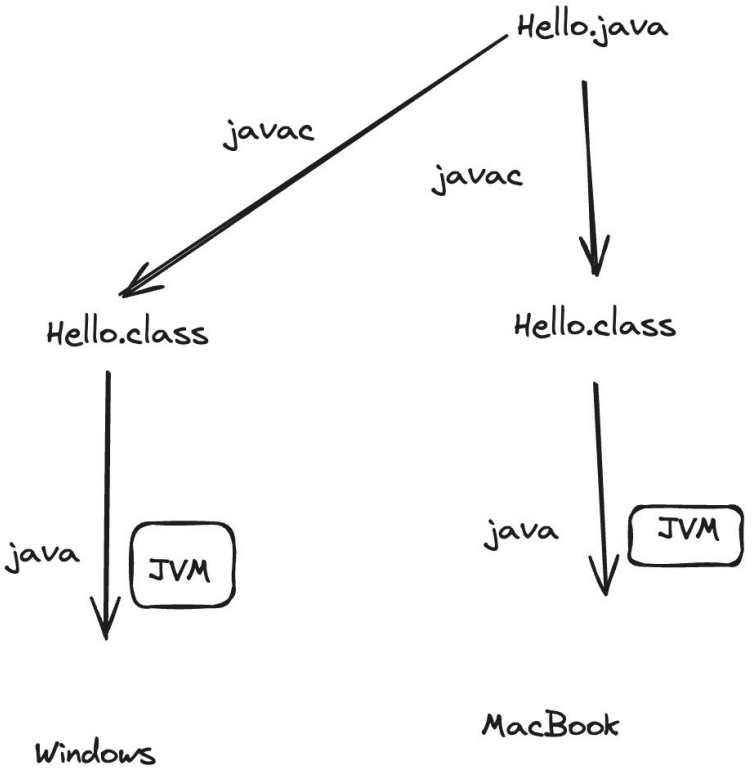
Java Backend Development Live-85

lecture-1

Class Agenda

- System setup (IDE and JDK)
- Introduction to java: java, javac, bytecode, platform independent.
- Project setup and execution of java program
- Java OOPs Concepts: Object, Class, Encapsulation, Inheritance
- Inheritance vs Composition

Execution of Java Code



Java
:Platform Independent

Class and Object

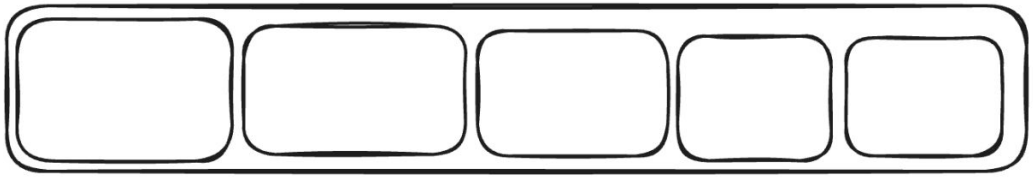
- Class − A class can be defined as a template/blueprint that describes the behavior/state of the object of its type
- Object − Objects have states and behaviors.
- No argument Constructors
- Parameterized Constructors
- Local variables
- Instance variables
- Class variables

Classes & Objects in Real World

Class:

Design of 1000sqft House.

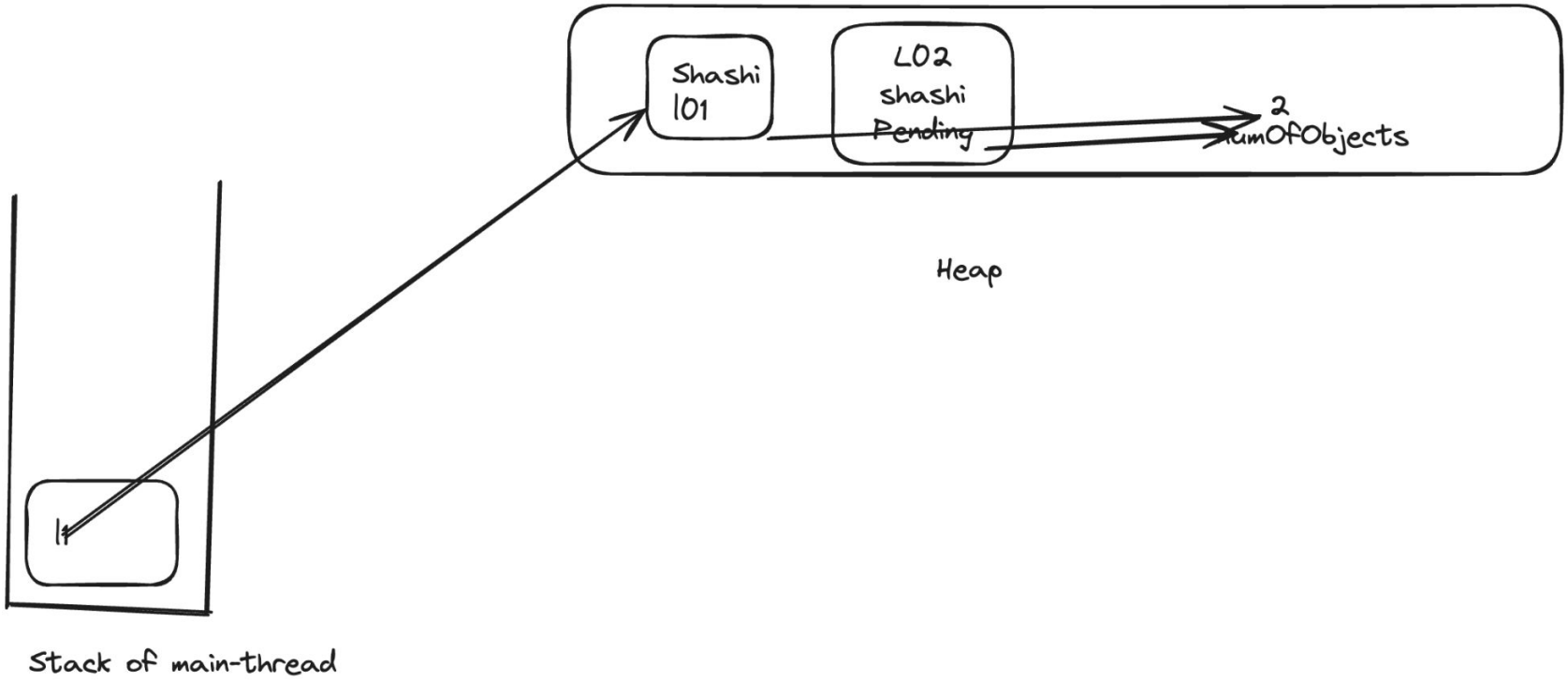
Heap
5000 sqft



HeapOutOfMemory

Design of Apple iPhone-13

Objects Creation in Java



Encapsulation & Data Hiding

An object stores its state in **fields** and exposes its behavior through **methods.**

Methods operates on object's internal state and serve as the primary mechanism for object-to-object communication.

Hiding internal state (from the outside classes) and requiring all interactions to be performed through an object's publicly exposed methods is known as encapsulation.

It is to make sure **sensitive** data is hidden from users.

How to achieve encapsulation?

By using “access modifiers”, as

Declare class **variables**/**attributes** as private

Define public **getter** and **setter** methods to access and update the value of private variable.

Access Modifiers

Access modifiers for classes, attributes, methods and constructors.

Modifier	Description
public	The code is accessible for all classes
private	The code is only accessible within the declared class
default	The code is only accessible in the same package.
protected	The code is accessible in the same package and subclasses in any outside package.

Access Levels:

Access Levels				
Modifier	Class	Package	Subclass	World
public	Y	Y	Y	Y
protected	Y	Y	Y	N
no modifier	Y	Y	N	N
private	Y	N	N	N

Inheritance

A ways to organise interrelated classes i.e how a subclass can inherit fields and methods from a superclass, thereby making a hierarchy.

SuperClass: The class whose features are inherited is known as superclass(or a base class or a parent class).

SubClass: The class that inherits the other class is known as subclass(or derived class, extended class, or child class). The subclass can add its own fields and methods in addition to the superclass fields and methods.

Common behavior can be defined in superclass and inherited into subclass using the **extends** keyword.

Benefit of Inheritance

Reusability:

Inheritance supports the concept of reusability, i.e when we want to create a new class and there is already existing class that includes some of the code that we want, we can drive our new class from the existing class. By doing this, we are reusing the fields and methods of the existing class.

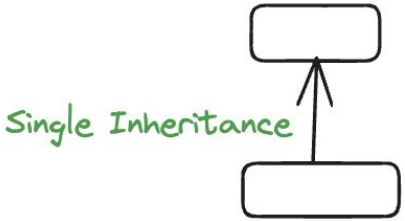
@Override instructs the compiler that you intend to override a method in the superclass.

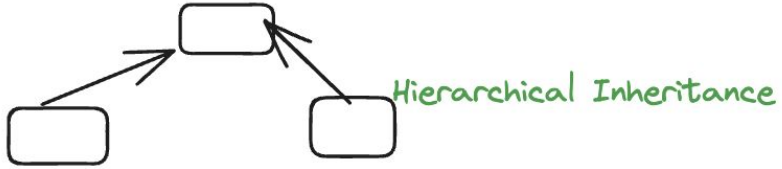
Types of inheritance

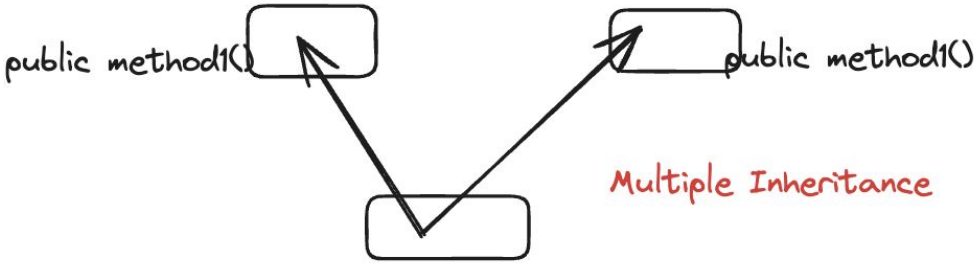
- Single Inheritance
- Multilevel Inheritance
- Hierarchical Inheritance
- Multiple Inheritance
- Hybrid Inheritance

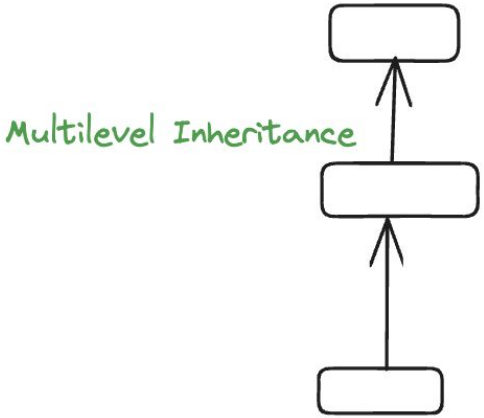
Java does not allow multiple and hybrid inheritance, while using classes.

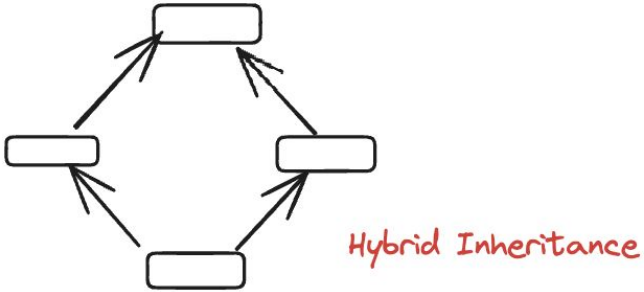
However, we may use multiple and hybrid Inheritance by using interface..


Single Inheritance


Hierarchical Inheritance


public method1()
public method1()
Multiple Inheritance


Multilevel Inheritance


Hybrid Inheritance

Creating object of child class

Super keyword

A subclass inherits all of the public and protected members of its parent, no matter what package the subclass is in.

Casting objects

- Subclass to superclass
- Superclass to subclass

Inheritance vs. Composition

Inheritance is an **“is-a”** relationship.

- Laptop is a computer.
- Car is a vehicle

Composition is a “has-a” relationship.

- Car has a Engine.
- Laptop has a OS.

We do composition by having an instance of another class as a field of our class, instead of extending it.

Interfaces in Java

Interface specify what a class must do. It does not specifies “How” to do it.

- Methods signature by default public
- Abstract method, default methods, and static methods

When a class implements an interface, it promises to provide the behavior published by that interface OR the class must declared as abstract.

Interface can extend multiple interfaces.

public interface A extends B, C {}

A class can implement multiple interfaces.

Multiple Inheritance Problem

Diamond Problem (Ambiguity)

Java forces the class implementing multiple interfaces to provide its own implementation to remove ambiguity.

Interfaces are used to achieve total abstraction and loose coupling.