SNS COLLEGE OF ENGINEERING

Kurumbapalayam (Po), Coimbatore – 641 107

**An Autonomous Institution**

Accredited by NAAC – UGC with 'A' Grade

Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

23CSB101

# OBJECT ORIENTED PROGRAMMING

## Constructors

By

M.Kanchana

Assistant Professor/CSE

## Constructors

- A **constructor** is a special function in a class that is automatically called when an object of the class is created. It initializes the object's properties.

- The constructor ensures that an object starts in a well-defined state.

## Constructors

```
public class Main {
    public static void main(String[] args)
{
    int x=10;  //local variable
    System.out.println(x);
    }
}
```

Output

```
10
```

```
public class Main {
    public static void main(String[] args)
{
    int x;
    System.out.println(x);
    }
}
```

Output:

```
variable x might not have been initialized
```

**local variables do NOT get default values** in Java.

## Constructors

```
public class Main {
 int x=10;  // instance

public static void main(String[] args)
{
System.out.println(x);
    }}
```

Output

**non-static variable x cannot be referenced from a static context**

This happens when you try to access a non-static variable inside a static method (like main()), without creating an object of the class

```
public class Main {
 int x;  // instance
    public static void main(String[] args)
{
Main obj =new Main();
System.out.println(obj.x);
    }
}
```

Output:

**0**

# Constructors

.

```
public class Main {
 int x;  // instance variable
    public static void main(String[] args)
{
Main obj =new Main();  // Java provides a default constructor
System.out.println(obj.x);
    }
}
```

Output:

**0**

int → 0
boolean → false
double → 0.0
String (or any object reference) → null

## Constructors

.
```
class Box {
    int width, height;

    Box() {
        width = 5;
        height = 10;
    }
}
```

If we relied on default values, both width and height would be 0.

## Constructors

### Rules for creating constructor:

·

1.Constructor name must be same as its class name
2.Constructor must have no explicit return type
3.Constructors can be declared public or private (for a Singleton)
4.Constructors can have no-arguments, some arguments.
5.A constructor is always called with the new operator
6.The default constructor is a no-arguments;
7.If you don't write ANY constructor, the compiler will generate the default one;
8.Constructors CAN'T be static, final or abstract;
9.When overloading constructors (defining methods with the same name but with different arguments lists) you must define them with different arguments lists (as number or as type)

# Constructors

Types of constructors

·

There are two types of constructors:

1. Default constructor
2. No-arg constructor
3. Parameterized constructor

# Constructors

- **Default Constructor**

```
class Box {

int width, height;
// No explicit constructor

 }
```

**No arg Constructor**

```
class Box {
    int width, height;

    Box() {
        width = 5;
        height = 10;
    }
}
```

## Constructors

- **Parameterized constructor**
  - A constructor that takes parameters is known as parameterized constructor.
  - Used to provide different values to the distinct objects.

```
   Box(double w, double h, double d)
{
width=w;
height=h;
depth=d;
}
```

## Constructors

```
class Box
{
double width;
double height;
double depth;
Box(double w, double h, double d)
{
width=w;
height=h;
depth=d;
}
double volume()
{
return width*height*depth;
}}
```

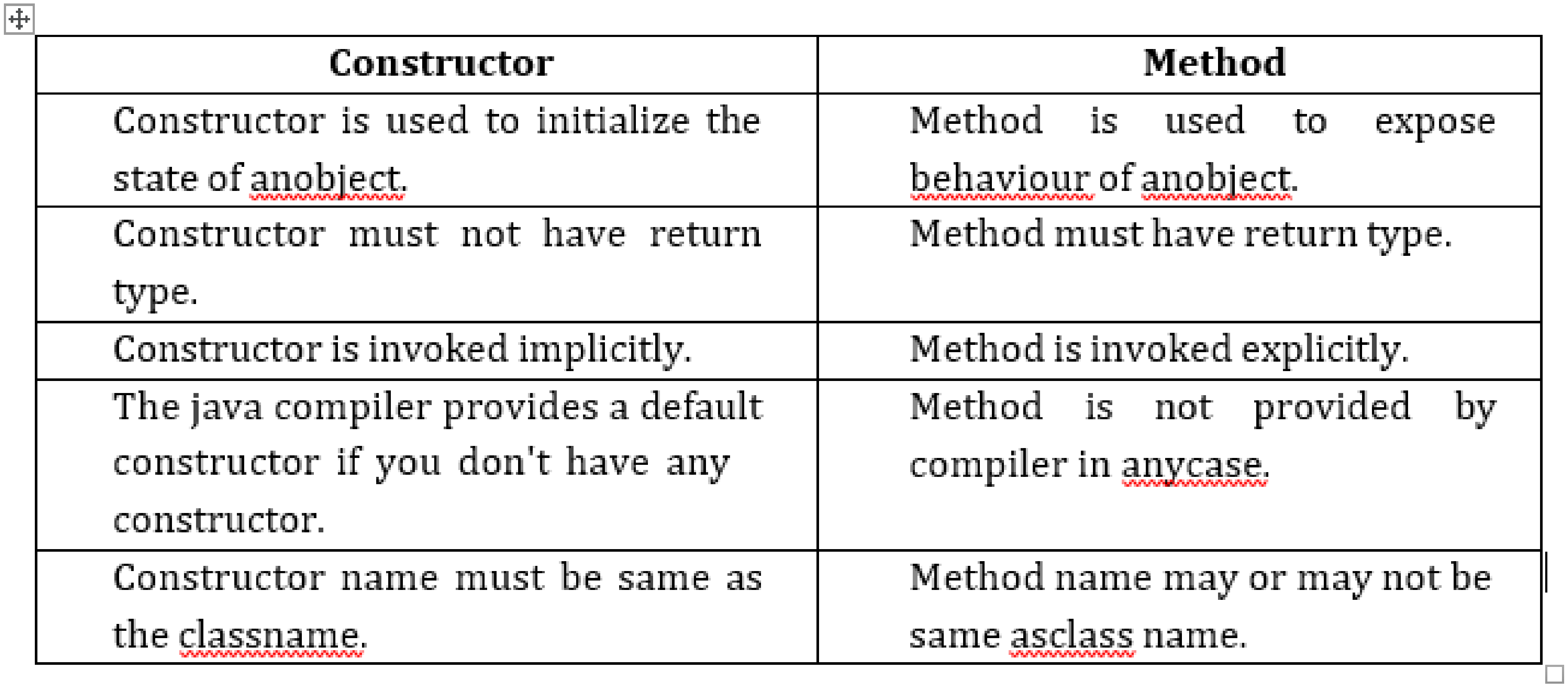
```
class BoxDemo
{
public static void main(String arg[])
{
Box mybox1=new Box(10,20,15);
Box mybox2=new Box(3,6,9);
double vol;

// Get volume of First box
vol=mybox1.volume();
System.out.println("Volume is " +vol);

// Get volume of second box
vol=mybox2.volume();
System.out.println("Volume is " +vol);
}}
```

## Constructors

.

```
Output:
Volume is 3000.0
Volume is 162.0
```

# Constructors



| Constructor | Method |
|---|---|
| Constructor is used to initialize the state of anobject. | Method is used to expose behaviour of anobject. |
| Constructor must not have return type. | Method must have return type. |
| Constructor is invoked implicitly. | Method is invoked explicitly. |
| The java compiler provides a default constructor if you don't have any constructor. | Method is not provided by compiler in anycase. |
| Constructor name must be same as the classname. | Method name may or may not be same asclass name. |

## Constructors

- **this** keyword

  In java, this is a reference variable that refers to the current object.

  **Uses:**

  1. this keyword can be used to refer current class instance variable.
  2. this() can be used to invoke current class constructor.
  3. this keyword can be used to invoke current class method (implicitly)
  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 keyword can also be used to return the current class instance.

## Constructors

•

**Name collision:**

- happens when a local variable (such as a method parameter) has the same name as an instance variable (class variable).
- In such cases, the **this** keyword is used to differentiate the instance variable from the local variable.

## Constructors

**1. this keyword can be used to refer current class instance variable**

•

```
class Example {
    int value;
void setValue(int value)
{
        this.value = value;  // Refers to the instance variable
    }
    void display() {
        System.out.println("Value: " + this.value);
    }
    public static void main(String[] args) {
        Example obj = new Example();
        obj.setValue(10);
        obj.display();  // Output: Value: 10
    }}
```

## Constructors

### Constructor Chaining :

•

- Constructor chaining happens when one constructor calls another constructor within the same class using **this().**

```
Person() {
    this(“Kavin"); // calls another constructor
  }
```

# Constructors

## 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.

```
Person() {
      this(“Kavin");
   }
   Person(String name) {
      this.name = name;
   }
```

# Constructors

## 2. this() can be used to invoke current class constructor.

```
• class Person {
      String name;

      Person() {
          this(“Kavin");
      }
      Person(String name) {
          this.name = name;
      }
      void display() {
          System.out.println("Name: " + this.name);
      }
```

```
public static void main(String[] args) {
    Person p1 = new Person();
    Person p2 = new Person(“Navin");
    p1.display();  // Output: Name: Kavin
    p2.display();  // Output: Name: Navin
}}
```

## Constructors

### 3.this to Invoke the Current Class Method (Implicitly)

•
```
class Example {

    void show() {
        System.out.println("Show method called");
    }

    void display() {
        this.show();  // `this` is optional here
    }

    public static void main(String[] args) {
        Example obj = new Example();
        obj.display();  // Output: Show method called
    }}
```

# Constructors

## 4. **this Passed as an Argument in Method Call**

•

```
class Example {
    void method1(Example obj) {
        System.out.println("Method called using 'this' as an argument");
    }
    void method2() {
        method1(this);  // Passing the current instance
    }
    public static void main(String[] args) {
        Example obj = new Example();
        obj.method2();  // Calls method2(), which in turn calls method1(this)
    }
}
```

## Constructors

### 5.this Passed as an Argument in Constructor Call

```
class B {
    B(A obj) {
        System.out.println("Constructor of B is called");
    }
}
class A {
    void method() {
        B obj = new B(this);  // Passing current instance
    }

    public static void main(String[] args) {
        A obj = new A();
        obj.method();  // Output: Constructor of B is called
    }}
```

## Constructors

### 6.this Used to Return the Current Class Instance

```
class Example {
    Example getObject() {
        return this;  // Returning the current instance
    }

    void display() {
        System.out.println("Method called on returned instance");
    }

    public static void main(String[] args) {
        Example obj = new Example();
        obj.getObject().display();  // Calls display() on the returned
instance(method chaining )
    }
}
```

## Constructors

### Method Chaining

- Method chaining is a technique in Java (and other programming languages) where multiple methods are called on the same object in a single line.

- It works because each method returns the current object (this), allowing the next method to be called immediately on the returned instance.

# Constructors

## Method Chaining

Imagine you walk into a library and ask the librarian:
You: "Can I borrow a book?"
Librarian: "You already have the book with you!"

Since getObject() returns the current object (this), we can call methods on the returned instance in one line instead of writing:

Example obj = new Example();
**obj.getObject();**
**obj.display();**

We combine them into:
**obj.getObject().display();**

THANK YOU