CLASS & OBJECTS

(1) Class & objects :

- Class is a blueprint which define some properties and behavior. where An object is an instance of a class which has those properties and behavior attached.
- A class is not allocated memory when it is defined. But an object is allocated memory when it is created.
- Class is a logical entity whereas objects are physical entities.
- A class is declared only once. On the other hand, we can create multiple objects of a class.
- A class is a way to arrange data and behavior information. It is a template that must be implemented by it's object.
- A class can also be seen as a user-defined data type where any object of defined data type has some predefined properties and behaviors.

Ex-

```java
package org.example.opps_concept.ClassesAndObject;

public class MainClass {
    public static void main(String[] args) {
        Dog d1 = new Dog();
        d1.name="Tommy";
        d1.bark();

        Dog d2 = new Dog();
        d2.name="Jommy";
        d2.walk();
    }
}


class Dog{

    // class properties
    String name;
    int age;


    // class behavior
    void bark(){
        System.out.println(name+" is barking");
    }
    void walk(){
        System.out.println(name+" is walking");
    }
}
```

(2) Method overloading:

- Two or more method can have same name inside the same class if provided that they have different number of arguments or different types of arguments. This feature called method overloading.
- It is not method overloading if we only change the return type of methods. There must be differences in the number of parameters.

  Ex-

```
package org.example.opps_concept.ClassesAndObject.methodOverloading;

public class MethodOverloading_2 {
    public static void main(String[] args) {
        Greet obj = new Greet();
        obj.greetings();
        obj.greetings("sahil");
    }
}

class Greet{
    void greetings(){
        System.out.println("hello,good morning");
    }

    void greetings(String name){
        System.out.println("hello,"+name+" good morning");
    }

    // only changing return type . not overloading
//    int greetings(){
//        return 9;
//    }
}
```

(3) Constructors:

- Constructors are invoked implicitly when you instantiate objects.
- The two rules for creating constructor are :

  (a) The name of the constructor should be the same name as the class.

  (b) A java constructor must not have a return type.
- If a class doesn't have a constructor , the java compiler automatically creates a default constructor during-runtime. The default constructor initializes instance variable with default values.

- Default-constructor: a constructor that is automatically created by the java compiler if it is not explicitly defined .
- A constructor can not be abstract, static or final .
- A constructor can be overloaded but can not be overridden .

  Ex-

```
package org.example.opps_concept.ClassesAndObject.constructor;

public class Constructor_3 {
    public static void main(String[] args)  {
        Complex c1 = new Complex(2,3);
        c1.printComplex();
        Complex c2 = new Complex(4,1);
        c2.printComplex();
    }
}


class Complex{
    int a;
    int b;

    // when you create obj
    // it has to go through constructor
    // so inside constructor you can
    //update class properties
    public Complex(int real,int imaganiry){
        a=real;
        b=imaganiry;
    }

    void printComplex(){
        System.out.println(a+","+b);
    }
}
```

(4) This keyword:

- In java, this keyword is used to refer to the current object inside a method or a constructor.
- We mostly use this keyword to remove any ambiguity in variable names, we can also use it to invoke methods of the current class or to invoke a constructor of the current class.

  Ex-

```
package org.example.opps_concept.ClassesAndObject.thisKeyword;

public class ThisKeyword_4 {
    public static void main(String[] args)  {
    Complex num1 = new Complex(2,3);
        System.out.println("from obj: "+num1);
        num1.printComplex();
```

```java
    }
}

class Complex {

    int a;
    int b;

    // ambiguity remove
    public Complex(int a, int b) {
        this.a=a;
        this.b=b;
    }

    void printComplex(){
        System.out.println("from this: "+this);
        this.a=47;
        this.b=45;
        System.out.println(a+","+b);
    }

}
```