



A T M E
College of Engineering

GOLD

QS I-GAUGE
INDIAN CYBER RATING



Object-Oriented Programming with Java

Module 2: INTRODUCING CLASSES AND METHODS



Introducing Classes

Class Fundamentals

A class is a blueprint or template used to create objects. It defines the properties (data members) and behaviors (methods) of an object.

Syntax

```
class ClassName {  
    // data members // methods  
}
```

Example

```
class Student {  
    int id; String name;  
}
```

Here, Student is a class containing two data members: id and name.

Declaring Objects

An object is an instance of a class.

Syntax

ClassName objectName;

Example

Student s1;

This only declares a reference variable. No object is created yet.

Assigning Object Reference Variables

To create an object, use the new keyword.

Syntax

```
ClassName objectName = new ClassName();
```

Example Program

```
class Student {  
    int id;  
    String name;  
    void display() {  
        System.out.println(id + " " + name);  
    }  
    public static void main(String[] args) {  
        Student s1 = new Student();  
        s1.id = 101;  
        s1.name = "Divya";  
        s1.display();  
    }  
}
```

Introducing Methods

A method is a block of code that performs a specific task.

Syntax

```
returnType methodName(parameters) { // statements }
```

Example Program

```
class Addition { int add(int a, int b) {  
    return a + b; }  
public static void main(String[] args) {  
    Addition obj = new Addition();  
    int result = obj.add(10, 20);  
    System.out.println("Sum = " + result); } }
```

Output

Sum = 30

Constructors

A constructor is a special method used to initialize objects.

Characteristics

- Constructor name is the same as the class name.
- It has no return type.
- It is called automatically when an object is created.

Default Constructor

```
class Student {  
    Student() { System.out.println("Constructor called"); }  
    public static void main(String[] args) {  
        Student s1 = new Student(); } }  
}
```

Output

Constructor called

Parameterized Constructor

```
class Student { int id;
```

String name;

```
    Student(int i, String n) {  
        id = i; name = n; }  
    void display() {  
        System.out.println(id + " " + name); }  
    public static void main(String[] args) {  
        Student s1 = new Student(101, "Divya");  
        s1.display(); } }  
}
```

The this Keyword

this refers to the current object.

Uses of this

- 1.Refer to current class variables
- 2.Invoke current class methods
- 3.Invoke current class constructors

Example Program

```
class Student { int id;  
    Student(int id) {  
        this.id = id; }  
    void display() {  
        System.out.println("ID = " + this.id); }  
    public static void main(String[] args) {  
        Student s1 = new Student(101);  
        s1.display(); }  
}
```

Garbage Collection

Garbage collection automatically destroys objects that are no longer referenced.

Example Program

```
class Test {  
    public void finalize() {  
        System.out.println("Object destroyed"); }  
    public static void main(String[] args) {  
        Test obj = new Test();  
        obj = null;  
        System.gc();  
    }  
}
```

Methods and Classes

Method Overloading

Method overloading means defining multiple methods with the same name but different parameter lists.

Example Program

```
class MathOp {  
    int add(int a, int b) {  
        return a + b; }  
    int add(int a, int b, int c) {  
        return a + b + c; }  
    public static void main(String[] args) {  
        MathOp obj = new MathOp();  
        System.out.println(obj.add(10, 20));  
        System.out.println(obj.add(10, 20, 30)); } }
```

Using Objects as Parameters

Objects can be passed as arguments to methods.

Example Program

```
class Number {  
    int value;  
    Number(int v) {  
        value = v;  
    }  
    void compare(Number n) {  
        if (value == n.value) System.out.println("Equal");  
        else System.out.println("Not Equal"); }  
    public static void main(String[] args) {  
        Number n1 = new Number(10);  
        Number n2 = new Number(10);  
        n1.compare(n2);  
    }  
}
```

Argument Passing

Java uses **pass by value**.

Primitive Example

```
class Test {  
    void change(int x) { x = 50; }  
    public static void main(String[] args) {  
        Test obj = new Test();  
        int a = 10;  
        obj.change(a);  
        System.out.println(a);  
    } }
```

Returning Objects

A method can return an object.

Example Program

```
class Box { int width; Box(int w) {  
width = w; }  
Box create() {  
Box b = new Box(50);  
return b; }  
public static void main(String[] args) {  
Box b1 = new Box(10);  
Box b2 = b1.create();  
System.out.println(b2.width);  
}  
}
```

Recursion

Recursion is the process in which a method calls itself.

Example: Factorial

```
class Factorial { int fact(int n) {  
    if (n == 1) return 1; else return n * fact(n - 1); }  
public static void main(String[] args) {  
    Factorial f = new Factorial();  
    System.out.println(f.fact(5));  
    } }
```

Access Control

Java provides four access specifiers.

Access Specifier	Same Class	Same Package	Subclass	Other Package
private	Yes	No	No	No
default	Yes	Yes	No	No
protected	Yes	Yes	Yes	No
public	Yes	Yes	Yes	Yes

Example

```
class Demo {  
    private int a = 10;  
    void show() { System.out.println(a); }  
    public static void main(String[] args) {  
        Demo d = new Demo();  
        d.show(); } }
```

static

A static member belongs to the class rather than objects.

Static Variable

```
class Student { int id;  
    static String college = "ATME";  
    Student(int i) { id = i; }  
    void display() {  
        System.out.println(id + " " + college); }  
    public static void main(String[] args) {  
        Student s1 = new Student(101); ,  
        Student s2 = new Student(102);  
        s1.display();  
        s2.display(); } }
```

Static Method

```
class Test {  
    static void greet() {  
        System.out.println("Welcome"); }  
    public static void main(String[] args) {  
        Test.greet();  
    }  
}
```

final

The final keyword is used for variables, methods, and classes.

Final Variable

```
class Demo {  
    final int MAX = 100; void show() {  
        System.out.println(MAX); }  
    public static void main(String[] args) {  
        Demo d = new Demo();  
        d.show(); } }
```

Final Method

```
class A { final void display() {  
System.out.println("Final Method"); } }  
class B extends A { // display() cannot be overridden }
```

Final Class

```
final class A { } //  
class B extends A { } // Error
```

Nested and Inner Classes

A class declared inside another class is called a nested class.

Static Nested Class

```
class Outer {  
    static class Inner { void show() {  
        System.out.println("Static Nested Class"); } }  
    public static void main(String[] args) {  
        Outer.Inner obj = new Outer.Inner();  
        obj.show(); } }
```

Inner Class (Non-static)

```
class Outer {  
    class Inner {  
        void show() {  
            System.out.println("Inner Class"); } }  
    public static void main(String[] args) {  
        Outer out = new Outer();  
        Outer.Inner in = out.new Inner();  
        in.show(); } }
```



A T M E
College of Engineering

GOLD

QS I-GAUGE
INDIAN CYBER RATING



THANK YOU