**JAVA CLASSES — HANDS-ON NOTES**

**What is a Class?**

A **class** is a *blueprint* or *template* from which objects are created.  
It groups **variables (fields)** and **methods (functions)** together.

Example:

// A simple class

class Car {

// Fields / Properties

String brand;

int speed;

// Method / Behavior

void drive() {

System.out.println(brand + " is driving at " + speed + " km/h");

}

}

**What is an Object?**

An **object** is a *real instance* of a class — it has its own values.

Example:

public class Main {

public static void main(String[] args) {

// Creating objects

Car car1 = new Car();

car1.brand = "Tesla";

car1.speed = 120;

Car car2 = new Car();

car2.brand = "BMW";

car2.speed = 100;

// Using the method

car1.drive();

car2.drive();

}

}

Output:

Tesla is driving at 120 km/h

BMW is driving at 100 km/h

**Constructor**

A **constructor** initializes objects.  
It has the same name as the class and **no return type**. Example:

class Student {

String name;

int age;

// Constructor

Student(String n, int a) {

name = n;

age = a;

}

void show() {

System.out.println(name + " is " + age + " years old");

}

}

public class Main {

public static void main(String[] args) {

Student s1 = new Student("Ravi", 21);

Student s2 = new Student("Meena", 20);

s1.show();

s2.show();

}

}

Output:

Ravi is 21 years old

Meena is 20 years old

**INHERITANCE — HANDS-ON NOTES**

**What is Inheritance?**

Inheritance allows a class to **reuse** another class’s properties and methods.

**Syntax:**

class ChildClass extends ParentClass

Think of it as:  
*Child inherits traits from Parent.*

**Example 1: Single Inheritance**

class Animal {

void eat() {

System.out.println("Animal eats food");

}

}

class Dog extends Animal {

void bark() {

System.out.println("Dog barks");

}

}

public class Main {

public static void main(String[] args) {

Dog d = new Dog();

d.eat(); // Inherited method

d.bark(); // Own method

}

}

Output:

Animal eats food

Dog barks

**Example 2: Multilevel Inheritance**

class Animal {

void eat() {

System.out.println("Eating...");

}

}

class Dog extends Animal {

void bark() {

System.out.println("Barking...");

}

}

class Puppy extends Dog {

void weep() {

System.out.println("Weeping...");

}

}

public class Main {

public static void main(String[] args) {

Puppy p = new Puppy();

p.eat(); // from Animal

p.bark(); // from Dog

p.weep(); // from Puppy

}

}

Output:

Eating...

Barking...

Weeping...

**Example 3: Hierarchical Inheritance**

class Animal {

void eat() {

System.out.println("Eating...");

}

}

class Dog extends Animal {

void bark() {

System.out.println("Barking...");

}

}

class Cat extends Animal {

void meow() {

System.out.println("Meowing...");

}

}

public class Main {

public static void main(String[] args) {

Dog d = new Dog();

Cat c = new Cat();

d.eat(); d.bark();

c.eat(); c.meow();

}

}

**super Keyword**

Used to:

1. Access parent class method or variable
2. Call parent constructor

👉 Example:

class Animal {

Animal() {

System.out.println("Animal created");

}

void sound() {

System.out.println("Animal sound");

}

}

class Dog extends Animal {

Dog() {

super(); // Calls Animal constructor

System.out.println("Dog created");

}

void sound() {

super.sound(); // Calls Animal method

System.out.println("Dog barks");

}

}

public class Main {

public static void main(String[] args) {

Dog d = new Dog();

d.sound();

}

}

🧠 Output:

Animal created

Dog created

Animal sound

Dog barks