OBJECT-ORIENTED PROGRAMMING (OOPS) CONCEPTS IN JAVA

Object-Oriented Programming (OOP) is a programming paradigm that organizes software design around data, or objects, rather than functions and logic.

Java is an object-oriented language that follows key OOP principles such as **Encapsulation**, **Inheritence**, **Polymorphism**, and **Abstraction**.

1. Principles of OOP in Java:

a) Encapsulation :

Encapsulation is the concept of wrapping the data (variables) and methods (functions) that operate on the data into a single unit called a **class**.

The class's data is hidden from direct access by other classes and is only accessible through public getter / setter methods.

Example :

```
class Person {
    private String name;      // private variable
                                (encapsulation)
    // Public getter method for accessing private variable
public String getName () {
    return name;
}

// Public setter method for modifying private variable
public void setName (string name) {
    this.name = name;
}
}
```

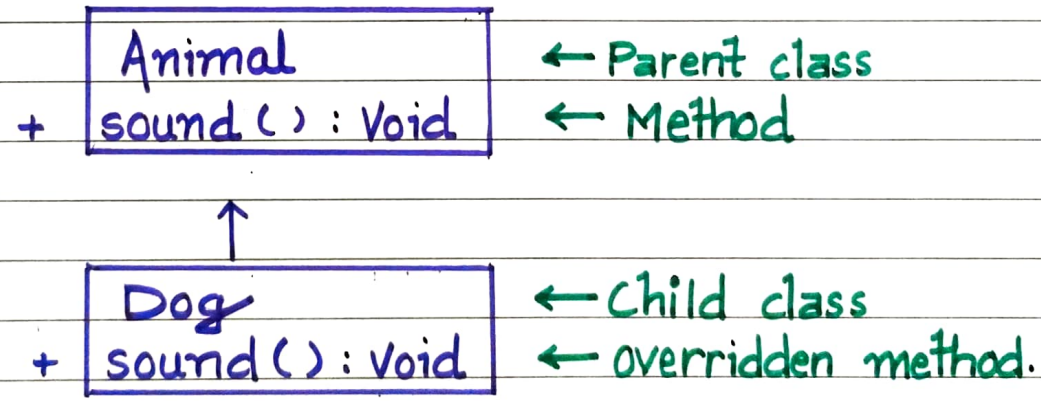
b) Inheritence :

Inheritence allows one class (child class) to inherit the properties and behaviors of another class (parent class). This promotes code reuse and the creation of a hierarchial class structure.

Example :

```
class Animal {
    void sound() {
        system.out.println ("Animal makes a sound");
    }
}

Class Dog extends Animal {
    void sound () {
        System.out.println ("Dog barks");
    }
}
```



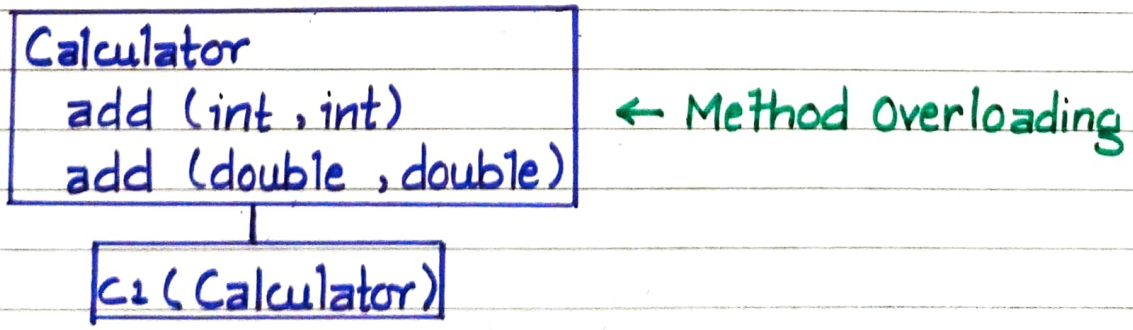
@Curious_.programmer

c) Polymorphism :

Polymorphism allows methods to behave differently depending on the object that invokes them.
This can be achieved through Method Overloading (compile-time) and Method Overriding. (runtime polymorphism).

Example : Method Overloading :

```
class Calculator {
    int add (int a , int b) {
        return a+b ;
    }
// Method overloading : same method name with
  different parameters.
double add (double a , double b) {
    return a + b ;
  }
}
```

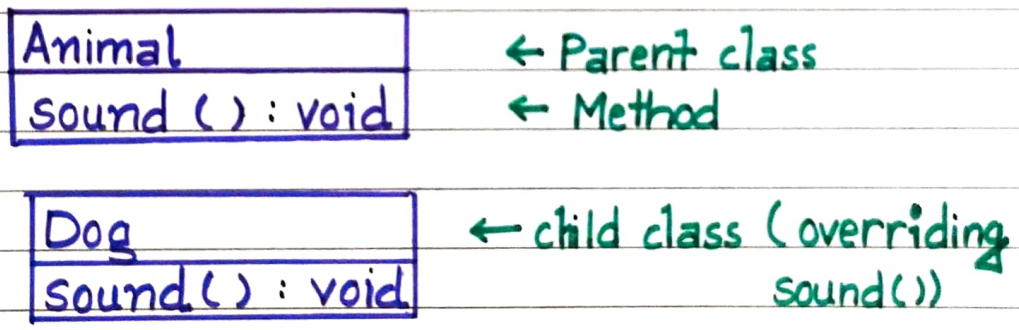


@Curious_.programmer

Example : Method Overriding :

```
class Animal {
    void sound() {
        System.out.println("Animal makes a sound");
    }
}

Class Dog Extends Animal {
    @Override
    void sound() {
        System.out.println("Dog barks");
    }
}
```



@Curious_programmer

d) Abstraction:

Abstraction is the concept of hiding the implementation details and exposing only the essential features of an object. This can be achieved using abstract classes or interfaces.

Example with Abstract class:

```
abstract class Shape {
    abstract void draw(); // Abstract method

  void color() {
    system.out.println("Coloring the shape");
  }
}

Class Circle extends Shape {
   @Override
   void draw() {
     System.out.println("Drawing circle");
  }
}
```

2. Classes and Objects:

- **Class**: A blueprint for creating objects, defining properties (attributes) and behaviours (methods).

- **Object**: An instance of a class that holds data and can perform actions.

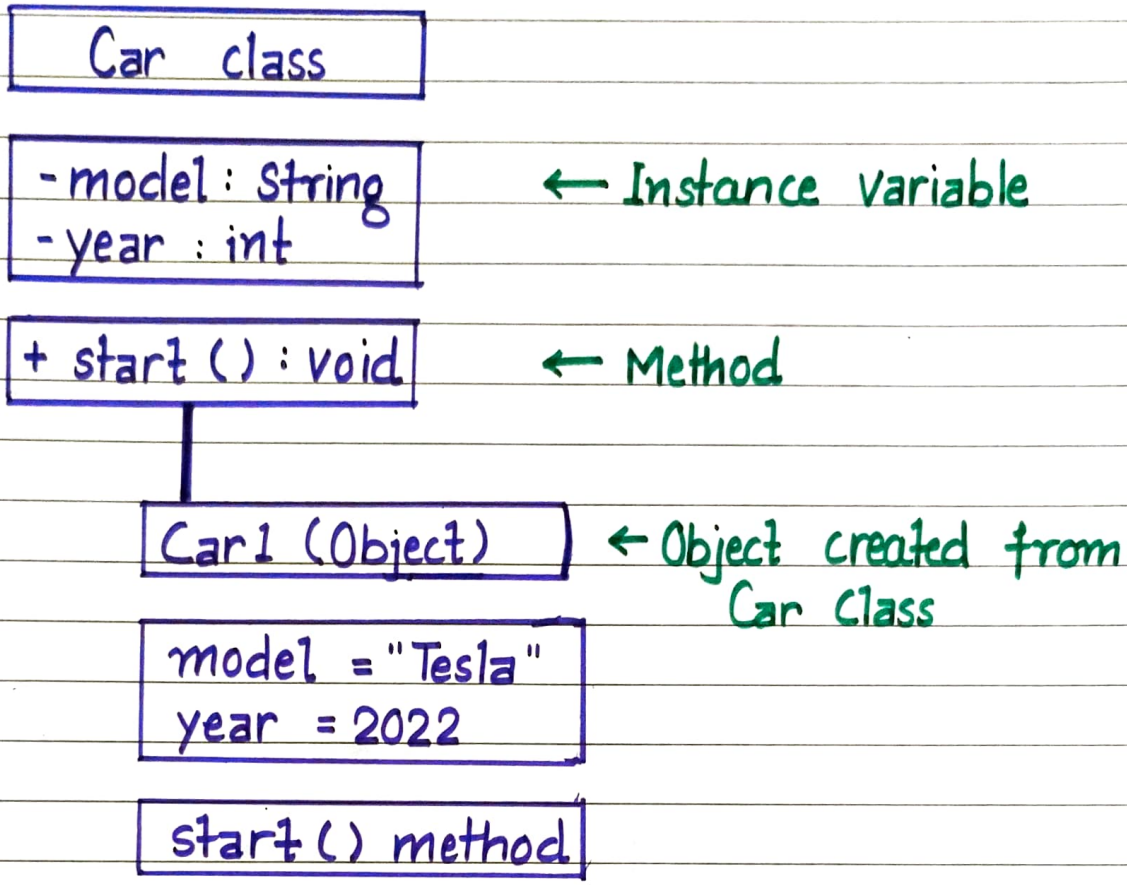
Example:

```
class Car {
    String model;
    int year;

  void start() {
     System.out.println("Car is starting");
   }
}
```



```
public class Main {
   public static void main(String[] args) {
       Car car1 = new Car(); //creating an object
       car1.model = "Tesla";
       car1.year = 2022;
       car1.start();    //calling the method
   }
}
```



3. Methods and Constructors:

- **Method:** A block of code that performs a specific task and can return a value.

- **Constructor:** A special method used to initialize objects. It is called when an object is created and does not have a return type.

Example of Constructor :

```
class Person {
    String name;
    int age;

  // constructor
  Person (String name, int age) {
    this.name = name;
    this.age = age;
  }
}

public class Main {
    public static void main (String[] args) {
      Person p1 = new Person ("John", 25);
    }                          //constructor called
}
```

4. this Keyword:

The this keyword refers to the current object instance. It is used to differentiate between instance variables and method parameters when they have the same name.

Example:

```
class Car {
    String model;
// constructor using 'this' to differentiate between
   instance variable and parameter

Car (String model) {
    this.model = model;
        //'this.model' refers to instance variable,
            'model' is the parameter.
  }
}
```

5. Static and Instance Members:

- **Instance Members:** Belong to an object and are created when the object is instantiated.

- **Static Members:** Belong to the class itself and are shared across all objects.

Example:

```
class Counter {
    int instanceCount = 0; // Instance variable
    static int staticCount = 0; // static variable

 // Method to increment counts
    void increment () {
        instantceCount ++ ;
        staticCount ++ ;
    }
}
```

@ Curious_.programmer

```
public class Main {
  public static void main (String [] args) {
    Counter c1 = new Counter ();
    Counter c2 = new Counter ();

    c1.increment ();
    c2.increment ();
```

```
        System.out.println ("Instance Count for c1 : " +
                            c1.instanceCount ); // 1
        System.out.println ("Instance Count for c2 : "+
                            c2.instanceCount ); // 1
        System.out.println ("Static Count (shared across
                            instances ): " + Counter.StaticCount); //2
    }
}
```

@Curious_programmer