp1 {
    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();
    }
}
```

Output:

Drawing 10

Java Lambda Expression Example

```
interface Drawable{
   public void draw();
}
 public class LambdaExp2{
   public static void main(String[] args) {
      int width=10;
       //with lambda
      Drawable d2=( )->{
         System.out.println("Drawing "+width);
      };
      d2.draw();
   }
}
```

Output:

Drawing 10

Java Lambda Expression Example: No Parameter

```
interface Sayable{
   public String say();
}
public class LambdaExp3{
public static void main(String[] args) {
   Sayable s=()->{
      return "I have nothing to say.";
   };
   System.out.println(s.say());
}
}
```

Output:

I have nothing to say

Java Lambda Expression Example: Single Parameter

```
interface Sayable{
   public String say(String name);
}

public class LambdaExp4{
   public static void main(String[] args) {

    // Lambda expression with single parameter.
      Sayable s1=(name)->{
         return "Hello, "+name;
      };
```

```
System.out.println(s1.say(" CSE-III "));

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

Output:

CSE-III
CSE-III

Daily Quiz

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

Daily Quiz

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]

Topic Links

- 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

MCQs

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

MCQs

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();

MCQs

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.

		SECTION – A	[8]	
1.	Attempt all parts.		(4×1=4)	CO
	a.	The type of Arguments the main method accepts is ____. a) integer[ ] b) String c) float[ ] d) String[ ]	(1)	CO2
	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	(1)	CO1
	c.	Using _____, we can force immediate termination of a loop. a) break b) continue c) return d) goto	(1)	CO1
	d.	The _____ statement is used to explicitly return from a method. a) break b) continue c) return d) goto	(1)	CO2
2.	Attempt all parts.		(2×2=4)	CO
	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	(2)	CO1

Page 1 of 2

Conti….

		number)		

	b.	What is the output of the following Java code snippet?
		int i=0;
		for(i=1; i <= 6; i++)
		{
		if (i % 3 = = 0)
		continue;
		System.out.print (i+",");
		}
			(2)	CO1

		SECTION – B		
3.	Answer any two 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
		SECTION – C		
4	Answer any one 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 one 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 &5are Long answer type (within unit choice) questions carrying 6marks each. Attempt any one part *a or b.*

Conti….

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

Conti..

		SECTION – B	**[10Marks]**	
3.	**Answer any two of the following-**		**(2×5=10)**	
	a.	**Explain the types of polymorphism in java using suitable examples for each type.**	**(5)**	**CO2**
	b.	**Explain the difference between interface and abstract class in java using suitable example.**	**(5)**	**CO2**
	c.	**Explain inheritance in java. Explain all types of inheritance supported by java using suitable examples.**	**(5)**	**CO2**
		SECTION – C	**[12Marks]**	
4	**Answer any one of the following-**		**(1×6=6)**	
	a.	**Explain the working of "this" and "super" keyword in java. Illustrate each using suitable example.**	**(6)**	**CO2**
	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.**	**(6)**	**CO2**
5.	**Answer any one of the following-**		**(1×6=6)**	
	a.	**Explain the concept of access modifiers (public, private, protected) using packages.**	**(6)**	**CO3**
	b.	**Explain overloading and overriding of methods? Illustrate overloading and overriding of methods in Java with suitable examples.**	**(6)**	**CO2**

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. InputOutputStream
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
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] a) super keyword b) Garbage collection c) Interface d) Static data members e) final keyword	10
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

Old University Question Paper

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] 1. Method to extract a particular character of a string. 2. Reverse a String.	10
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 checkEligibilty(int age, int weight) which throw an ArithmeticException with a message "Student is not eligible for registration" when age < 12 and weight < 40, otherwise it prints "Student Entry is Valid!!". [CO3]	10

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.manaresults.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.

References

Text Books:
(1) Herbert Schildt," Java - The Complete Reference", McGraw Hill Education 12th edition
(2) Herbert Schildt," Java: A Beginner's Guide", McGraw-Hill Education 2nd edition
(3) James Rumbaugh et. al, "Object Oriented Modeling and Design", PHI 2nd 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.