**Calling Same class constructors using this keyword**

|  |
| --- |
| We can’t call a constructor in another constructor by using it’s name, we have to use this keyword to call the same class constructor and we have to use super keyword to call the super class constructor. |

**In a constructor How many constructors we can call?**

|  |
| --- |
| 1. Only one constructor 2. And the calling statement must be the first statement. |

**Example on calling constructor by using this keyword**

class One

{

    int a,b;

    One()

    {  System.out.println("default constructor");

    }

    One(int a,int b)

    {   this();

        this.a=a;

        this.b=b;

        System.out.println("parameterized constructor");

    }

    public void display()  //we can call instance methods by using object

    {

        System.out.println("Object state...");

        System.out.println("a:\t"+a);

        System.out.println("b:\t"+b);

    }

}

public class ThisDemo2

{

    public static void main(String[] args)

    {

            One o1=new One(10,20);

            o1.display();

    }

}

Output:

default constructor

parameterized constructor

Object state...

a: 10

b: 20

**types of instance methods?**

|  |
| --- |
| **There are 2 types of instance methods available in Java**   1. Accessor methods 2. Mutator methods |

**What is object state?**

|  |
| --- |
| The data existed in an object is called as object state |

**Accessor method**

|  |
| --- |
| It access or gets the object state  Ex: in the above example display method is accessor method  //accessor methods      public void display()  //we can call instance methods by using object      {          System.out.println("Object state...");          System.out.println("a:\t"+a);          System.out.println("b:\t"+b);      } |

**Mutator method**

|  |
| --- |
| It is a method which modifies the object state |

**Example on accessor and mutator methods**

class Emp

{

    private int eno;

    private String ename;

    private float sal;

    //mutator method

    public void setEno(int eno)//e1

    {   //eno=101

        this.eno=eno;  //e1.eno=101;

    }

    //mutator method

    public void setEname(String ename)

    {   this.ename=ename;

    }

    //mutator method

    public void setSal(float sal)

    {   this.sal=sal;

    }

    //accessor method

    public int getEno()//e1

    {   return eno;  //return this.eno;

    }

    //accessor method

    public String getEname()

    {   return this.ename;

    }

    //accessor method

    public float getSal()

    {   return sal;

    }

}

public class AccessorMutator

{

    public static void main(String[] args)

    {   Emp e1=new Emp();

        Emp e2=new Emp();

        e1.setEno(101);

        e1.setEname("NagiReddy");

        e1.setSal(400000.00f);

        System.out.println("e1 Object state........");

        System.out.println(  e1.getEno() );

        System.out.println(  e1.getEname() );

        System.out.println(  e1.getSal() );

    }

}

Output:

e1 Object state........

101

NagiReddy

400000.0

**Inheritance**

|  |
| --- |
| The process of including fields and methods of one class into another class is called as inheritance. |

**There are different forms of inheritance available**

1. Single level /simple inheritance
2. Multiple inheritance

**Other forms**

1. Multilevel Inheritance (extension of single level inheritance)
2. Hierarchical Inheritance (extension to single level)
3. Hybrid inheritance (combination of 2 or more forms)

**Single Level Inheritance**

|  |
| --- |
| If a class inherits only one Base class at a time, then it is called as simple or single level inheritance. |

class Base

{   static int s1=100;

    int a,b;

    static{

        System.out.println("base class is loaded...");

    }

    Base()

    {   System.out.println("Base class constructor");

    }

    static void fun1()

    {System.out.println("Base static method..");

    }

    void display1()

    {   System.out.println("Base instance method..");

        System.out.println("a:\t"+a);

        System.out.println("b:\t"+b);

    }

}

class Child extends Base

{

    static int s1=1000;

    int c,d;

    Child()

    {   super();

        System.out.println("Child class constructor");

    }

    static{

        System.out.println("Child class is loaded...");

    }

    void display2()

    {

        System.out.println("Object state....");

        System.out.println("c:\t"+c);

        System.out.println("d:\t"+d);

    }

}

public class SimpleInher

{

    public static void main(String[] args)

    {

       Child c1=new Child();

       c1.display1();

       c1.display2();

    }

}

Output:

base class is loaded...

Child class is loaded...

Base class constructor

Child class constructor

Base instance method..

a: 0

b: 0

Object state....

c: 0

d: 0

**Another example**

class Base

{   static int s1=100;

    int a,b;

    static{

        System.out.println("base class is loaded...");

    }

    Base()

    {   System.out.println("Base class constructor");

    }

    static void fun1()

    {System.out.println("Base static method..");

    }

    void display1()

    {   System.out.println("Base instance method..");

        System.out.println("a:\t"+a);

        System.out.println("b:\t"+b);

    }

}

class Child extends Base

{

    static int s1=1000;

    int c,d;

    Child()

    {   this(10,20);

        System.out.println("Child class constructor");

    }

    Child(int c,int d)

    {   //super();

        this.c=c;

        this.d=d;

    }

    static{

        System.out.println("Child class is loaded...");

    }

    void display2()

    {

        System.out.println("Object state....");

        System.out.println("c:\t"+c);

        System.out.println("d:\t"+d);

    }

}

public class SimpleInher

{

    public static void main(String[] args)

    {

       Child c1=new Child(100,200);

       c1.display1();

       c1.display2();

       Child c2=new Child();

    }

}

output

base class is loaded...

Child class is loaded...

Base class constructor

Base instance method..

a: 0

b: 0

Object state....

c: 100

d: 200

Base class constructor

Child class constructor

**Method overloading**

**What is method overloading?**

|  |
| --- |
| **We can write more than one method with same name with different signature in a class. It is called as method overloading.** |

**Example on method overloading**

class MyMath

{

    //add(int,int)

    int add(int a,int b)

    {   return a+b;

    }

    //add(int,int,int)

    int add(int a,int b,int c)

    {   return a+b+c;

    }

    //add(int,float)

    float add(int a,float b)

    {   return a+b;

    }

    //add(float,int)

    float add(float a,int b)

    {   return a+b;

    }

    //add(float,float)

    float add(float a,float b)

    {   return a+b;

    }

    //add(String,int)

    String add(String name,int n)

    {   return name+n;

    }

}

public class MethodOverload

{

    public static void main(String[] args)

    {

        MyMath mm=new MyMath();

        System.out.println(mm.add("Madhu", 9));

        System.out.println(mm.add(10,20));

        System.out.println(mm.add(10,20,30));

        System.out.println(mm.add(10,20.50f));

        System.out.println(mm.add(10.50f,20.50f));

        System.out.println(mm.add(10.50f,20));

    }

}

Output:

Madhu9

30

60

30.5

31.0

30.5

**What is polymorphism?**

|  |
| --- |
| 1. It has the ability to take more than one form 2. If we implement method overloading or method overriding it is coming under polymorphism. |

**What is Method Overriding?**

|  |
| --- |
| We can write a method in child class which is already existed in base class. It is called as method overriding.  Rules to follow at the time of overriding a method   1. Access modifier must be same or higher scope 2. Return type same 3. Method name same 4. Parameters count, type and order must be same 5. If base class method throws an exception, Child method may or may not throw an exception |

**What is the default access modifier which is applied to the members of a class?**

|  |
| --- |
| Package private |

**Access modifiers**

|  |
| --- |
| 1. Private (same class) 2. Package private (within the package) 3. Protected (same package and child or sub child classes of other packages) 4. Public (in any package) |

**What is super?**

|  |
| --- |
| 1. Super is a keyword which represents super class object 2. After overriding an instance method, we can call base class instance method in child class by using super keyword 3. By using super keyword, we can call the base class constructor in child constructor. |

**Method overriding example**

class Base

{   static int s1=100;

    int a,b;

    static{

        System.out.println("base class is loaded...");

    }

    Base(int a,int b)

    {

        System.out.println("Base class constructor");

        this.a=a;

        this.b=b;

    }

    static void fun1()

    {System.out.println("Base static method..");

    }

    void display()

    {   System.out.println("Base instance method..");

        System.out.println("a:\t"+a);

        System.out.println("b:\t"+b);

    }

}

class Child extends Base

{

    static int s1=1000;

    int c,d;

    Child(int a,int b,int c,int d)

    {   super(a,b);

        this.c=c;

        this.d=d;

    }

    static{

        System.out.println("Child class is loaded...");

    }

    public void display()

    {

        super.display();

        System.out.println("Object state....");

        System.out.println("c:\t"+c);

        System.out.println("d:\t"+d);

    }

}

public class SimpleInher

{

    public static void main(String[] args)

    {

        Child c1=new Child(100,200,300,400);

        c1.display();

    }

}

Output:

base class is loaded...

Child class is loaded...

Base class constructor

Base instance method.

a: 100

b: 200

Object state....

c: 300

d: 400

**Upcasting & Down Casting**

**What is upcasting?**

|  |
| --- |
| 1. Process of assigning Child class object to base type reference variable is called as upcasting. 2. During performing upcasting no need to perform explicit typecasting. |

**Down casting?**

|  |
| --- |
| 1. Process of assigning Base type reference variable (which contains Child class object) to Child type reference variable is called as down casting. 2. During down casting, you have to perform explicit type casting |

**Small example on Upcasting & Downcasting**

class Base{

}

class Child extends Base{

}

public class UpCastDownCast

{

    public static void main(String[] args)

    {

         //Upcasting

         Base b=new Child();

         //Downcasting

         Child c=(Child)b;

         //Child c=(Child)new Base(); //you will get a ClassCastException during execution

    }

}

**Static & Dynamic Binding**

**What is static binding?**

|  |
| --- |
| 1. Static binding is also called as compile time binding or early binding 2. At the time of compilation java compiler decides which members of a class has to invoke or execute at runtime. |

**What is dynamic binding?**

|  |
| --- |
| 1. At the time of program execution JVM decides which method has to invoke at runtime. 2. Only instance methods support dynamic binding if it is overridden   Note: only when you implement upcasting you can see dynamic binding. |

**Example on static and dynamic binding**

class Base

{

    static int s=100;

    int n=200;

    static void fun1()

    {   System.out.println("Base class static method...");

    }

    void display()

    {System.out.println("Base class instance method");

    }

}

class Child extends Base

{

    static int s=1000;

    int n=2000;

    static void fun1()

    {   System.out.println("Child class static method...");

    }

    void display()

    {System.out.println("Child class instance method");

    }

}

public class UpCastDownCast

{

    public static void main(String[] args)

    {

        //upcasting

        Base b=new Child();    //correct

        System.out.println( b.s ); //compile time binding

        System.out.println(b.n);  //

        b.fun1();

        b.display();

        //if it is instance method(yes)

        //is it overriden in the Child class or not(yes)

        //dynamic binding (means the methods from Child class will be invoked)

        //it was decided during program execution by JVM.

    }

}

Output:

100

200

Base class static method...

Child class instance method

**Compile time binding example**

class Base

{

    static int s=100;

    int n=200;

    static void fun1()

    {   System.out.println("Base class static method...");

    }

    void display()

    {System.out.println("Base class instance method");

    }

}

class Child extends Base

{

    static int s=1000;

    int n=2000;

    static void fun1()

    {   System.out.println("Child class static method...");

    }

    void display()

    {System.out.println("Child class instance method");

    }

}

public class UpCastDownCast

{

    public static void main(String[] args)

    {

        Child c=new Child();

        System.out.println(c.s);

        System.out.println(c.n);

        c.fun1();

        c.display();

    }

}

Output:

1000

2000

Child class static method...

Child class instance method

**Compile time polymorphism**

|  |
| --- |
| If compile time binding is performed during implementation of polymorphism, then it is called as compile time polymorphism. |

**Example on compile time polymorphism**

class MyMath

{

    //add(int,int)

    int add(int a,int b)

    {   return a+b;

    }

    //add(int,int,int)

    int add(int a,int b,int c)

    {   return a+b+c;

    }

    //add(int,float)

    float add(int a,float b)

    {   return a+b;

    }

    //add(float,int)

    float add(float a,int b)

    {   return a+b;

    }

    //add(float,float)

    float add(float a,float b)

    {   return a+b;

    }

    //add(String,int)

    String add(String name,int n)

    {   return name+n;

    }

}

public class MethodOverload

{

    public static void main(String[] args)

    {

        MyMath mm=new MyMath();

        System.out.println(mm.add("Madhu", 9));

        System.out.println(mm.add(10,20));

        System.out.println(mm.add(10,20,30));

        System.out.println(mm.add(10,20.50f));

        System.out.println(mm.add(10.50f,20.50f));

        System.out.println(mm.add(10.50f,20));

    }

}

**What is dynamic polymorphism**

|  |
| --- |
| At the time implementing polymorphism if dynamic binding is performed then it is called as dynamic polymorphism. |

**Example**

class Base

{

    static int s=100;

    int n=200;

    static void fun1()

    {   System.out.println("Base class static method...");

    }

    void display()

    {System.out.println("Base class instance method");

    }

}

class Child extends Base

{

    static int s=1000;

    int n=2000;

    static void fun1()

    {   System.out.println("Child class static method...");

    }

    void display()

    {System.out.println("Child class instance method");

    }

}

public class UpCastDownCast

{

    public static void main(String[] args)

    {

        //upcasting

        Base b=new Child();    //correct

        b.display();

        //if it is instance method(yes)

        //is it overriden in the Child class or not(yes)

        //dynamic binding (means the methods from Child class will be invoked)

        //it was decided during program execution

    }

}

Output:

Child class instance method

**Abstract classes**

**What is an abstract class?**

|  |
| --- |
| 1. It is a class which contains zero or more abstract methods and some implemented methods. 2. We can’t create objects for abstract classes 3. We can inherit an Abstract classes but the child class has to implement(override) all the abstract methods of abstract class(base class).   Note: if a class contains an abstract method you have to declare that class as abstract  Abstract class Example:  abstract class One  {      abstract int add(int a,int b);      int sub(int a,int b)      {return a-b;      }  } |

**What is an abstract method?**

|  |
| --- |
| 1. It is a non-implemented method, which contains only method header not body. 2. It should be declared by using a keyword called abstract   Ex: abstract int add(int a,int b); |

**What is a concrete method?**

|  |
| --- |
| It is an implemented method |

**What is a concrete class?**

|  |
| --- |
| It contains all implemented methods and we can create object for it. |

**Example on abstract class**

abstract class One

{

    abstract int add(int a,int b);

    int sub(int a,int b)

    {return a-b;

    }

}

class MyMath extends One

{

    int add(int a,int b)

    {return a+b;

    }

}

public class AbstractDemo

{

    public static void main(String[] args)

    {

        One o=new MyMath();

        System.out.println(   o.add(10,2)  );

        System.out.println(  o.sub(10,2)  );

    }

}

Output:

12

8

**Another example**

abstract class One

{

    abstract int add(int a,int b);

}

class MyMath extends One

{

    int add(int a,int b)

    {return a+b;

    }

    int sub(int a,int b)

    {return a-b;

    }

}

public class AbstractDemo

{

    public static void main(String[] args)

    {

        One o=new MyMath();  //upcasting

        System.out.println(   o.add(10,2)  );

        MyMath mm=(MyMath)o;//downcasting

        System.out.println(  mm.sub(10,2)  );

    }

}

Output:

12

8

**Concepts we have covered till now**

* + 1. **Introduction**
    2. **Features**
    3. **Jdk installations**
    4. **VSCode installation**
    5. **JCL**
    6. **Java API**
    7. **Escape sequences**
    8. **Naming conventions**
    9. **Compilation**
    10. **Execution**
    11. **Data types**
    12. **If**
    13. **If..else**
    14. **If..else..if**
    15. **Nested if**
    16. **Switch**
    17. **Array**
    18. **1D arrays**
    19. **2d arrays**
    20. **3d arrays**
    21. **Jagged arrays**
    22. **For loop**
    23. **For each**
    24. **While**
    25. **Do..while..**
    26. **Functions(methods)**
    27. **Class**
    28. **Object**
    29. **Instance variables**
    30. **Static variables**
    31. **Instance methods**
    32. **Static methods**
    33. **Instance block**
    34. **Static block**
    35. **Constructors**
    36. **This keyword**
    37. **String constant pool**
    38. **Accessor and mutator methods**
    39. **Inheritance**
    40. **Super()**
    41. **Method overloading**
    42. **Method overriding**
    43. **Upcasting & Down casting**
    44. **Static and dynamic binding**
    45. **Compile time and runtime Polymorphism**
    46. **Abstract classes**
    47. **Interfaces**