Constructor, Interface

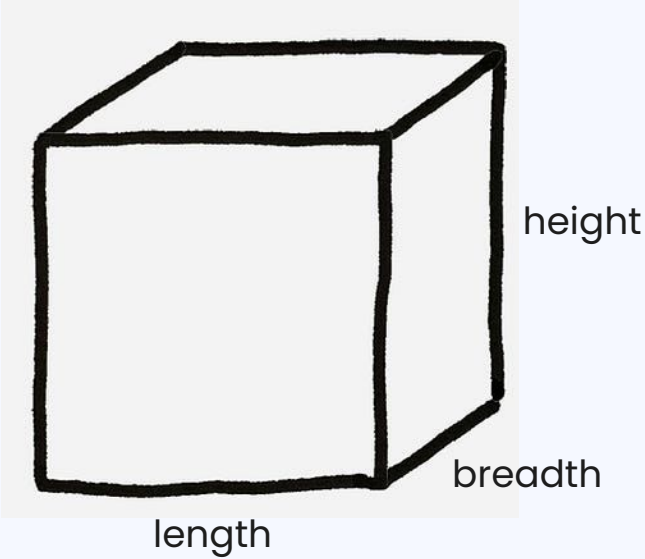
Ratna Mufidah, S.Kom., M.Kom.

Constructor in Java

A constructor in Java is a **special method (a block of codes similar to the method)** that is **used to initialize objects**.

The constructor is **called when an object of a class is created**. It can be used to **set initial values** for **object attributes**.

Why is a constructor needed?



Think of a Box

What do you need if you want to create **an object**?

The Rules for Creating Constructors

1. The **constructor's and class's name** must be **identical**
2. A Constructor **must have no explicit return type**
3. A constructor **cannot** be any of these: **static, synchronized, abstract,** or **final**
4. **Access modifiers can be used in constructor** declaration to control its access i.e which other class can call the constructor

➔ Constructors are **called only once at the time of Object creation** while method(s) can be called any number of times.

The Syntax for The Constructor

```
class ClassName
{
   .......

   // A Constructor
   ClassName() {
   }

   .......
}
```

When Constructor is called?

Each time an object is created using a **new()** keyword

```
// This statement calls above constructor.
ClassName obj = new ClassName();
```

Types of Constructors in Java

1. Default Constructor
2. Parameterized Constructor
3. Copy Constructor

Default Constructor

A constructor that has **no parameters**

Syntax:

`<class_name>(){}`

dcons.java

```java
public class dcons {
    // Default Constructor
    public dcons() {
        System.out.println("Default Constructor Called");
    }
}
```

ConsMain.java

```java
public class ConsMain {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        dcons dc1 = new dcons();
    }

}
```

Parameterized Constructor

A constructor that has **parameters**

Syntax:

```
<class_name>(parameter){}
```

Box.java

```java
public class Box {
    int length;
    int breadth;
    int height;

    //Parameterized Constructor
    public Box(int length, int breadth, int height) {
        this.length = length;
        this.breadth = breadth;
        this.height = height;
    }

    public void info() {
        System.out.println("length = "+this.length);
        System.out.println("breadth = "+this.breadth);
        System.out.println("height = "+this.height);
    }
}
```

Parameterized Constructor

BoxMain.java

```
2  public class BoxMain {
3
4      public static void main(String[] args) {
5          // TODO Auto-generated method stub
6          Box b1 = new Box(6, 4, 5);
7          b1.info();
8      }
9
10 }
```

Copy Constructor

A a constructor that **creates an object using another object** of the same Java class

Student.java

```java
public class Student {
    // data members of the class.
    String name;
    int id;

    // Parameterized Constructor
    public Student(String name, int id) {
        // TODO Auto-generated constructor stub
        this.name = name;
        this.id = id;
    }

    // Copy Constructor
    public Student(Student ostudent) {
        // TODO Auto-generated constructor stub
        this.name = ostudent.name;
        this.id = ostudent.id;
    }

}
```

Copy Constructor

StudentMain.java

```java
public class StudentMain {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        // This would invoke the parameterized constructor.
        System.out.println("First Object");
        Student std1 = new Student("Anisa", 21);
        System.out.println("Student Name :" + std1.name
                           + " and Student ID :" + std1.id);

        System.out.println();

        // This would invoke the copy constructor.
        System.out.println("Second Object");
        Student std2 = new Student(std1);
        System.out.println("Copy Constructor used Second Object");
        System.out.println("Student Name :" + std2.name
                           + " and Student ID :" + std2.id);
    }

}
```

Interface in Java

An interface is a completely "abstract class" that is used to group related methods with empty bodies

Another way to achieve **abstraction** in Java

Syntax for Java Interfaces

```
interface <Interface_name> {

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

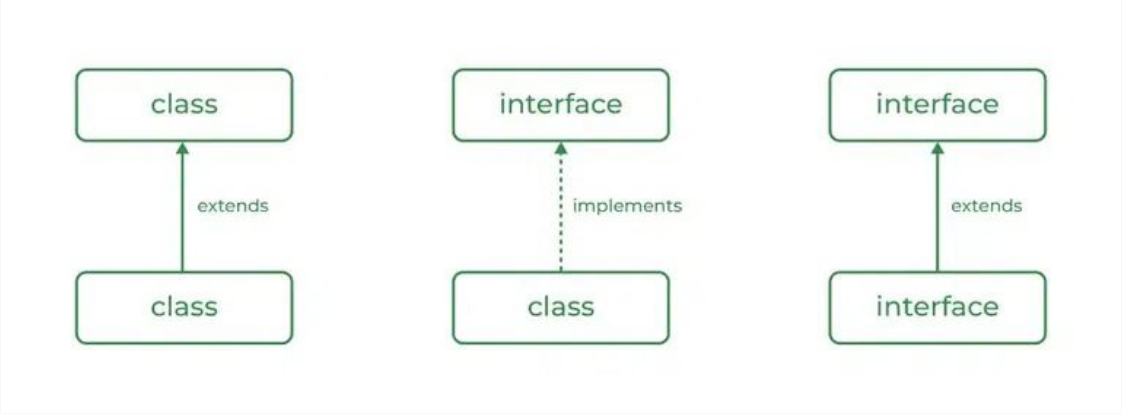
Why And When To Use Interfaces?

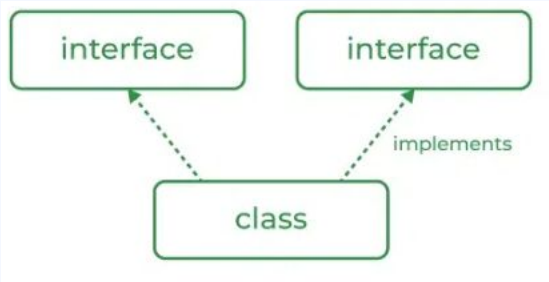
1. To achieve security - hide certain details and only show the important details of an object (interface).

2. Java does not support "multiple inheritance" (a class can only inherit from one superclass). However, it can be achieved with interfaces, because the class can implement multiple interfaces.

3. Abstract classes may contain non-final variables, whereas variables in the interface are final, public, and static.

Relationship Between Class and Interface





Keyword: **implements**

Example - 1

Animal.java

```java
interface Animal {
    // interface
      public void animalSound(); // interface method (does not have a body)
      public void run(); // interface method (does not have a body)
}
```

Cat.java

```java
class Cat implements Animal{
    public void animalSound() {
        // The body of animalSound() is provided here
        System.out.println("The cat says: miauw miauw");
      }
      public void run() {
        // The body of sleep() is provided here
        System.out.println("The cat can run fast");
      }
}
```

Example - 1

AnimalMain.java

```java
public class AnimalMain {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Cat cat1 = new Cat();  // Create a Cat object
        cat1.animalSound();
        cat1.run();
    }

}
```

Example - 2

Interface1.java

```
1 interface Interface1 {
2     public void Method1();
3 }
```

Interface2.java

```
1 interface Interface2 {
2     public void Method2();
3 }
```

DemoInterface.java

```
1 class DemoInterface implements Interface1,Interface2 {
2       public void Method1() {
3             System.out.println("My method is...");
4       }
5       public void Method2() {
6             System.out.println("My other method is...");
7       }
8 }
```

Example - 2

InterfaceMain.java

```
1  public class InterfaceMain {
2
3      public static void main(String[] args) {
4          // TODO Auto-generated method stub
5          DemoInterface di1 = new DemoInterface();
6          di1.Method1();
7          di1.Method2();
8      }
9
10 }
```

Thanks!

Ada pertanyaan?

References

https://www.geeksforgeeks.org/
https://www.baeldung.com/
https://www.w3schools.com/
https://www.javatpoint.com/


slidesgo