**What is class**

In Java, **a class** is like a blueprint or template for creating objects (instances). It defines the structure and behavior that the objects of that class will have. A class encapsulates data for the object (through variables) and methods (functions) that operate on the data.

**Some analogy to understand the class and object concept clearly.**

|  |  |
| --- | --- |
| Analogy | Concept |
| Cookie and cutter analogy. | You can think of a class like a **cookie cutter**, and an **object** as the **cookie** that’s made from that cutter. The class defines how the cookie should look, and the object is the actual cookie created from it. |
| Blueprint and House | You can think of a class as a blueprint for a house, and an object as the actual house built from that blueprint. The blueprint (class) contains all the details about how the house should be designed (such as the number of rooms, windows, doors, etc.), but the house (object) is the real, tangible instance built based on that blueprint. |
| Recipe and Dish | A class is like a recipe, and an object is the dish prepared from that recipe. The recipe (class) lists all the ingredients and steps needed to make the dish, but the dish (object) is the final product you can actually eat, made by following the recipe. |
| Car Design and Car | Think of a class as the design of a car, and an object as the actual car that gets built. The design (class) specifies how the car should look, its parts, and its features, while the car (object) is the real, physical vehicle created using that design. |
| Library and Book | A class is like a library that holds many books, and an object is like an individual book from that library. The library (class) contains a collection of knowledge, rules, and guidelines, while a book (object) is a specific instance of that knowledge or story, with content you can interact with. |
| Factory and Product | A class is like a factory that produces items, and an object is the individual product created by the factory. The factory (class) has the machines, workers, and processes that define how items are made, while the product (object) is the actual item that comes off the assembly line. |

**An simple structure of an class**

class\_keyword class\_name {

// body of class

}

**And that’s how a actual class look like.**

public class Car {

// Fields (attributes)

String color;

String model;

// Constructor (initializes the object)

public Car(String color, String model) {

this.color = color;

this.model = model;

}

// Method (behavior)

public void displayDetails() {

System.out.println("Car Model: " + model + ", Color: " + color);

}

}

|  |  |  |  |
| --- | --- | --- | --- |
| Main Part of Class | | Working | |
| Fields (Variables): | | These store the state or attributes of the object. They define the properties or characteristics the object will have. | |
| Methods: | | These define the behavior or functionality of the objects. Methods represent the actions the object can perform. | |
| Constructors: | | Special methods used to initialize new objects. Constructors are called when you create a new instance of a class, and they typically assign initial values to the instance variables. | |
| Fields (variable) | | **Method** | **Constructor** |
| Object | **Variables** | **Action type** | **Value** |
| Car | * color * model * engine\_type | * startEngine() * accelerate() * brake() | String color = “blue”  String model = “Toyota Camry 2024”  String engine\_type = “3.5-liter V-6 engine” |
| Perfume | * fragrance * thickness * volume | * spray() * changeFragrance() | String fragrance = “Lavender”  String thickness = “Liquid”  String volume = “50.0 (millilitres)” |
| Ball | * weight * size * material | * bounce() * roll() * inflate() | String weight = “0.5 (kg)”  String size = “7.5 (inches)”  String material = “Rubber” |
| Library Book | * title * author * isbn * year\_published | * open() * read() * close() * borrow() | String title =”The Great Gatsby”  String author = “F. Scoot Fitzgerald”  long isbn = “9780743273565”  short Year\_Published = “1925” |
| Smartphone | * brand * model * batteryl\_life * screen\_size | * makeCall() * sendText() * takePicture() * unlockPhone() | String brand = “Apple”  String model = “iPhone 16 pro”  short battery\_life = “22.7” hr  float screen\_size = “6.1” inch |
| Laptop | * brand * processor\_type * ram\_size * storage\_size | * powerOn() * runProgram() * restart() * powerOff() * hibernate() * sleep() | String brand = “Dell”  String processor\_type = “Intel i-7”  int ram\_size = “32” GB  String storage\_size = “1TB” |

**All Rules of defining a class**

|  |  |  |
| --- | --- | --- |
| No | Area | Rules |
| 1 | **Class Declaration** | 1. A class is declared using the class keyword followed by the class name. 2. By convention, class names start with a capital letter. 3. A class is just a schema or a blueprint which defines how and object is created. 4. A class is not occupy any memory in ram because its an schema not a real world property. |
| 2 | **Access Modifiers** | 1. Classes can have **access modifiers** like public, private, or protected. 2. A class must be declared **public** if it’s in a separate file. 3. Default access (no modifier) means the class is accessible only within its own package. |
| 3 | **Instance Variables (Fields)** | 1. Variables inside a class are called **fields** or **instance variables**. 2. Fields represent the state of the object. 3. They can be of any data type: primitive types (like int, float, etc.) or objects. |
| 4 | **Constructors** | 1. A constructor is a special method that is called when an object is created. 2. A constructor has the same name as the class and no return type (not even void). 3. If no constructor is defined, Java provides a **default constructor** (no arguments, does nothing). The reason is being to initialized all the instance variable to its default value. |
| 5 | **Methods** | 1. Methods define the **behavior** of the class. 2. A method is defined by specifying a return type, method name, and parameters. 3. Methods can return values, or they can be void (no return value). |
| 6 | **Objects (Instances of a Class)** | 1. Once a class is defined, you can create objects (instances) of the class using the new keyword. 2. new keyword is also as an operator which is known as (memory allocation operator) used for allocating memory in heap and also return the reference (memory address) in a object variable. 3. Objects hold their own state based on the class's fields. |
| 7 | **Class Names and Object Names** | 1. Class names are typically written in **PascalCase** (each word starts with an uppercase letter), e.g., Car, Perfume, Ball, Laptop, SmartPhone. 2. Object names are typically written in **camelCase** (first word lowercase, subsequent words capitalized), e.g., myCar, myPerfume, myBall, myLaptop, mySmartPhone. |
| 8 | **Static Members** | 1. A static variable or method belongs to the class itself, not to instances(object) of the class. 2. They are shared by all instances of the class. 3. They are initialized only once when the class firsts load. 4. A static variable can be changed, and the new value will be reflected across all instances. 5. Static methods can be called without creating an object. |
| 9 | **Inheritance** | 1. A class can inherit from another class using the extends keyword. This allows the subclass to inherit fields and methods from the superclass. 2. A subclass can override methods of the superclass. 3. In java we can only extends one class at a time because multiple and hybrid inheritance is not allowed in java via extends keywords or inheritance. 4. If you want to implement multiple inheritance you have to use interface. |
| 10 | **Encapsulation** | 1. Java encourages **encapsulation**, which is the practice of keeping fields private and providing **public getter and setter methods** to access or modify those fields. 2. This hides the internal state of the object and protects it from unwanted modifications. |
| 11 | **Abstract Classes** | 1. An **abstract class** cannot be instantiated directly. It is meant to be subclassed. 2. Abstract classes can have both abstract methods (without a body) and concrete methods (with a body). |
| 12 | **Interfaces** | 1. An **interface** is similar to an abstract class but it only contains abstract methods (no method body). 2. A class can implement multiple interfaces (unlike inheritance, where only one class can be extended). 3. When implementing an interfaces in class we must have to use implements keyword after class name and then interface name. 4. If we have to implement multiple interface in same class so we use (,) comma separator. |