**OBJECT ORIENTED PROGRAMMING**

***Department of Software Engineering***

### Public Relations, MUET, SZAB Campus Khairpur Mir's - College & University | Facebook - 9,847 PhotosMentor:

#### Engr. Abdul Hafeez Babar [hafeez.babar@muetkhp.edu.pk](mailto:hafeez.babar@muetkhp.edu.pk)

***Mehran University of Engineering and Technology, SZAB Campus Khairpur Mir’s***



# Java Class Attributes and Java Methods



## Java Class Attributes

* Create a class called "Main" with two attributes: x and y:

### Example

public class Main {

int x = 5;

int y = 3;

}

***Note:*** Another term for class attributes is fields.



## Accessing Attributes

* + Access attributes by creating an object of the class, and by using the dot syntax (.):

### Example

* + - Create an object called "myObj" and print the value of x:

public class Main { int x = 5;

public static void main(String[] args) { Main myObj = new Main(); System.out.println(myObj.x);

}

}

## Modify Attributes



#### Example

* Set the value of x to 40:

public class Main { int x;

public static void main(String[] args) { Main myObj = new Main();

myObj.x = 40;

System.out.println(myObj.x);

}

}

**Or override existing values: Example**

* Change the value of x to 25:

public class Main { int x = 10;

public static void main(String[] args) { Main myObj = new Main();

myObj.x = 25; // x is now 25

System.out.println(myObj.x);

}

}



##### *If you don't want the ability to override existing values, declare the attribute as final:*

**Example**

public class Main { final int x = 10;

public static void main(String[] args) {

Main myObj = new Main();

myObj.x = 25; // will generate an error: cannot assign a value to a final variable

System.out.println(myObj.x);

}

}

* The final keyword is useful when you want a variable to always store the same value, like PI (3.14159...).
* The final keyword is called a "modifier".



## Multiple Objects

* If you create multiple objects of one class, you can change the attribute values in one object, without affecting the attribute values in the other:

### Example

##### *Change the value of x to 25 in myObj2, and leave x in myObj1 unchanged:*

public class Main {

int x = 5;

public static void main(String[] args) { Main myObj1 = new Main(); // Object 1 Main myObj2 = new Main(); // Object 2 myObj2.x = 25;

System.out.println(myObj1.x); // Outputs 5

System.out.println(myObj2.x); // Outputs 25

}

}



## Multiple Attributes

##### *You can specify as many attributes as you want:*

**Example**

public class Main { String fname = "John"; String lname = "Doe"; int age = 24;

public static void main(String[] args) { Main myObj = new Main();

System.out.println("Name: " + myObj.fname + " " +

myObj.lname);

System.out.println("Age: " + myObj.age);

}

}



# CONSTRUCTOR



## Constructors in Java

* + In Java, a constructor is a block of codes 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 in the memory.
  + It is a special type of method which is used to initialize the object.
  + Every time an object is created using the new() keyword, at least one constructor is called.
  + It calls a default constructor if there is no constructor available in the class.
  + In such case, Java compiler provides a default constructor by default.



## Constructor Types

* + - no-argument constructor
    - Default Constructor
    - parameterized constructor

***Note:*** It is called constructor because it constructs the values at the time of object creation. It is not necessary to write a constructor for a class. It is because java compiler creates a default constructor if your class doesn't have any.



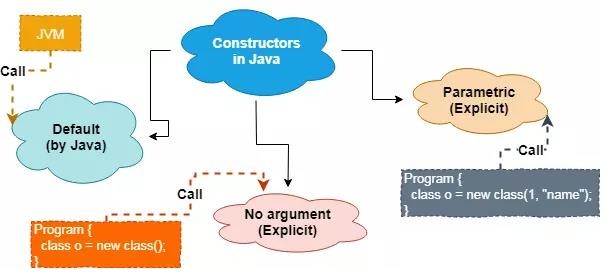
## Rules for creating Java constructor

##### *There are two rules defined for the constructor.*

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

***Note:*** We can use access modifiers while declaring a constructor. It controls the object creation. In other words, we can have private, protected, public or default constructor in Java.





## Java Default Constructor

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

### Syntax:z

class ClassName {

ClassName() {

}

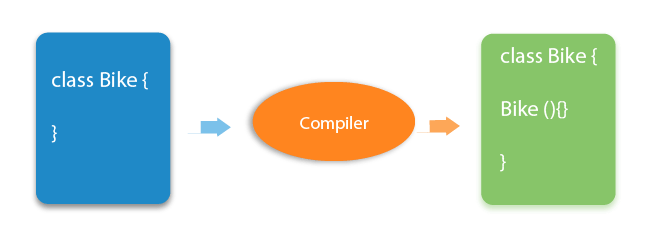
}

#### Example

Public class MyClass { Int num;

MyClass() { num = 100;

}

}

public class ConsDemo {

public static void main(String args[]) { MyClass t1 = new MyClass(); MyClass t2 = new MyClass();

System.out.println(t1.num + " " + t2.num);

}



}

***Rule***: If there is no constructor in a class, compiler automatically creates a default constructor.

## Why we use default Constructor?

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

the type.

### Example

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();

* In the above class, you are not creating any constructor so compiler provides you a default constructor. Here 0 and null values are provided by default constructor.

#### Output:



} 0 null

} 0 null



### Example

class Main {

int a; boolean b;

public static void main(String[] args) {

// A default constructor is called Main obj = new Main();

System.out.println("Default Value:"); System.out.println("a = " + obj.a); System.out.println("b = " + obj.b);

}

}



## 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.

### Example

* + Created the constructor of Student class that have two parameters. We can have any number of parameters in the constructor.

#### Output:

24 Waheed

22 Waqar

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(24,“Waheed"); Student4 s2 = new Student4(22,“Waqar");

//calling method to display the values of object s1.display();

s2.display();

}

}



### Example

public class Main { int modelYear; String modelName;

public Main(int year, String name) {

modelYear = year; modelName = name;

}

public static void main(String[] args) {

Main myCar = new Main(1969, "Mustang");

System.out.println(myCar.modelYear + " " + myCar.modelName);

}

}

// Outputs 1969 Mustang

## Java No-Arg Constructors

### Example : Java private no-arg constructor

class Main { int i;

// constructor with no parameter

private Main() { i = 5;

System.out.println("Constructor is called");

}

public static void main(String[] args) {

// calling the constructor without any parameter Main obj = new Main(); System.out.println("Value of i: " + obj.i);

}

Notice that we have declared the constructor as private.

Once a constructor is declared private, it cannot be accessed from outside the class. So, creating objects from outside the class is prohibited using the private constructor.

Here, we are creating the object inside the same

class. Hence, the program is able to access the

constructor.

}



However, if we want to create objects outside the class, then we need to declare the constructor as public.

### Example : Java public no-arg constructors

class Company {

String name;

// public constructor public Company() { name = “Hello";

#### Output:

Company name = Hello

}



}

class Main {

public static void main(String[] args) {

// object is created in another class

Company obj = new Company(); System.out.println("Company name = " + obj.name);

}

}



Hence, the Java compiler automatically creates the default constructor.

The default constructor initializes any uninitialized instance variables with default values.

### Type Default Value

boolean false

byte 0

short 0

int 0

long 0L

char \u0000

float 0.0f

double 0.0d

object Reference null



## Constructor Overloading

* 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 constructor 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.

**Example** //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+" "+name+" "+age);}

#### Output:

111 Ahmed 0

222 Waqar 25



public static void main(String args[]){ Student5 s1 = new Student5(22,“Ahmed"); Student5 s2 = new Student5(24,“Waqar",25); s1.display();

s2.display();

}

}

#### Example: Java Constructor Overloading

class Main { String language;

// constructor with no parameter

Main() {

this.language = "Java";

}

// constructor with a single parameter Main(String language) {

this.language = language;

**Output:**

}

public void getName() {

System.out.println("Programming Langauage: " + this.language);

Programming Language: Java

Programming Language: Python

}

public static void main(String[] args) {

// call constructor with no parameter Main obj1 = new Main();

// call constructor with a single parameter Main obj2 = new Main("Python");



obj1.getName();

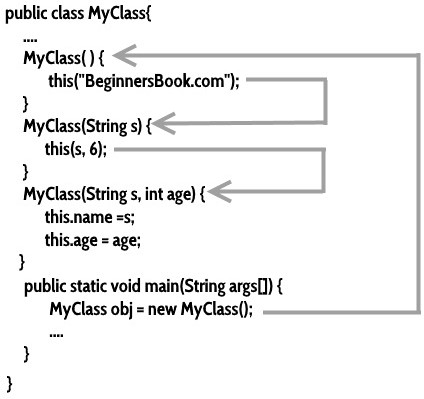
obj2.getName();

}

}

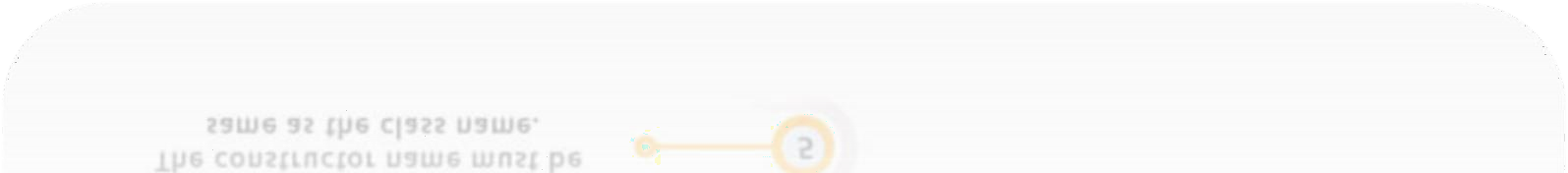


## Constructor Chaining

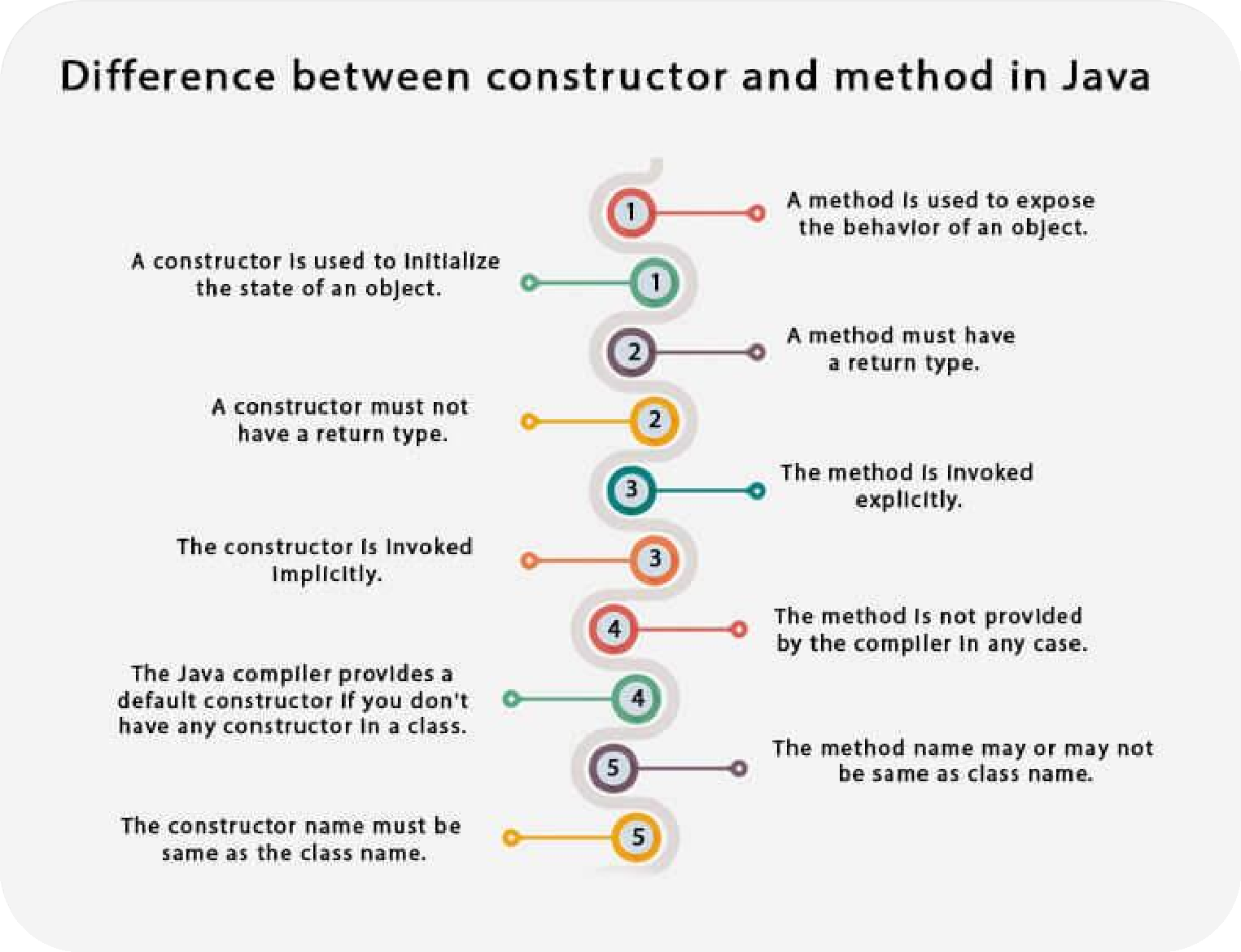




**Difference between constructor and method in Java**









## 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. They are:*

* + By constructor
  + By assigning the values of one object into another
  + By clone() method of Object class

**Example:**

In this example, we are going to copy the values of one object into another using Java constructor.

//Java program to initialize the values from one object to another object. class Student6{

int id;

String name;

//constructor to initialize integer and string

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+" "+name);}

#### Output:

24 Waqar

24 Waqar



public static void main(String args[]){ Student6 s1 = new Student6(24,“Waqar"); Student6 s2 = new Student6(s1); s1.display();

s2.display();

}

}

## 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.

### Example:

class Student7{ int id;

String name; Student7(int i,String n){ id = i;

name = n;

}

Student7(){}

void display(){System.out.println(id+" "+name);}

#### OUTPUT:

24 Waqar

24 Waqar



public static void main(String args[]){ Student7 s1 = new Student7(24,“Waqar"); Student7 s2 = new Student7(); s2.id=s1.id;

s2.name=s1.name; s1.display();

s2.display();

}

}



### Does constructor return any value?

* + Yes, it is the current class instance (You cannot use return type yet it returns a value).

#### Can constructor perform other tasks instead of initialization?

* + Yes, like object creation, starting a thread, calling a method, etc. You can perform any operation in the constructor as you perform in the method.

#### Is there Constructor class in Java?

* + - Yes.

#### What is the purpose of Constructor class?

* + 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.



## Some facts about Constructor in Java

* Constructor can be overloaded
* Constructor can be chained
* Call Constructor using this() and super()
* Constructor should have the same name as Class they belong to
* Constructor doesn't have return type
* Static initializer and instance initializer block are executed before the constructor
* Super class constructor is executed before subclass
* Construct invocation must be in fist line
* A constructor can not be final, abstract or synchronized
* Consider providing No argument constructor
* Constructor doesn't return anything

