What is class in java

A class is a blueprint for creating objects. It defines a type by bundling data and methods that work on the data into a single unit.

Create a Class

public class Main {

int x = 5;

}

**Create an Object**

An object is created from a class. We have already created the class named Main, so now we can use this to create objects.

Example

Create an object called "myObj" and print the value of x:

public class Main {

int x = 5;

public static void main(String[] args) {

Main myObj = new Main();

System.out.println(myObj.x);

}

}

**Multiple Objects**

public class Main {

int x = 5;

public static void main(String[] args) {

Main myObj1 = new Main(); // Object 1

Main myObj2 = new Main(); // Object 2

System.out.println(myObj1.x);

System.out.println(myObj2.x);

}

}

**Using Multiple Classes**

You can also create an object of a class and access it in another class.

Main.java

Second.java

public class Main {

int x = 5;

}

class Second {

public static void main(String[] args) {

Main myObj = new Main();

System.out.println(myObj.x);

}

}

## Java Constructors

A constructor in Java is a special 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:

Example

// Create a Main class

public class Main {

int x; // Create a class attribute

// Create a class constructor for the Main class

public Main() {

x = 5; // Set the initial value for the class attribute x

}

public static void main(String[] args) {

Main myObj = new Main(); // Create an object of class Main (This will call the constructor)

System.out.println(myObj.x); // Print the value of x

}

}

**Constructor Parameters**

public class Main {

int x;

public Main(int y) {

x = y;

}

public static void main(String[] args) {

Main myObj = new Main(5);

System.out.println(myObj.x);

}

}

**Another Example**

public class Main {

int modelYear;

String modelName;

public Main(int year, String name) {

modelYear = year;

modelName = name;

}

public static void main(String[] args) {

Main myCar = new Main(1969, "Mustang");

System.out.println(myCar.modelYear + " " + myCar.modelName);

}

}

**Java Inheritance (Subclass and Superclass)**

It is possible to inherit attributes and methods from one class to another. We group the "inheritance concept" into two categories:

subclass (child) - the class that inherits from another class

superclass (parent) - the class being inherited from

To inherit from a class, use the extends keyword.

**Example**

class Vehicle {

protected String brand = "Ford"; // Vehicle attribute

public void honk() { // Vehicle method

System.out.println("Tuut, tuut!");

}

}

class Car extends Vehicle {

private String modelName = "Mustang"; // Car attribute

public static void main(String[] args) {

// Create a myCar object

Car myCar = new Car();

// Call the honk() method (from the Vehicle class) on the myCar object

myCar.honk();

// Display the value of the brand attribute (from the Vehicle class) and the value of the modelName from the Car class

System.out.println(myCar.brand + " " + myCar.modelName);

}

}

**Java Polymorphism**

Polymorphism means "many forms", and it occurs when we have many classes that are related to each other by inheritance.

Example

class Animal {

public void animalSound() {

System.out.println("The animal makes a sound");

}

}

class Pig extends Animal {

public void animalSound() {

System.out.println("The pig says: wee wee");

}

}

class Dog extends Animal {

public void animalSound() {

System.out.println("The dog says: bow wow");

}

}

class Main {

public static void main(String[] args) {

Animal myAnimal = new Animal(); // Create a Animal object

Animal myPig = new Pig(); // Create a Pig object

Animal myDog = new Dog(); // Create a Dog object

myAnimal.animalSound();

myPig.animalSound();

myDog.animalSound();

}

}