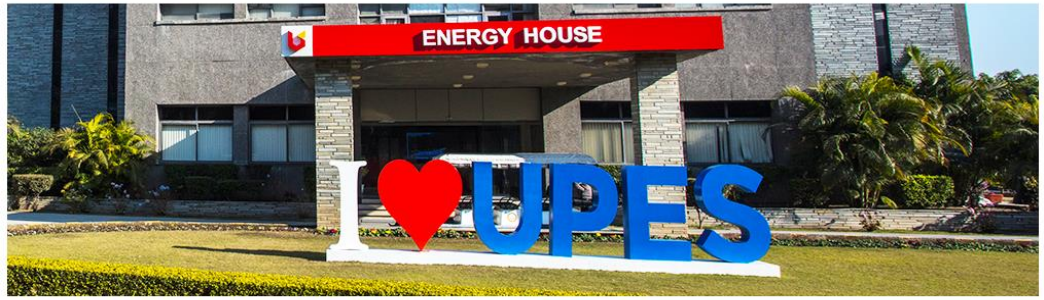


UPES

UNIVERSITY WITH A PURPOSE

Object Oriented Programming

UPES
UNIVERSITY WITH A PURPOSE

Constructors in Java

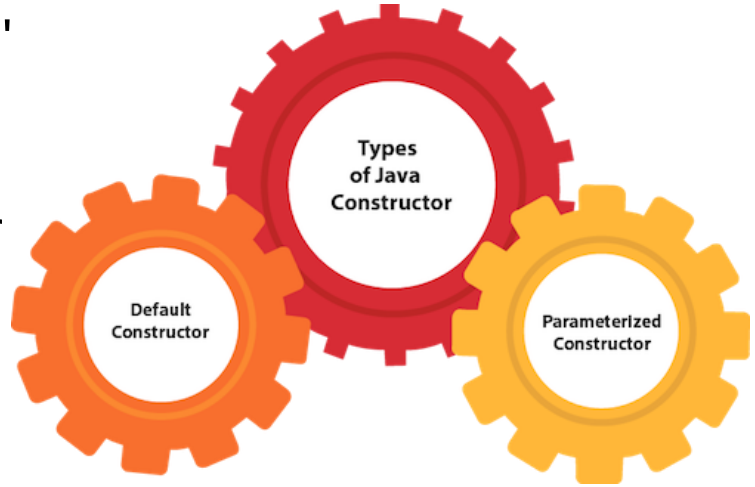
- In Java, a **constructor** is a block of code similar to the method. It is called when an instance of the class is created. At the time of calling constructor, memory for the object is allocated.
- It is a special type of method which is used to initialize the object. It is called constructor because it constructs the values at the time of object creation.
- Every time an object is created using the “new” keyword, at least one constructor is called.

Rules for creating Java constructor:

1. Constructor name must be the same as its class name.
2. A Constructor must have no explicit return type.
3. A Java constructor cannot be abstract, static, final.

Types of Java constructors

- A constructor is called "**Default Constructor**" when it doesn't have any parameter.



//Java Program to create and call a default constructor

```
class Bike1{
//creating a default constructor
Bike1(){System.out.println("Bike is crea
ted");}
//main method
public static void main(String args[]){
//calling a default constructor
Bike1 b=new Bike1();
}
}
```

Output:
Bike is created

UPES
UNIVERSITY WITH A PURPOSE

Java Default Constructor

- If there is no constructor in a class, compiler automatically creates a default constructor.

purpose of a default constructor?

The default constructor is used to provide the default values to the object like 0, null, etc., depending on the type.

UPES
UNIVERSITY WITH A PURPOSE

Java Default Constructor

//Example of default constructor that displays the default values

```
class Student3{
int id;
String name;
//method to display the value of id and name
void display(){System.out.println(id+" "+name);}
public static void main(String args[]){
//creating objects
Student3 s1=new Student3();
Student3 s2=new Student3();
//displaying values of the object
s1.display();
s2.display();  }  }
```

Output:

0 null
0 null

UPES
UNIVERSITY WITH A PURPOSE

Java Parameterized Constructor

- A constructor which has a specific number of parameters is called a parameterized constructor.

Why use the parameterized constructor?

The parameterized constructor is used to provide different values to distinct objects. However, you can provide the same values also.

```
//Java Program to demonstrate the use of the paramete
rized constructor.
class Student4{
    int id;
    String name;
    //creating a parameterized constructor
    Student4(int i,String n){
    id = i;
    name = n;       }
    //method to display the values
    void display(){System.out.println(id+" "+name);}

       public static void main(String args[]){
    //creating objects and passing values
    Student4 s1 = new Student4(111,"Karan");
    Student4 s2 = new Student4(222,"Aryan");
    //calling method to display the values of object

    s1.display();
    s2.display();
   }    }
```

UPES
UNIVERSITY WITH A PURPOSE

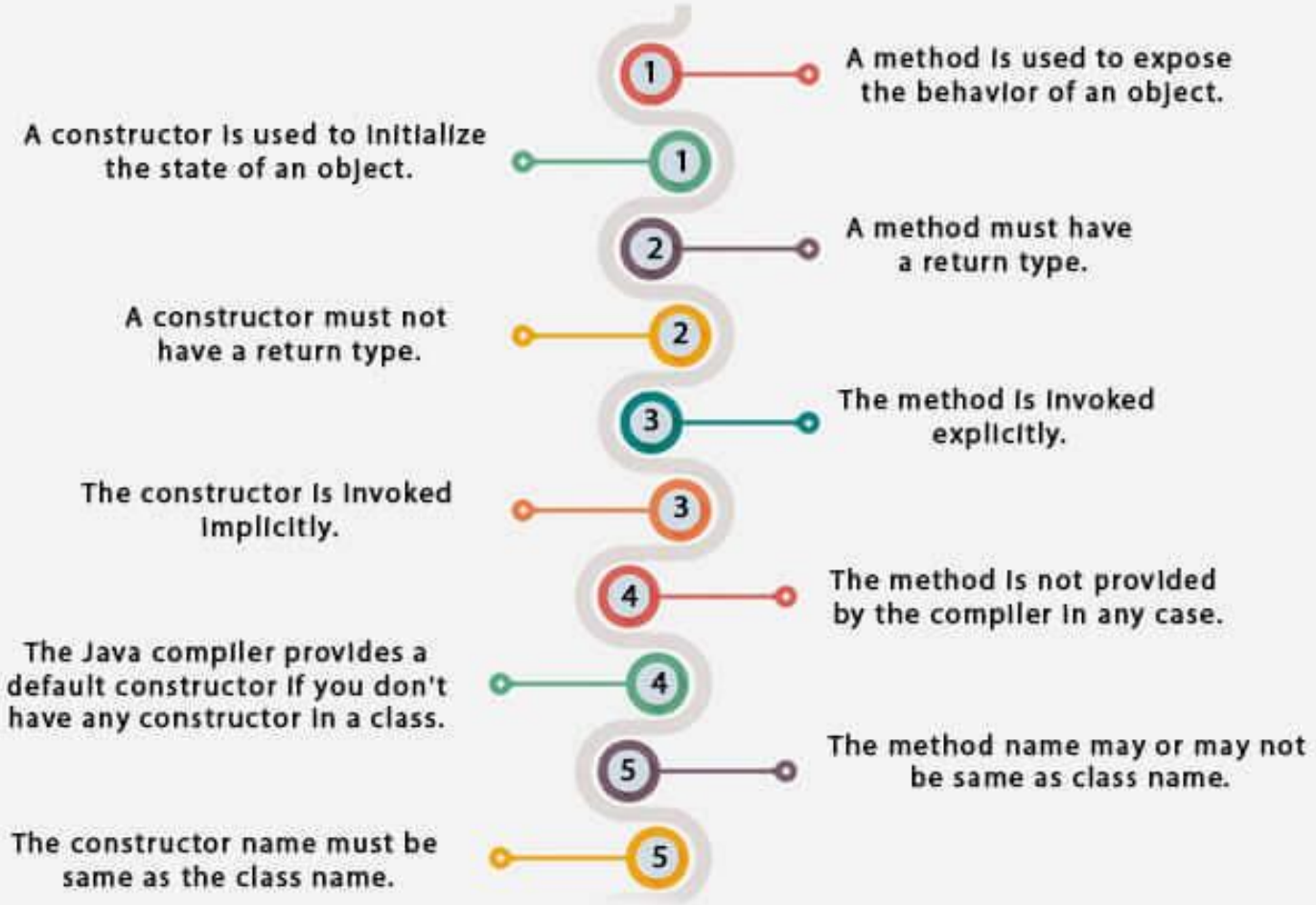
Constructor Overloading in Java

- In Java, a constructor is just like a method but without return type. It can also be overloaded like Java methods.
- Constructor overloading in Java is a technique of having **more than one constructors** with different parameter lists. They are arranged in a way that each constructor performs a different task. They are differentiated by the compiler by the number of parameters in the list and their types.

```
//Java program to overload constructors
class Student5{
    int id;  String name;  int age;
    //creating two arg constructor
    Student5(int i,String n){
    id = i;      name = n;  }
    //creating three arg constructor
    Student5(int i,String n,int a){
    id = i;   name = n;     age=a;  }
    void display(){System.out.println(id+" "+na
me+" "+age);}
    public static void main(String args[]){
    Student5 s1 = new Student5(111,"Karan");
    Student5 s2 = new Student5(222,"Aryan",25);

    s1.display();
    s2.display();
   }   }
```

UPES
UNIVERSITY WITH A PURPOSE

Difference between constructor and method in Java



UPES
UNIVERSITY WITH A PURPOSE

Java Copy Constructor

- There is no copy constructor in Java. However, we can copy the values from one object to another like copy constructor in C++.
- There are many ways to copy the values of one object into another in Java.

1. By constructor,
2. By assigning the values of one object into another,
3. By clone() method of Object class

```
//Java program to initialize the values from o
ne object to another object.
class Student6{
    int id;       String name;
    //constructor to initialize integer and st
ring
    Student6(int i,String n){
    id = i;      name = n;       }
   //constructor to initialize another object

    Student6(Student6 s){
    id = s.id;      name =s.name;       }
    void display(){System.out.println(id+" "+n
ame);}
   public static void main(String args[]){
    Student6 s1 = new Student6(111,"Karan");
    Student6 s2 = new Student6(s1);
    s1.display();
    s2.display();
   }    }
```

UPES
UNIVERSITY WITH A PURPOSE

Copying values without constructor

- We can copy the values of one object into another by assigning the objects values to another object. In this case, there is no need to create the constructor.

```
class Student7{
    int id;
    String name;
    Student7(int i,String n){
    id = i;
    name = n;       }
    Student7(){}
    void display(){System.out.println(id+" "+n
ame);}
       public static void main(String args[]){

    Student7 s1 = new Student7(111,"Karan");
    Student7 s2 = new Student7();
    s2.id=s1.id;
    s2.name=s1.name;
    s1.display();
    s2.display();
   }
}
```

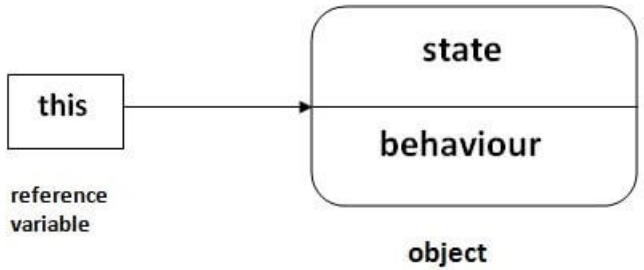
UPES
UNIVERSITY WITH A PURPOSE

Other Questions.

Q. Does constructor return any value?
Ans. Yes, it is the current class instance (You cannot use return type yet it returns a value).
Q. Can constructor perform other tasks instead of initialization?
Ans. Yes, like object creation. You can perform any operation in the constructor as you perform in the method.
Q. Is there Constructor class in Java?
Ans. Yes.
Q. What is the purpose of Constructor class?
Ans. Java provides a Constructor class which can be used to get the internal information of a constructor in the class. It is found in the **java.lang.reflect** package.

this keyword in java

- There can be a lot of usage of **java this keyword**. In java, this is a **reference variable** that refers to the current object.



Usage of java this keyword:

1. this can be used to refer current class instance variable.
2. this can be used to invoke current class method (implicitly)
3. this() can be used to invoke current class constructor.
4. this can be passed as an argument in the method call.
5. this can be passed as argument in the constructor call.
6. this can be used to return the current class instance from the method.

this keyword in java contd..

1) this: to refer current class instance variable

The this keyword can be used to refer current class instance variable. If there is ambiguity between the instance variables and parameters, this keyword resolves the problem of ambiguity.

- In the given example, parameters (formal arguments) and instance variables are same. So, we are using this keyword to distinguish local variable and instance variable.

Understanding the problem without this keyword:

```
class Student{
int rollno;
String name;
float fee;
Student(int rollno,String name,float fee){
rollno=rollno;
name=name;
fee=fee;  }
void display(){System.out.println(rollno+"
"+name+" "+fee);}  }
class TestThis1{
public static void main(String args[]){
Student s1=new Student(111,"ankit",5000f);
Student s2=new Student(112,"sumit",6000f);

s1.display();
s2.display();
}}
```

Output

0 null 0.0

0 null 0.0

UPES
UNIVERSITY WITH A PURPOSE

this keyword in java contd..

Solution of the previous problem by **this** keyword:

```
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();
}}
```

Output

111 ankit 5000

112 sumit 6000

this keyword in java contd..

- If local variables (formal arguments) and instance variables are different, there is no need to use this keyword like in the following program.
- It is better approach to use meaningful names for variables.

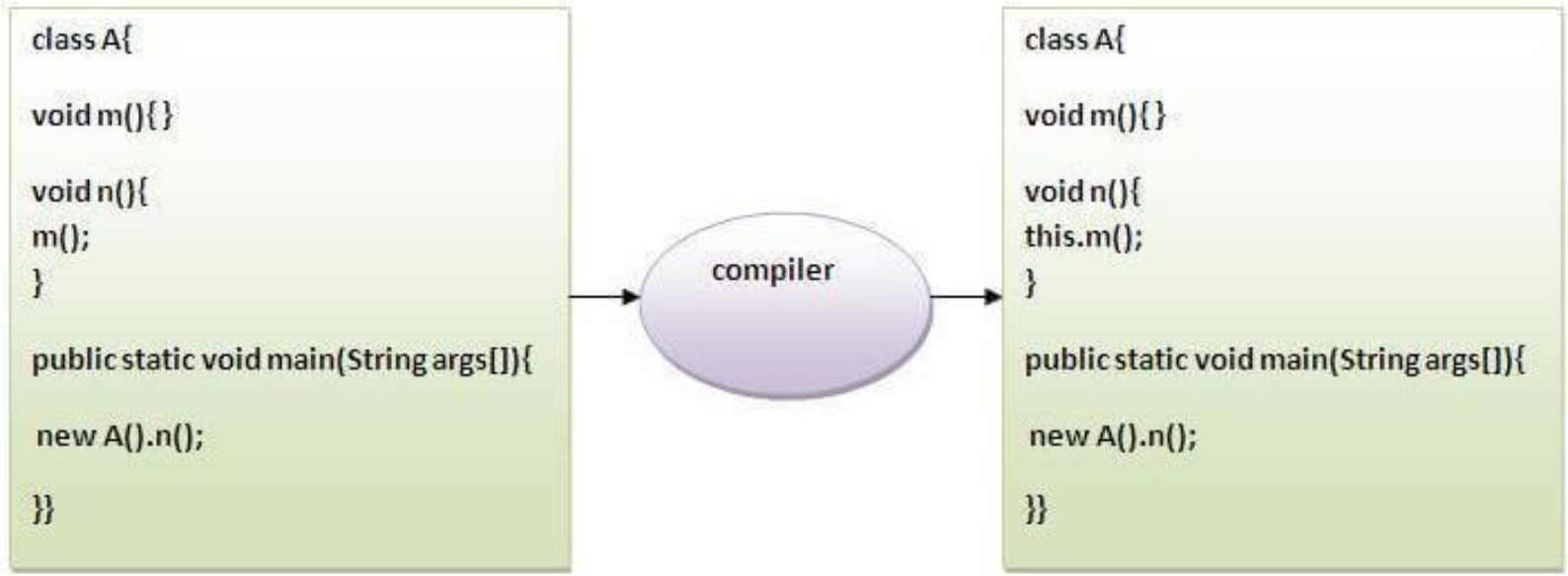
```
class Student{
int rollno;  String name;  float fee;

Student(int r,String n,float f){
rollno=r;  name=n;  fee=f;
}
void display(){System.out.println(rollno+"
"+name+" "+fee);}
}

class TestThis3{
public static void main(String args[]){
Student s1=new Student(111,"ankit",5000f);

Student s2=new Student(112,"sumit",6000f);

s1.display();
s2.display();
}}
```

Output
111 ankit 5000
112 sumit 6000

this keyword in java contd..

2) **this: to invoke current class method**

- You may invoke the method of the current class by using the this keyword. If you don't use this keyword, compiler automatically adds this keyword while invoking the method.



UPES
UNIVERSITY WITH A PURPOSE

this keyword in java contd..

3) **this() : to invoke current class constructor.**

- The this() constructor call can be used to invoke the current class constructor. It is used to reuse the constructor. In other words, it is used for constructor chaining.

Calling default constructor from parameterized constructor:

```
class A{
A(){System.out.println("hello a");}
A(int x){
this();
System.out.println(x);
}    }
class TestThis5{
public static void main(String args[]){
A a=new A(10);     }   }
```

Output:
hello a
10

UPES
UNIVERSITY WITH A PURPOSE

this keyword in java contd..

Calling parameterized constructor from default constructor:

```
class A{
A(){
this(5);
System.out.println("hello a");
}
A(int x){
System.out.println(x);
}
}
class TestThis6{
public static void main(String args[]){
A a=new A();
}}
```

Output:
5
hello a

UPES
UNIVERSITY WITH A PURPOSE

this keyword in java contd..

Output:
111 ankit java 0.0
112 sumit java 6000

Real usage of this() constructor call:

- The this() constructor call should be used to reuse the constructor from the constructor. It maintains the chain between the constructors i.e. it is used for constructor chaining.

```
class Student{
int rollno;
String name,course;
float fee;
Student(int rollno,String name
,String course){
this.rollno=rollno;
this.name=name;
this.course=course;   }
Student(int rollno,String name,String co
urse,float fee){
this(rollno,name,course);//reusing consts
ructor
this.fee=fee;   }
void display(){System.out.println(rollno
+" "+name+" "+course+" "+fee);}
}
class TestThis7{
public static void main(String args[]){

Student s1=new Student(111,"ankit","java
");
Student s2=new Student(112,"sumit","java
",6000f);
s1.display();
s2.display();
}}
```

UPES
UNIVERSITY WITH A PURPOSE

Java Variables

- **Variable** is name of *reserved area allocated in memory*. In other words, it is a *name of memory location*. It is a combination of "vary + able" that means its value can be changed.

Types of Variables:

There are three types of variables in java:

1. **local variable:** A variable declared inside the body of the method is called local variable. You can use this variable only within that method and the other methods in the class aren't even aware that the variable exists. A local variable cannot be defined with "static" keyword.

2. **instance variable:** A variable declared inside the class but outside the body of the method, is called instance variable. It is not declared as static. It is called instance variable because its value is instance specific and is not shared among instances.

3. **static variable:** A variable which is declared as static is called static variable. It cannot be local. You can create a single copy of static variable and share among all the instances of the class. Memory allocation for static variable happens only once when the class is loaded in the memory.

Java Variables contd..

Example to understand the types of variables in java:

```
class A{
int data=50;//instance variable
static int m=100;//static variable
void method(){
int n=90;//local variable
}
}//end of class
```

References

Gosling, J., Holmes, D. C., & Arnold, K. (2005). The Java programming language.

References

Gosling, J., Holmes, D. C., & Arnold, K. (2005). The Java programming language.