7 - Java Programming

API

-> Application Programming Inter

- Library (Pre written code to faster the development process).

Java Editions

-> 3 Types of edition.

- Java Standard Edition (SE)
- Java Enterprise Edition (EE)
- Java Micro Edition (ME)

Methods

Definition: Methods are **functions defined inside a class** that describe the **behavior** of objects of that class.
Every Method is written inside a class. A class is a container of methods.

Naming Convention

- **Pascal Case Convention:** `ThisIsAName`
- **Camel Case Convention:** `thisIsAName`
- **Snake Case Convention:** `this_is_a_name`

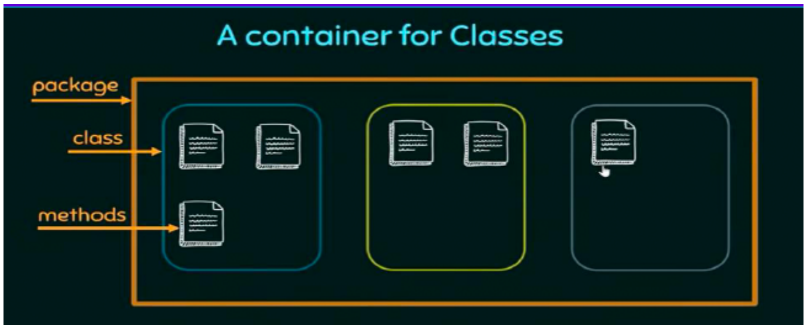
Structure

-> Each java program must have a class that contain main() method.

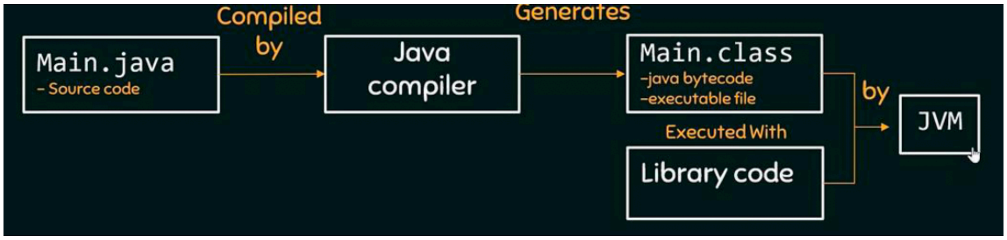
```
public class temp {
    public static void main(String[] args) {

    }
}
```

Package

A container of classes.


A container for Classes

Compiling Diagram

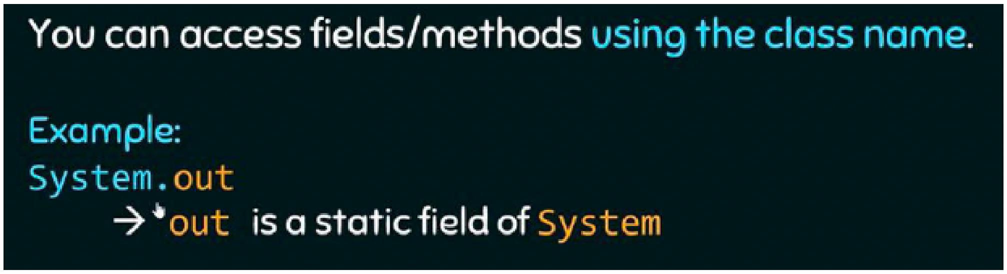


Access Modifier

1. **Public:** access from everywhere.
2. **Private:** access only insider the class.
3. **Protected:** access from everywhere.
4. **Default:** access from everywhere.

Non-access Modifier

-> Static


You can access fields/methods using the class name.

Example:
System.out
→ out is a static field of System

Calling non-static method from static method

- create a object of the class then all types of the method can be called by it.

```
public class code {
    public static void main(String[] args) {
        code obj = new code();
        obj.display();
    }
    private void display(){
        System.out.println("This is non-static method");
    }
}
```

Primitive vs Reference Type

- Primitive → actual value
- Reference → address of an object

Immutable Objects

- An object whose content can not be changed

Scanner class

- input.next() --> reads a string
- input.nextln() --> reads a int
- .nextChar() --> nothing is there like this

```
import java.util.Scanner;

public class ain{
    public static void main(String[] args){
        Scanner input = new Scanner(System.in);
        int x = input.nextInt();
    }
}
```

Array

```
datatype arr[] = new datatype[size];
```

Array pass by reference:

```
public class code{
    public static void main(String[]args){
        int num[] = {1,2};
        change(num);
        printarray(num);
    }
    public static void change(int arr[]){
        arr[0] = 2;
        arr[1] = 1;
    }
    public static void printarray(int arr[]) {
        for (int i = 0; i < arr.length; i++) {
            System.out.print(arr[i] + " ");
        }
        System.out.println();
    }
}
```

Anonymous Object

- A value that has been created but has no name.

```
import java.awt.Point; // --> must needed
public class code{
    public static void main(String[]args){
        Point p = getPoint();
        System.out.println("X: " + p.x + " Y: " + p.y);
    }
    public static Point getPoint(){
        return new Point(1,2);
    }
}
```

- array of anonymous object.

```
public class code{
    public static void main(String[]args){
        int num[] = getnums();
        printarray(num);
    }
    public static int[] getnums(){
        return new int[]{1,2,3,4,5};
    }
    public static void printarray(int arr[]){
        for(int i=0;i<arr.length;i++){
            System.out.print(arr[i] + " ");
        }
        System.out.println();
    }
}
```

Array class methods

```
    Arrays.sort(arr);
    int i = Arrays.binarySearch(arr, 4); // if not found returns -ve
    Arrays.fill(arr, fromIndex, toIndex, value);
    System.out.println(Arrays.toString(arr)) // [1, 2, 3]
    arr.length() // size
```

ArrayList class

```
import java.util.ArrayList;

public class code{
    public static void main(String[]args){
        ArrayList<Integer> list = new ArrayList();
        list.add(10); // .add(value)
        list.add(20);
        list.add(30);
        list.add(40);
        list.add(50);
        System.out.println("ArrayList: " + list);
        list.remove(2); // .remove(index)
        System.out.println("After removing element at index 2: " + list);
        list.set(1, 25); // .set(index, value)
        System.out.println("After setting index 1 to 25: " + list);
        int value = list.get(3); // .get(index)
        System.out.println("Element at index 3: " + value);
        int size = list.size(); // .size()
        System.out.println("Size of the ArrayList: " + size);
    }
}
```

Output:

```
ArrayList: [10, 20, 30, 40, 50]
After removing element at index 2: [10, 20, 40, 50]
After setting index 1 to 25: [10, 25, 40, 50]
Element at index 3: 50
Size of the ArrayList: 4
```

for-each loop

```java
import java.util.ArrayList;

public class code{
    public static void main(String[]args){
        ArrayList<Integer> list = new ArrayList();
        list.add(10); // .add(value)
        list.add(20);
        list.add(30);
        list.add(40);
        list.add(50);
        System.out.println("ArrayList: ");
        for (int u : list){
            System.out.println(u);
        }
    }
}
```

Inheritance

```java
class calculator{ // shurute public deuya jabe na
    int s;
    int add(int x, int y){
        s = x + y;
        return s;
    }
    int sub(int x, int y){
        s = x - y;
        return s;
    }
}

public class code extends calculator{
    public static void main(String[] args){
        code cal = new code();
        System.out.println("Sum: " + cal.add(10,20));
        System.out.println("Difference: " + cal.sub(30,15));
    }
}
```

Uses of super keyword

- used to refer immediate parent class instance variable.

- `super()` used to invoke immediate parent class constructor.

```
class Human{
    String name = "Human";
}

class Person extends Human{
    String name = "Person";
    public void show(){
        System.out.println(super.name);
    }
}
class Employee extends Person{
    String name = "Employee";
    public void show(){
        System.out.println(name);
        System.out.println(super.name);
        super.show();
    }
}
public class code{
    public static void main(String[] args){
        Employee emp = new Employee();
        emp.show();
    }
}
```

Output:

```
Employee
Person
Human
```

super() method

```
class Person{
    String name = "Person";
    public Person(){
        System.out.println("Name: " + name);
    }
    public Person(String s){
        name = s;
        System.out.println("Name: " + name);
    }
}
class Employee extends Person{
    String name = "Employee";
    public Employee(){
        // super();
        super("priashis");
        System.out.println("Name: " + name);
    }
}
public class code{
    public static void main(String[] args){
        Employee e = new Employee();
    }
}
```

Output:

```
Name: priashis
Name: Employee
```

Abstract Class

- A class which contains the abstract keyword in its declaration is known as abstract class.
- If a class is declared abstract cannot be instantiated.

```
abstract class Vehicle {
    abstract void start();
}

public class Main {
    public static void main(String[] args) {
        Vehicle v = new Vehicle(); // ❌ ERROR: Cannot instantiate abstract
class
    }
}
```

Example:

```
abstract class Shape{
    String name = "Shapre";
    public abstract void display();
}
```

```
class Circle extends Shape{
    String name = "Circle";
    public void display(){
        System.out.println("Name :" + name);
    }
}

public class code{
    public static void main(String[] args) {
        Circle c = new Circle();
        c.display();
    }
}
```

Output:

```
Name :Circle
```

Interface

- An **interface** is a **blueprint of a class**.
- It contains:
  - **Abstract methods** (methods without a body).
  - **Constants** (variables that are `public static final` by default).
- **Cannot** have constructors (because you cannot instantiate an interface).
- Achieves **abstraction**: hides implementation details, shows only functionality.
- Supports **multiple inheritance** (a class can implement multiple interfaces).

```
interface FI {
    public void displayFI();
}

interface SI {
    public void displaySI();
}

class Demo implements FI, SI {
    public void displayFI() {
        System.out.println("From FI");
    }

    public void displaySI() {
        System.out.println("From SI");
    }
}

public class code {
    public static void main(String[] args) {
        Demo d = new Demo();
        d.displayFI();
        d.displaySI();
    }
}
```

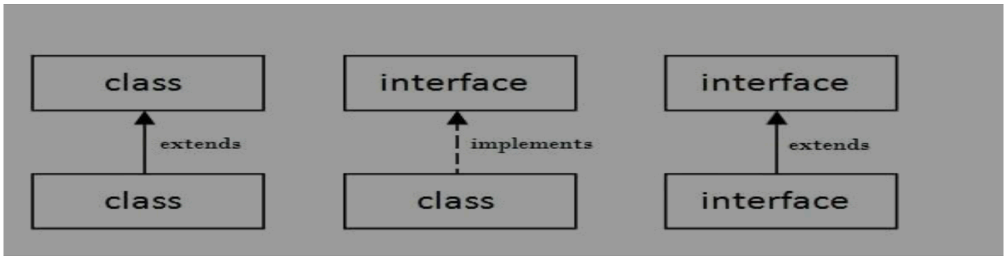
Output:

```
From FI
From SI
```

Abstract class Vs Interface

Abstract Class	Interface
can have abstract and non-abstract method	can have only abstract method
doesn't support multiple inheritance	supports multiple inheritance
can have final, non-final, static and non-static variable. (final in java = const in c++)	only have static and final variable
can provide the implementation of interface	can't provide the implementation of abstract class
can extend another Java class and implement multiple interface	can extend another Java interface only
Abstract keyword is used	Interface keyword is used
can be extended using keyword **extends**	can be implemented using keyword **implements**
can have members like private, protected etc.	public by default

Relation between class & interface


class
extends
class
interface
implements
class
interface
extends
interface