## 💡 **Java me Variable Kitne Type ke Hote hain?**

Java me mainly **3 types ke variables** hote hain:

| **Type** | **Kahan Use Hota Hai** | **Memory Lifetime** |
| --- | --- | --- |
| 1. Local Variable | Method ke andar | Method khatam to memory free |
| 2. Instance Variable | Class ke andar, lekin method ke bahar | Object ke saath survive karta hai |
| 3. Static Variable | Class ke andar, static keyword ke saath | Class ke saath ek hi baar banta hai |

### ✅ 1. **Local Variable**

📌 Ye sirf method/function ke andar hi kaam karta hai.  
🔒 Iska scope sirf us block/method tak hota hai.

**class Test {**

**void show() {**

**int x = 10; // local variable**

**System.out.println("x = " + x);**

**}**

**}**

### ✅ 2. **Instance Variable**

📌 Ye variable class ke andar hota hai (lekin kisi method ke bahar).  
🔗 Har object ka apna alag instance hota hai.

**class Student {**

**String name; // instance variable**

**int age; // instance variable**

**void show() {**

**System.out.println("Name: " + name + ", Age: " + age);**

**}**

**}**

### ✅ 3. **Static Variable**

📌 static keyword ke saath use hota hai.  
🌐 Ek hi copy poori class ke liye hoti hai (common for all objects).

class Student {

String name;

int age;

static String college = "ABC College"; // static variable

void show() {

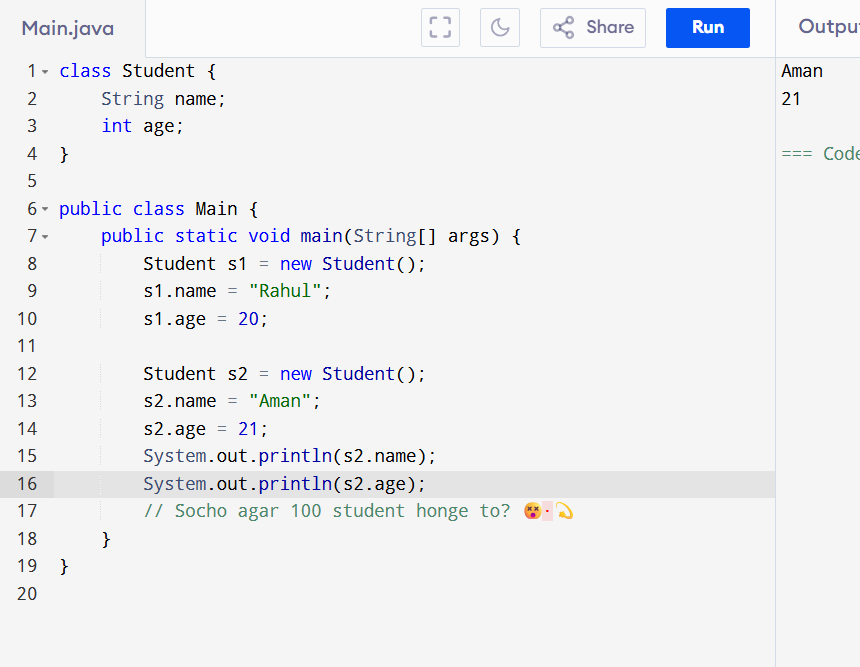
System.out.println(name + " | " + age + " | " + college);

}

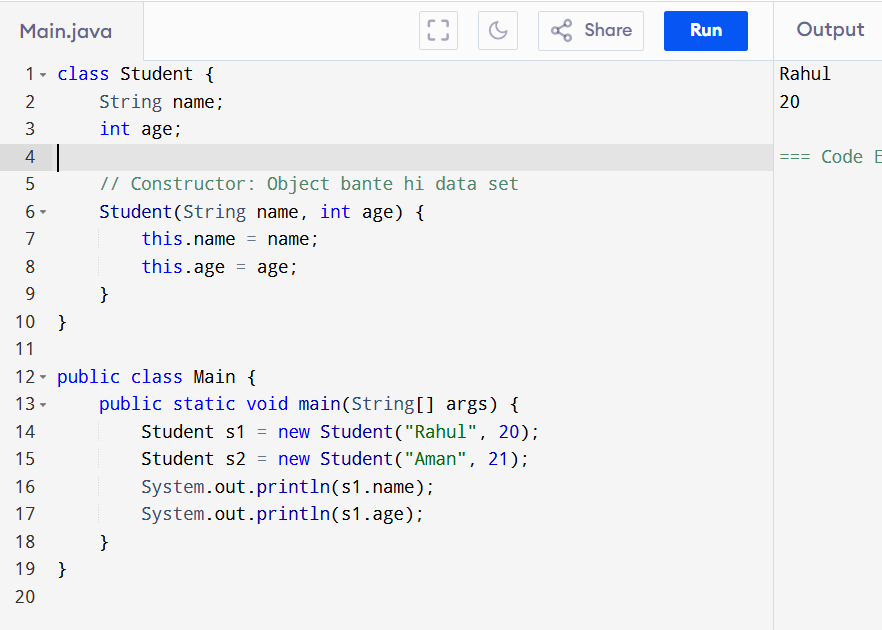
}

**🔧 Situation: pass 100 students data hai**

**🔴 1. Bina Constructor ke kaise karega? (Manually har baar set karega)**

****

**🡪Constructor**

****

**Bina constructor:** Object banta hai… phir Ham manually data dete hai .  
**Constructor ke saath:** Object bante hi data bhi set ho jaata hai ✅ .

****

**Defination :🡪**A constructor is a special method in Java that is used to initialize objects.  
It has the same name as the class and does **not have any return type.  
  
:::: 🡪**constructor ek special function hota hai jo object banate hi automatic chal jaata hai, aur wo object ke variables ko set (initialize) karta hai. Iska naam class ke naam jaisa hota hai aur koi return type nahi hoti.

**Toh Constructor Ki Zarurat Kyu Padti Hai?**

**1. Automatic Initialization — Object banate hi variables set karne ke liye**

**2. Code Saaf aur Maintain Karna Aasan Hotaa Hai**

Jab tum bahut saare objects banao (jaise 100 students), toh har baar manually properties set karna boring aur error-prone ho sakta hai.

Constructor se:

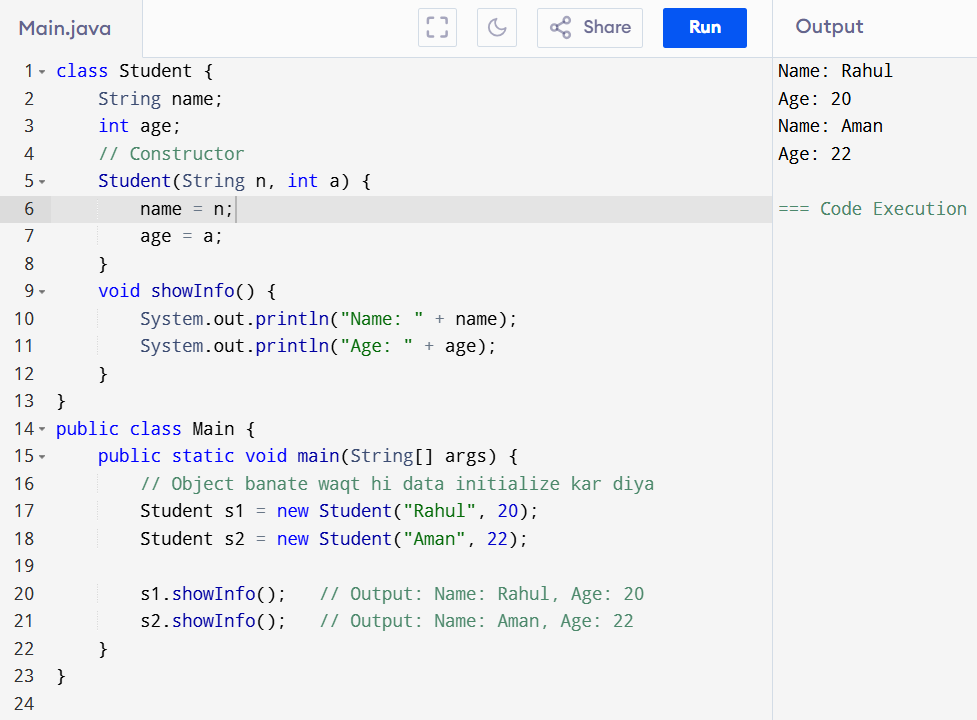
Student s1 = new Student("Rahul");

Student s2 = new Student("Aman");

Student s3 = new Student("Priya");

Direct, fast, aur simple!

: initial Object   
  
Without initialization: 🡪  
  
With Constructor (Initialization):



🡪##



**❌ Problem:**

* Agar s1.name set karna bhool gaye to object **incomplete** rahega.
* Har object ke liye alag se .name aur .age set karna padta hai.

✅ 2. **Constructor ke saath:**

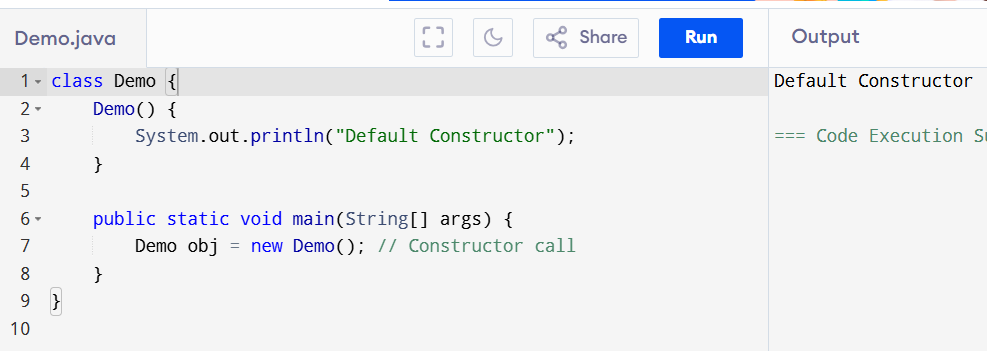


🡪✅ — Java mein mainly  **3 type constructor hote hain**:

1. **Default Constructor** – bina argument ke
2. **Parameterized Constructor** – argument ke sath
3. **Copy Constructor** – ek object se dusra object banana

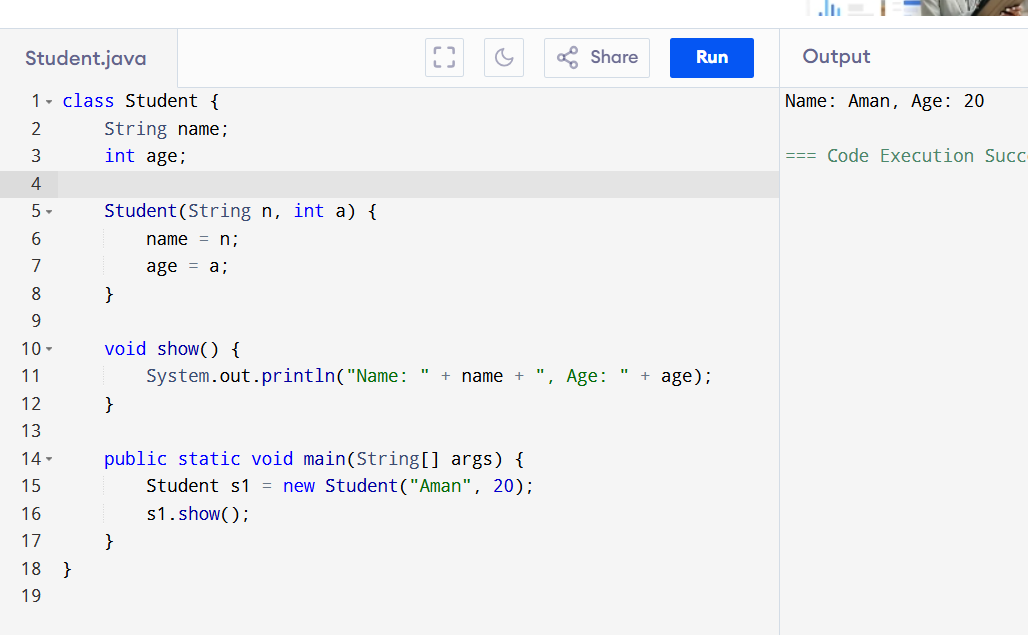
 1: **Bina kisi parameter ke hota hai**

 Agar hum khud na banayein, to Java compiler automatic ek default constructor bana deta hai



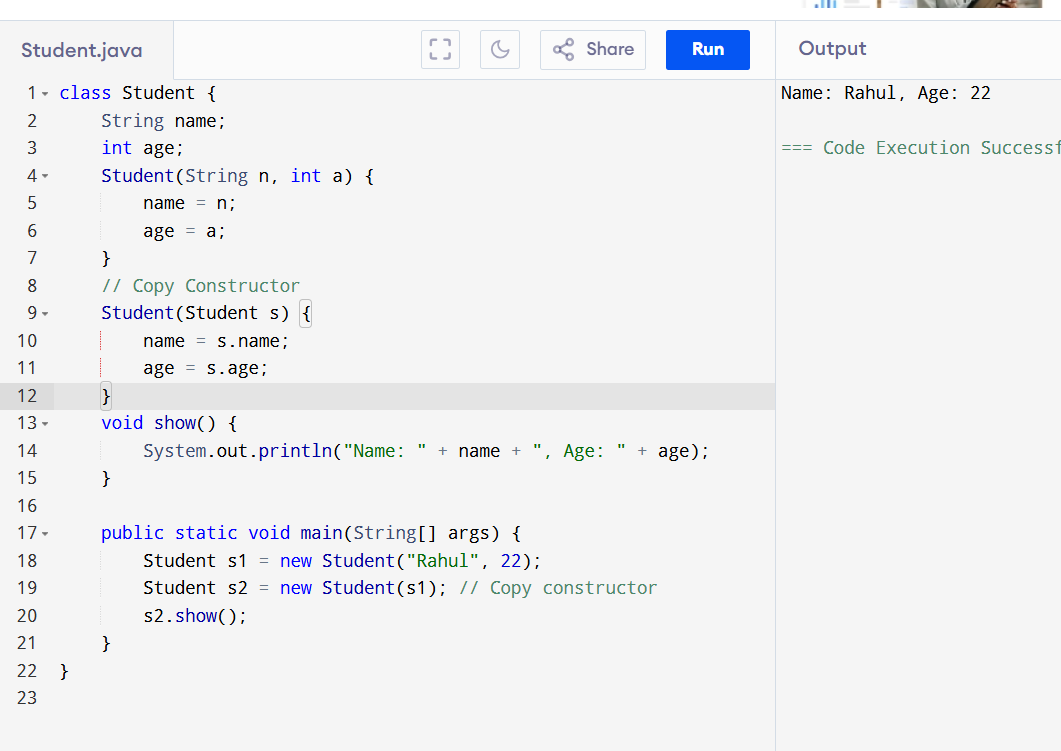
## 2. **Parameterized Constructor**

* **Isme parameters diye jaate hain** taaki object banate waqt values set ki jaa sakein.



## 🔷 3. **Copy Constructor (Manual Banate Hai)**

* **Ek object ki value dusre object me copy karta hai**
* Java me yeh automatic nahi hota, hume khud define karna padta hai.



## 🤔 ***Copy Constructor Kya Hai?***

🡪Copy constructor ka kaam hota hai:

🡪Ek object ki **sabhi values** ko copy karke **naya object** banana.

### **⚙️ Example: Real Life Socho**

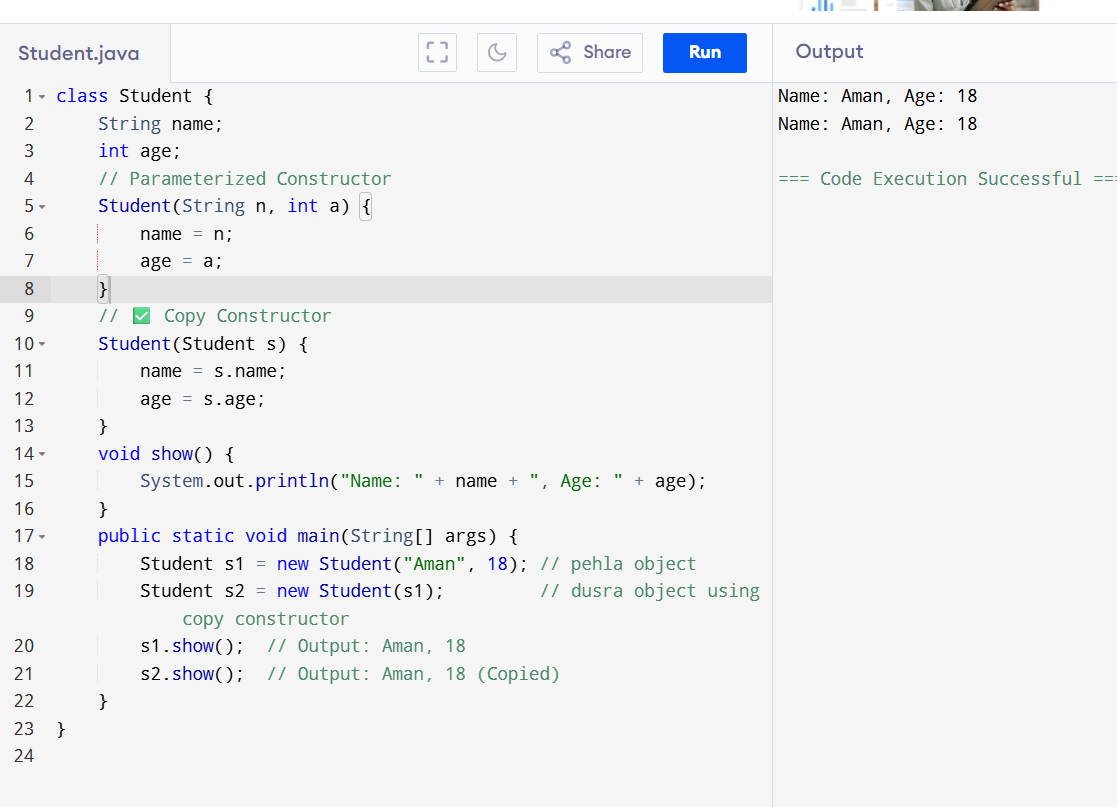
Tum ek "Student Form" bharte ho:

* Naam: Aman
* Age: 18

Ab teacher kehta hai:

"Ek aur student ka form banao, same details ke saath."

Toh tum **copy** kar doge — **same name aur age**.



🡺  
**Class and Object**

### \*Class ki Definition:

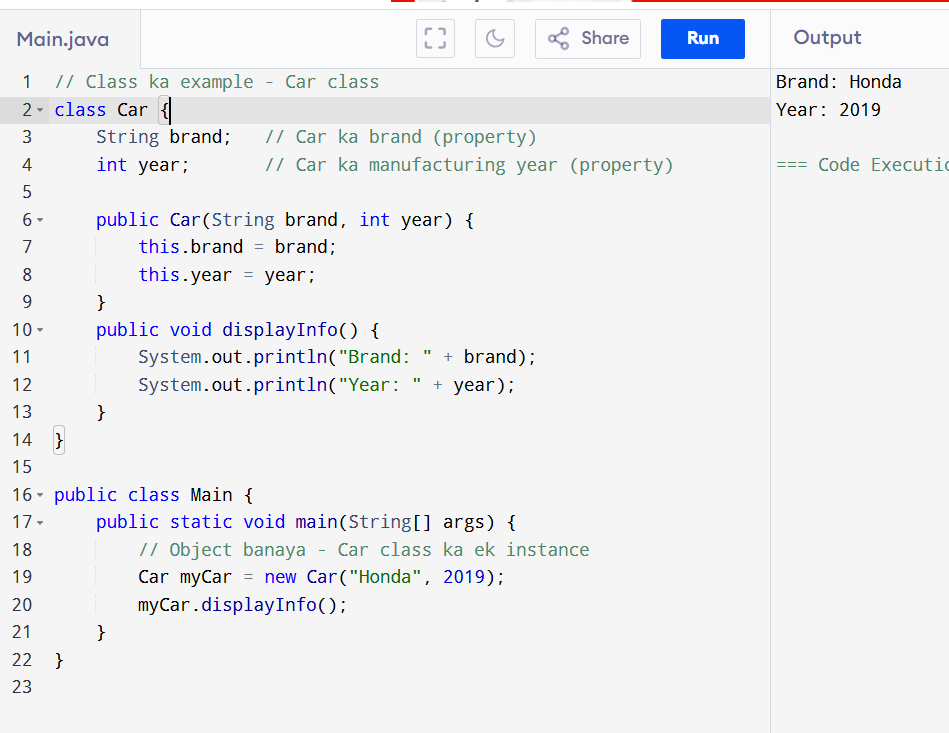
**Class** ek blueprint ya template hoti hai jisme variables (data) aur methods (functions) define kiye jaate hain. Iske through hum real-world entities ko programming mein represent karte hain.

**Simple shabdon mein:**  
Class ek design hota hai jisse hum ek object banate hain.

### **\*Object ki Definition:**

**Object** class ka ek real instance (actual copy) hota hai jo memory mein allocate hota hai. Object ke paas apni properties (data) hoti hain aur wo class ke methods ko use kar sakta hai.

**Simple shabdon mein:**  
Object class ka ek bana hua example hai jo hum program mein use karte hain.



# **Function aur Method: Definition & Difference**

### 1. **Function** kya hai?

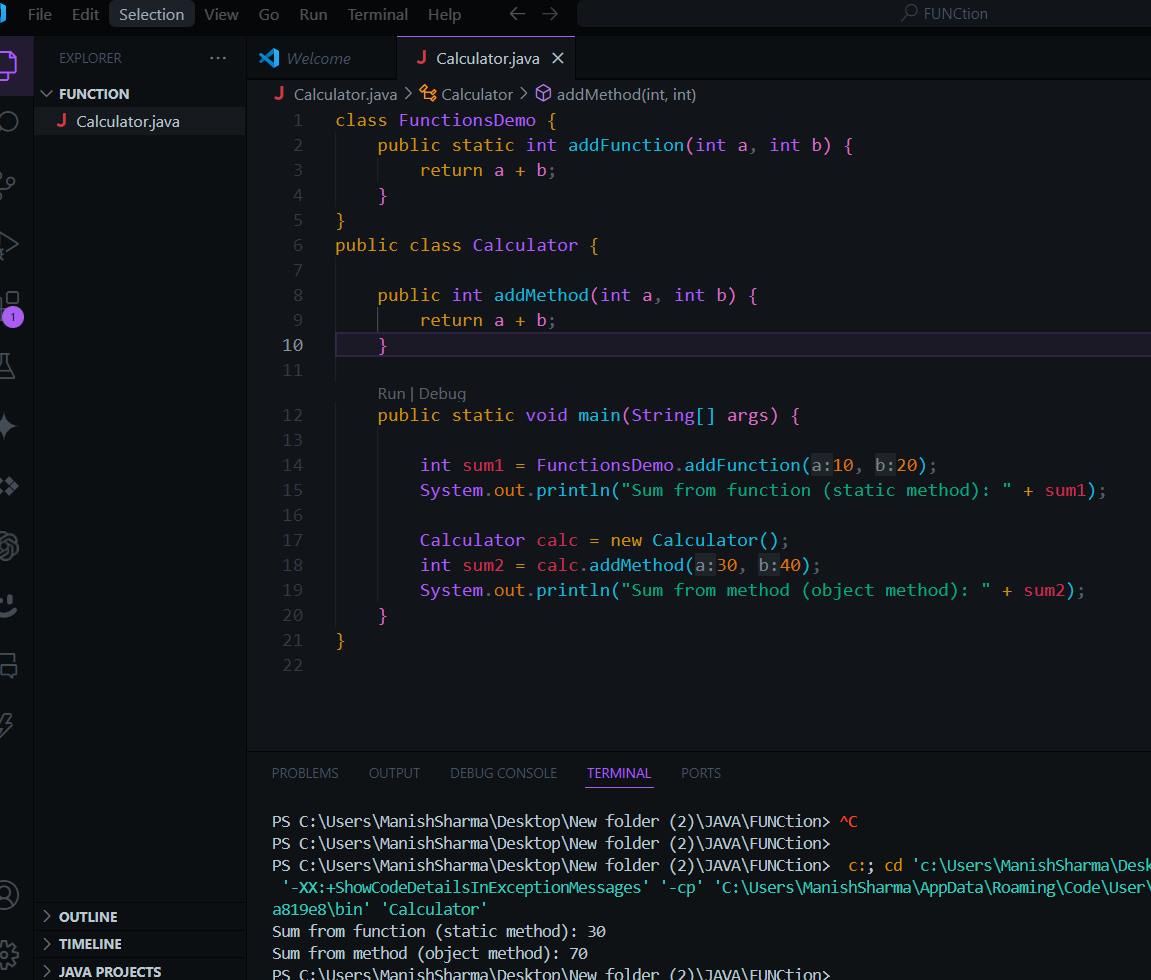
* **Function** ek aisa code block hota hai jo specific kaam karta hai aur optionally value return karta hai.
* General programming term hai, har language mein use hota hai (C, Python, Java, etc).
* Function independent ho sakta hai ya kisi class ka part nahi hota (jaise C ya Python mein).

### 2. **Method** kya hai?

* **Method** bhi ek function jaisa hi hota hai, lekin **ye sirf class ke andar hi hota hai**.
* Java mein, jo bhi function hota hai wo method hi hota hai kyunki Java mein sab kuch class ke andar hota hai.
* Matlab: Method = Class ke andar defined function.

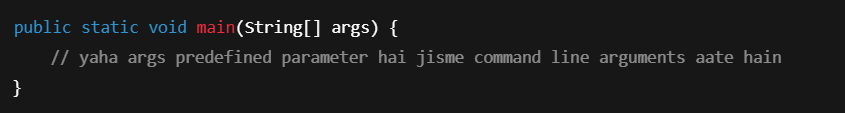
| **Term** | **Definition** |
| --- | --- |
| **Function** | Independent code block jo task karta hai, value return kar sakta hai. |
| **Method** | Class ke andar defined function, jo object ke saath related hota hai. |

| **Aspect** | **Function** | **Method** |
| --- | --- | --- |
| Location | Class ke bahar ya andar ho sakta hai (general) | Sirf class ke andar hota hai |
| Object-oriented term | General programming term | Object-oriented programming ka part |
| Usage in Java | Java mein method ke alawa function nahi hota | Java mein functions ko methods kehte hain |
| Call karne ka tarika | Direct call ho sakta hai | Object ke through ya class ke andar call hota hai |



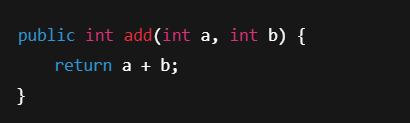
## 1. Predefined Parameters (Java mein direct nahi hote)

* **Predefined parameters** wo hote jo kisi programming language ya framework khud provide karta hai, jise user explicitly define nahi karta.
* Java mein directly predefined parameters nahi hote, lekin kuch special cases hain jaise main(String[] args) mein args parameter system se predefined hota hai (command line arguments ke liye).



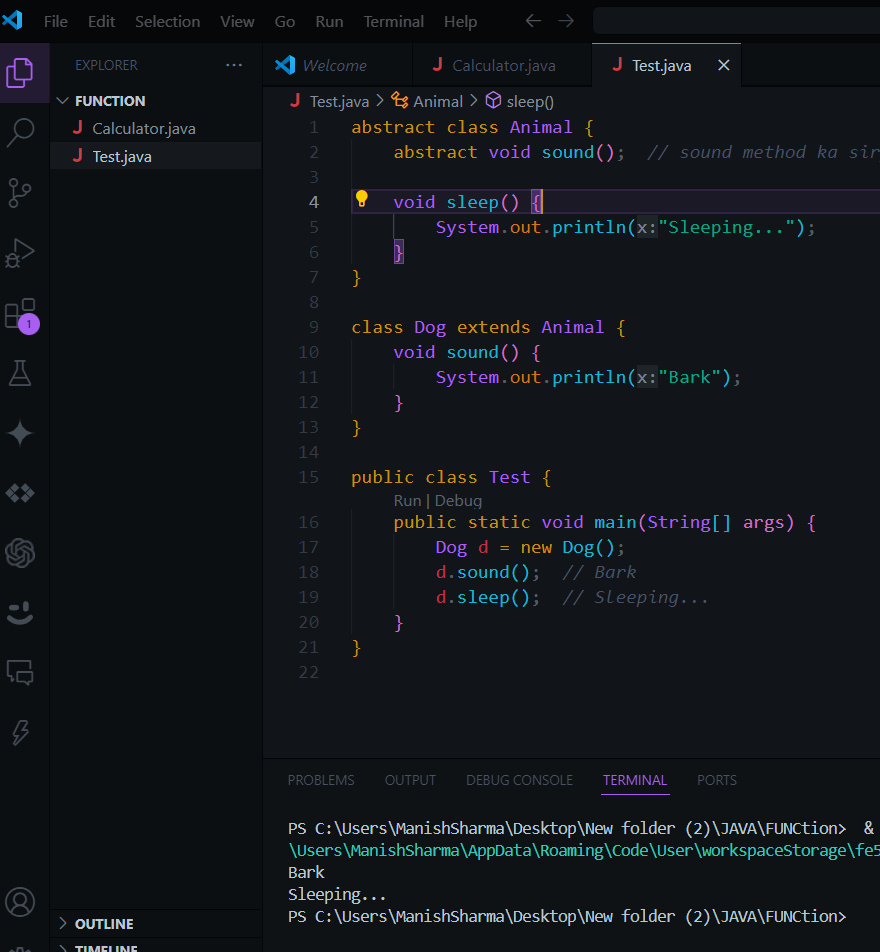
## 2. User-Defined Parameters

* **User-defined parameters** wo hote hain jo programmer khud method/function ke andar define karta hai.
* Ye parameters method ko input dete hain taaki method unke basis pe kaam kar sake.

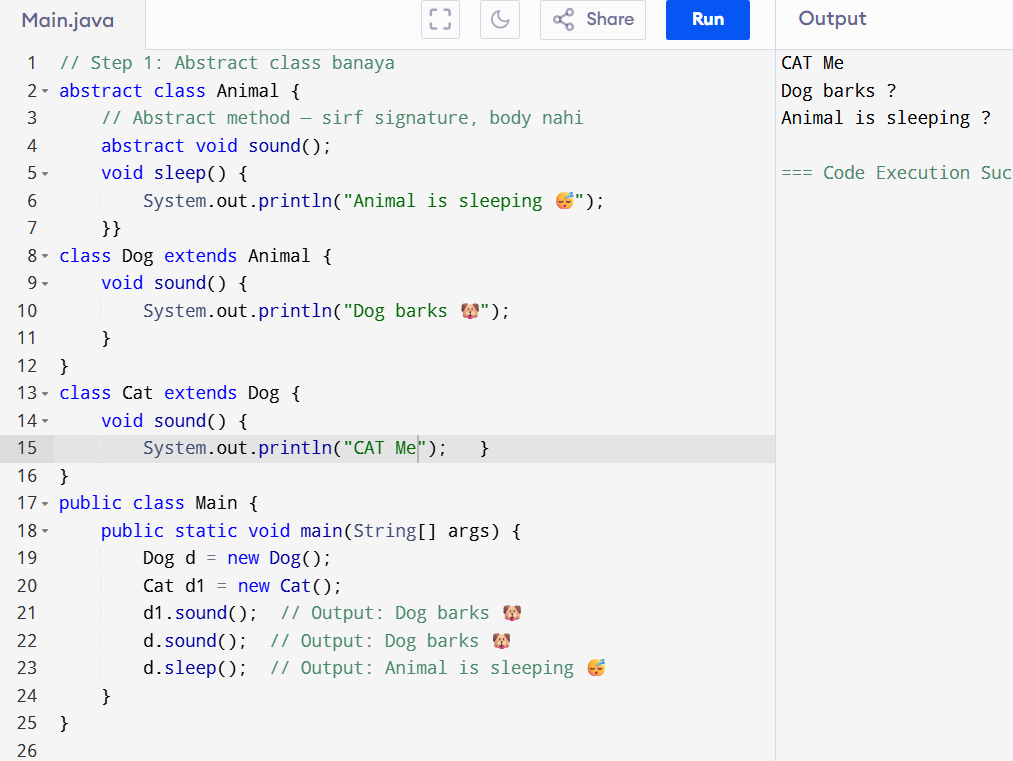


## 1. Abstract (सपना जैसा - पूरा नहीं, bas idea deta hai)

* **Abstract class**: Aisi class jo complete nahi hoti, sirf idea deti hai.
* **Abstract method**: Aisa method jiska body nahi hota, bas naam hota hai, baad mein subclass uska pura meaning deta hai.



**\*\* Jab bhi koi method abstract hota hai, toh us class ko bhi abstract banana padta hai.**

****

## 🡪 🔐 **final Keyword in Java**

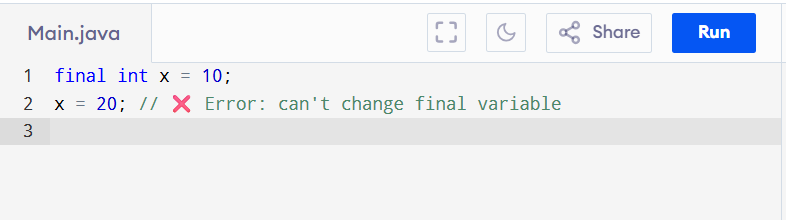
### 📘 **Definition:**

Java me final keyword ka use kisi **cheez ko fix/constant** banane ke liye hota hai — taaki usme **koi change na ho sake**.

🔹 final 3 cheezon ke saath use hota hai:

| **Use Case** | **Meaning** |
| --- | --- |
| final variable | Value fix ho gayi – badal nahi sakte (constant) |
| final method | Method override nahi ho sakta subclass me |
| final class | Class ko inherit (extend) nahi kar sakte |

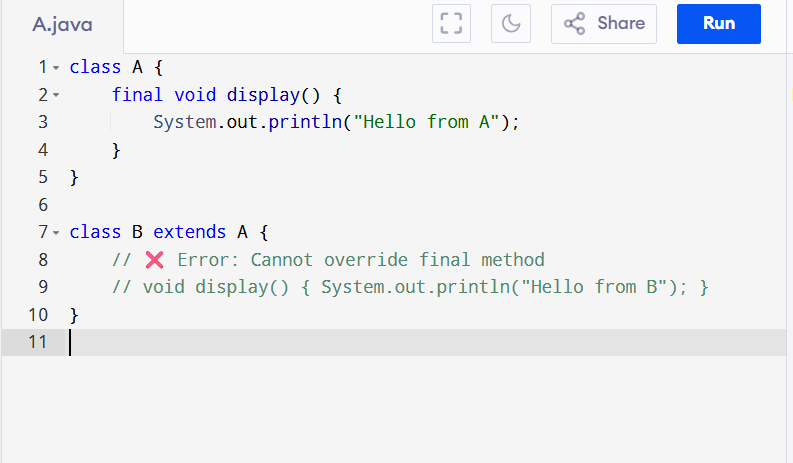
🔸 1. **Final Variable**



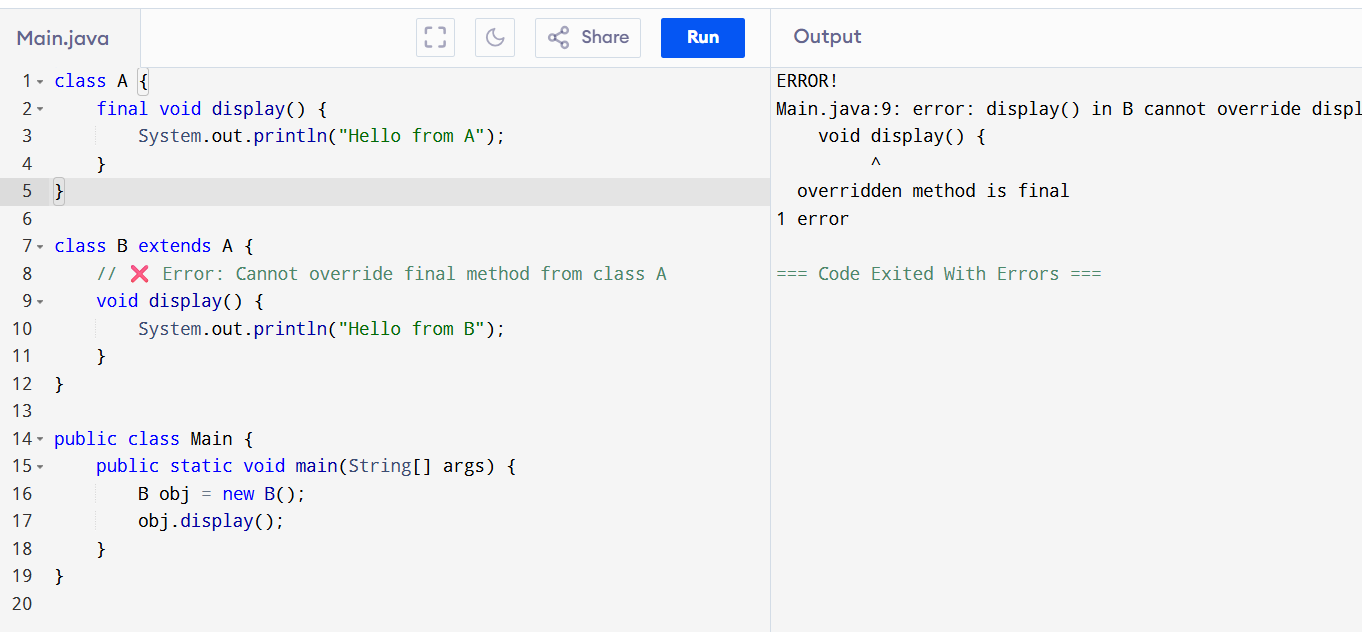
 x ki value **fix** ho gayi.

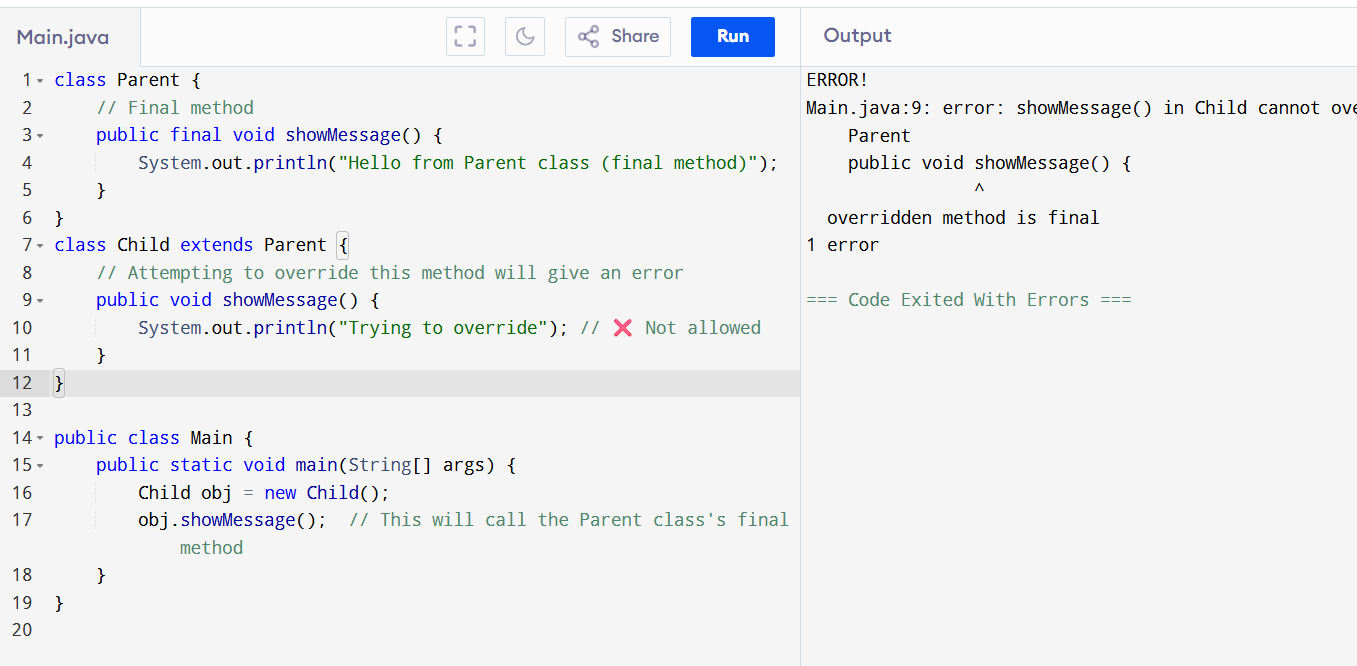
 Ab isse change karne ki koshish karoge toh **error aayega**.

2. **Final Method**

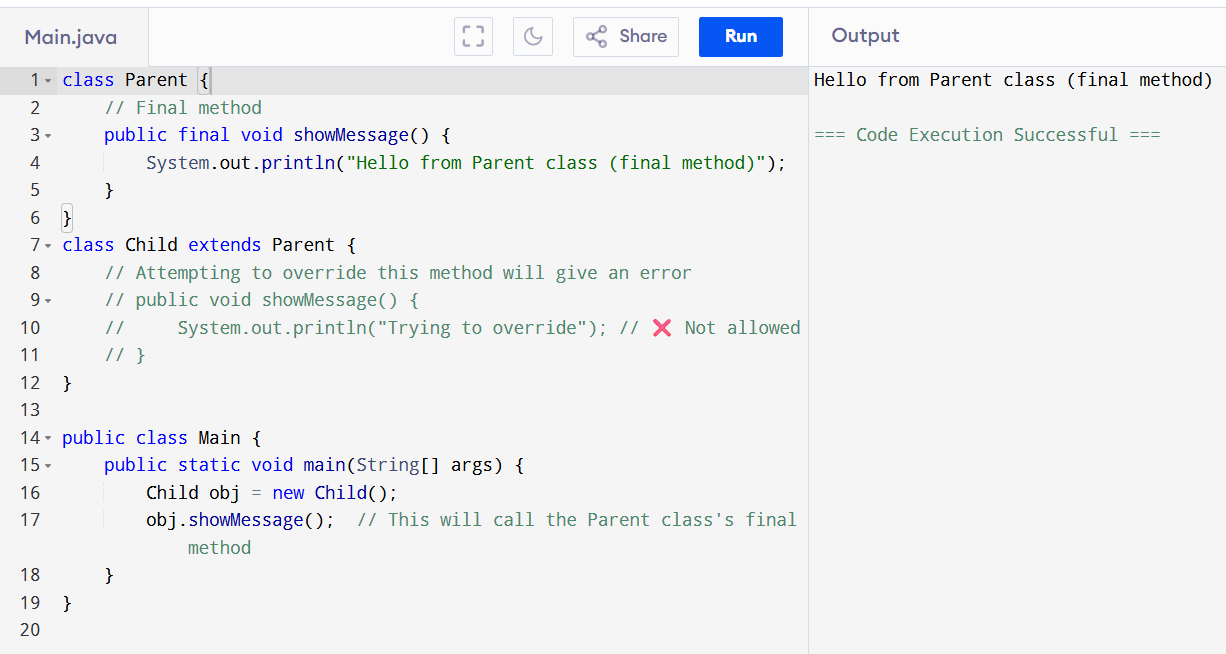


CODE :🡺



🡪ERROR CODE:  


Right Code 🡺

:🡺  


## 🡺 **✅ Final Class in Java**

Jab kisi class ko final declare kiya jaata hai, to **us class ko inherit (extend) nahi kiya ja sakta.**  
Yaani, koi doosri class us class ko **subclass nahi bana sakti**.

### 🔒 **Kyon use karte hain?**

* Jab aap chahte hain ki **koi class ka structure badla na jaaye**, tab use final banaya jaata hai.
* Yeh **security aur stability** ke liye hota hai.

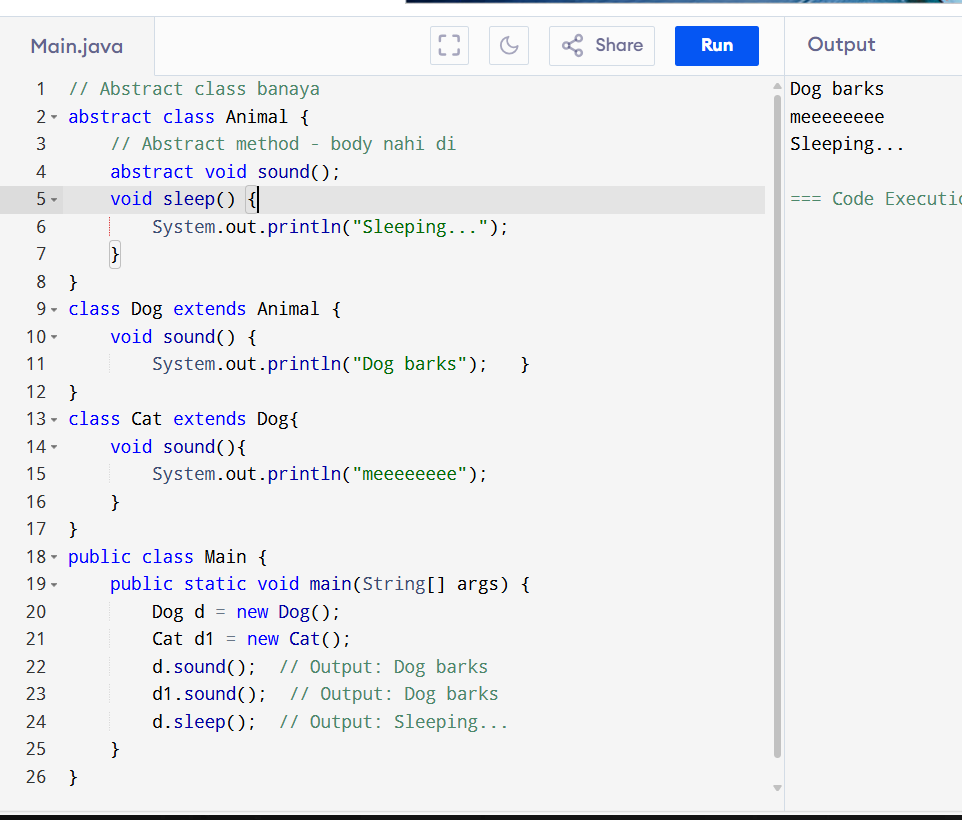


## **Abstract Method: 🡺**

"An abstract method is a method that has no body and must be implemented by the subclass.

## Simple Explanation (Hinglish Style):

Abstract method ek **aesa method hota hai jisme koi kaam (code) nahi hota**, sirf ye bataya jaata hai ki **ye method har child class me hona chahiye.**  
Ye sirf **abstract class** ke andar hota hai.



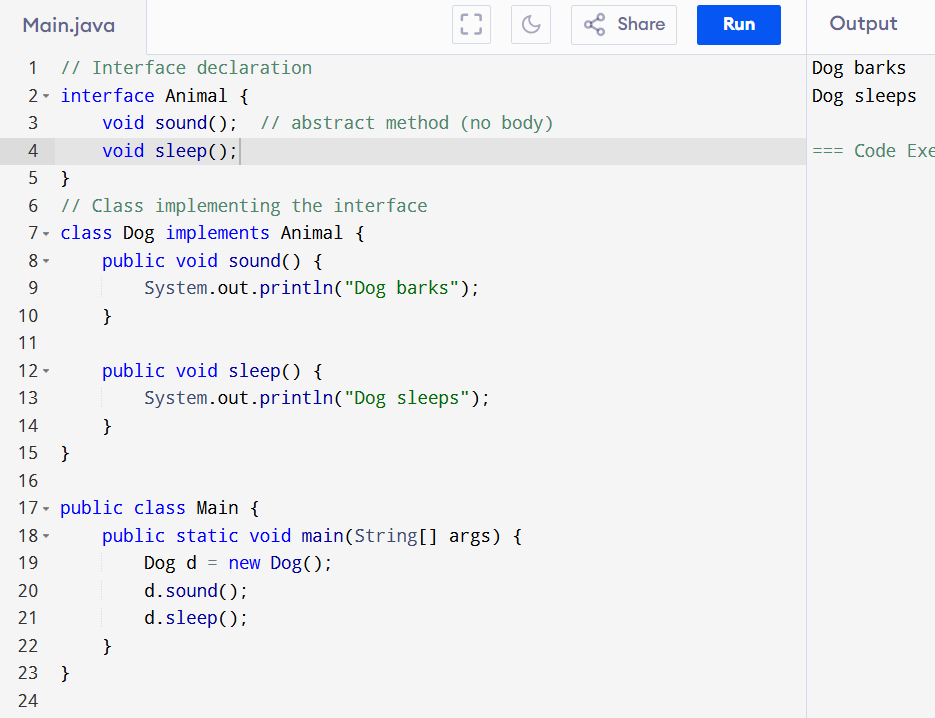
**Definition of Interface: 🡺**

"Interface is a rule book that tells what a class must do, but not how."

EX :=>

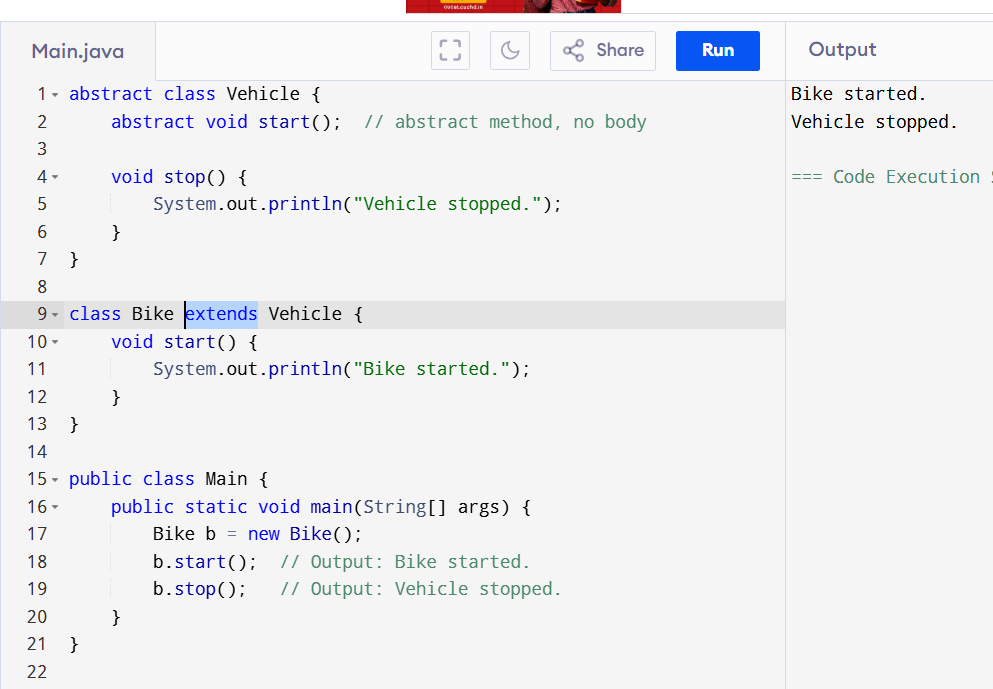
"Interface ek **contract** ya **rule book** jaisa hota hai.  
Jaise school ke teacher bolte hain —  
*‘Har student ko homework karna hai, test dena hai’* —  
to woh sirf **rules batate hain**, kaise karna hai ye student pe chhod dete hain."

SIMPLE 🡺  
( Interface ka matlab hai — kaam karne ka promise, kaam kaise karna hai woh class batayegi.” )

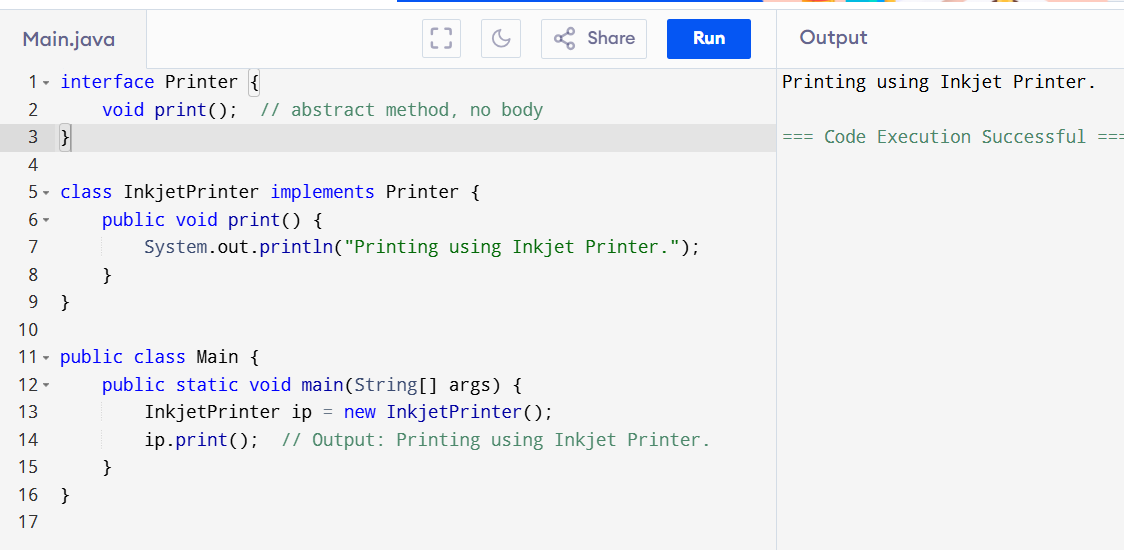


DIFFERNCE :

| **🔍 Feature** | **🧠 Abstraction (Abstract Class)** | **📘 Interface** |
| --- | --- | --- |
| 📌 Matlab kya hai? | Base class jisme **kuch method ka kaam chhupa hota hai** | Sirf **rules ya method names** define karta hai |
| 🔧 Code hota hai? | Abstract + normal method dono ho sakte hain | Sirf method names hote hain, **no code** |
| ⚙ Java me kaise? | abstract class se hota hai | interface keyword se banta hai |
| 🧱 Method body allowed? | Haan, kuch method ka code ho sakta hai | Nahi, sab method **without body** hote hain |
| 🧬 Variables | Can have variables (with different types) | Variables are **public + static + final** only |
| 🔗 Inheritance | extends se hota hai (ek hi class extend hoti hai) | implements se hota hai (multiple interface allowed) |
| 🧠 Kab use kare? | Jab kuch code common ho aur kuch chhupana ho | Jab sirf rules dene ho aur **100% abstraction** chahiye |

**ABSTRACT :**  


**Interface :**



## -------------------------------------------------------------------------------------------------------- **1. Interface ka object banta hai?**

**Nahi !**

Interface ka **direct object nahi bana sakte** Java me.

**Kyoon ?**

Interface sirf ek **blueprint (rules ka set)** hota hai, usme method ke bodies nahi hote. Isliye **uska koi asli object nahi hota**.

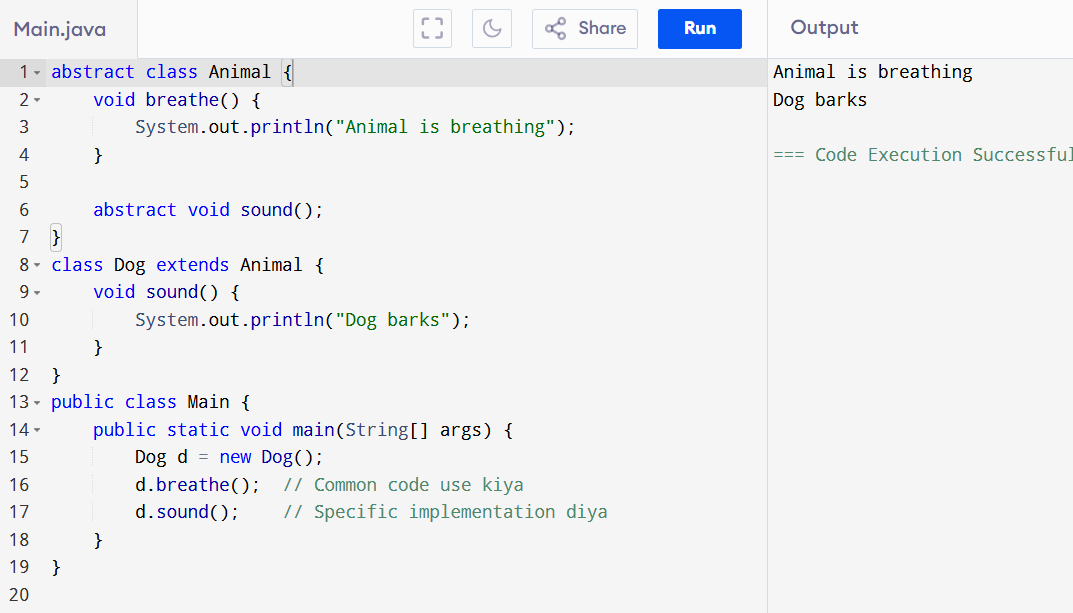
## **2. Abstract class ka object banta hai?**

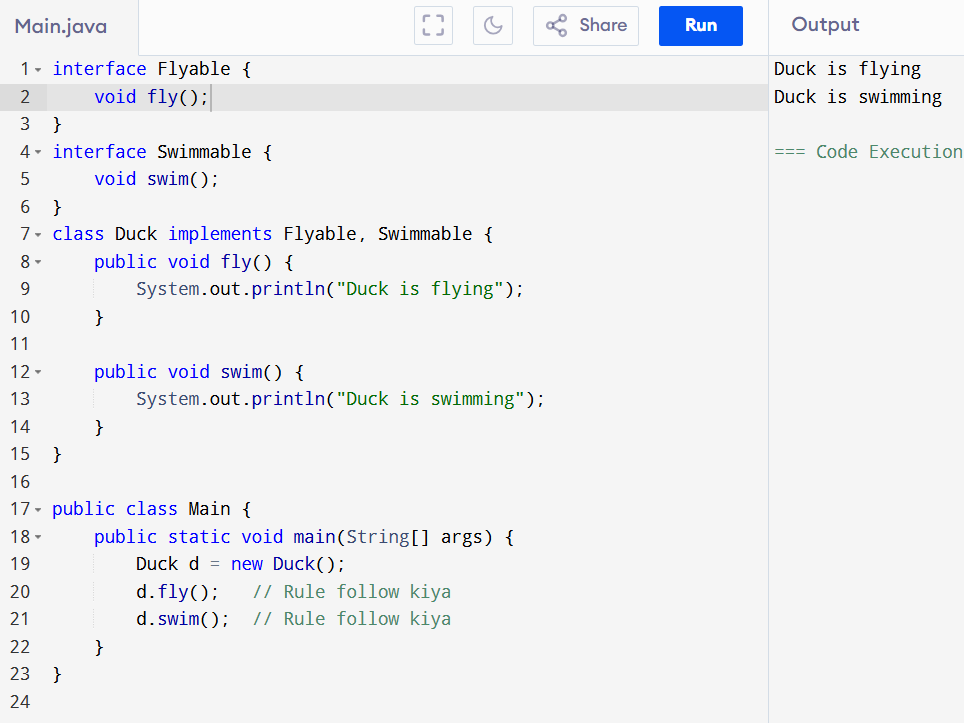
**Nahi!**  
Abstract class ka bhi **direct object nahi bana sakte**.

**Kyoon?**  
Kyuki abstract class me **kuch methods ka code nahi hota** (abstract methods), to Java ko pata nahi hota ki wo pura kaam kaise karega.

## **Par dono ka object kaise banta hai?**

* **Interface:**  
  Usko implement karne wali **class ka object** banta hai, jo saare methods ka code deta hai.
* **Abstract class:**  
  Usko extend karne wali **child class ka object** banta hai, jo abstract methods ko implement karta hai.

DIFFERENCE CODE 🡺 ------------------------------------------------------------------------  
  
  


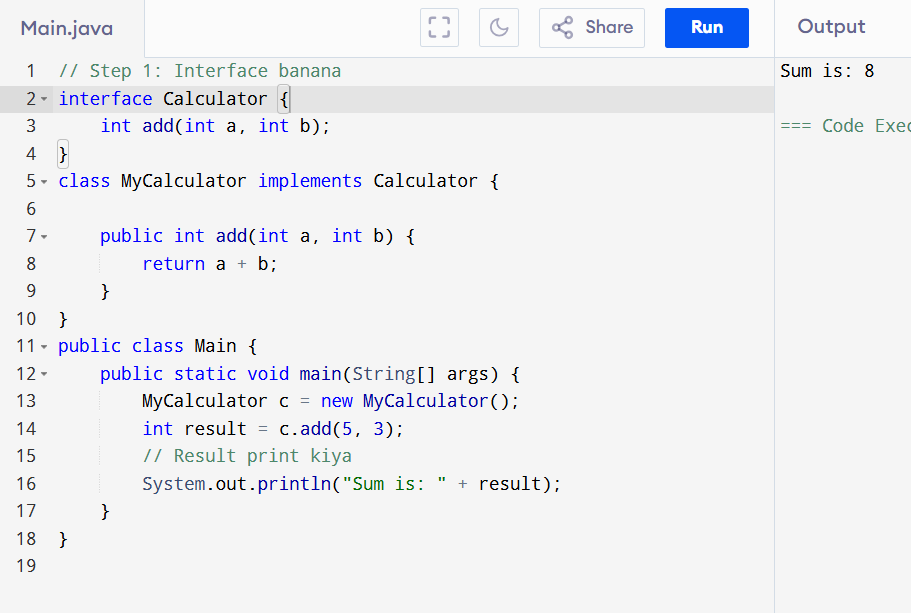


 Duck ek saath **do alag-alag abilities** (fly aur swim) le sakta hai.

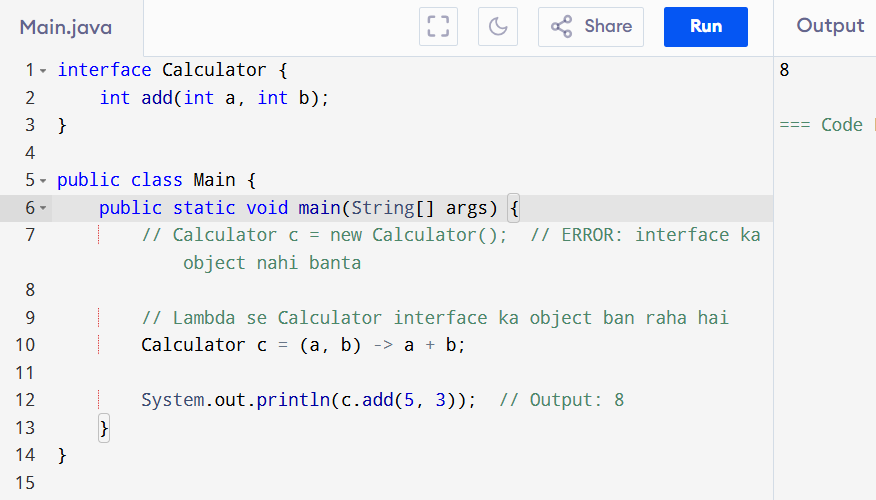
 Java class multiple abstract class extend nahi kar sakti, par multiple interface implement kar sakti hai.

**LAMBDA FUNCTION :** 🡺  
  
 ( "Lambda expression ek chhota function hota hai jo ek method ko inline likhne ka simple tareeka deta hai." )

🡺  
  
"Interface ka direct object nahi banta, par usko use karne ke 2 tareeke hote hain — Lambda ya Class."  
  
🡺  
w/o LAMBDA ==>



WITH LAMBDA 🡺



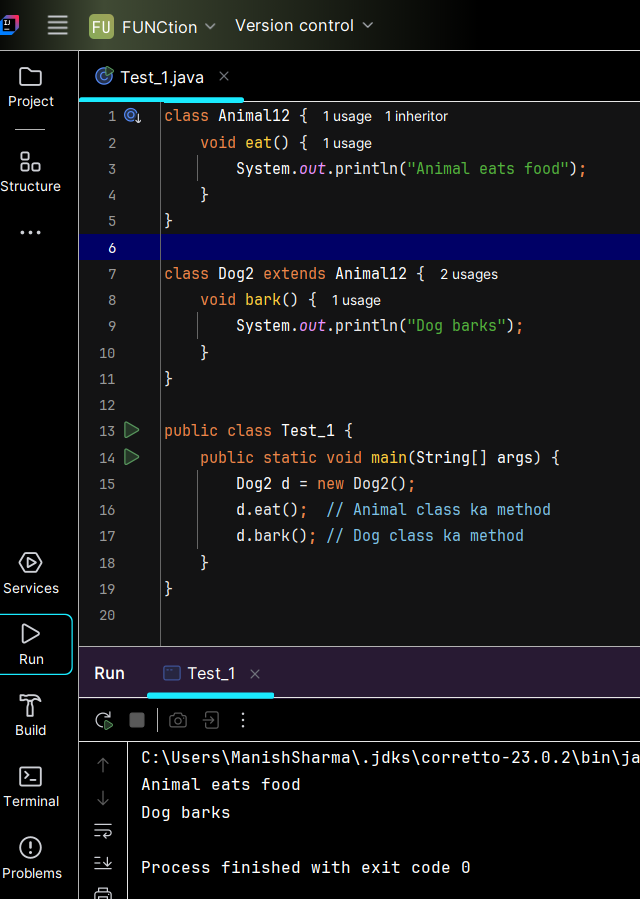
## 🡺 🧬 **Inheritance in Java**

### ✅ **Definition (Simple Words):**

**Inheritance ka matlab hota hai ek class (child) dusri class (parent) ke properties aur methods ko use kar sakti hai.**

Jaise ek **"Bachha apne Maa-Baap ki qualities inherit karta hai"**, waise hi programming mein ek class dusri class se features **"inherits"** karti hai.

| **Type** | **Description** |
| --- | --- |
| 1. Single Inheritance | Ek class dusri ek class se inherit karti hai |
| 2. Multilevel Inheritance | Ek class dusri se, wo kisi aur se (chain) |
| 3. Hierarchical Inheritance | Ek parent se kai child classes inherit |
| 4. Multiple Inheritance (via Interface) | Java class multiple interfaces inherit karti hai |



## 📘 **Types of Inheritance in Java –>**

### 🔰 **What is Inheritance?**

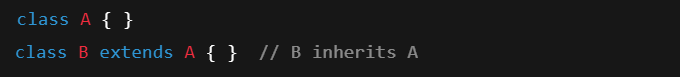
**Inheritance** ek object-oriented concept hai jisme ek class (child/subclass) dusri class (parent/superclass) ke properties (variables) aur methods (functions) ko use kar sakti hai.  
Isse **code reuse** hota hai, aur naye features add karna aasaan ho jata hai.

## 🔢 **Types of Inheritance in Java:**

### 1️⃣ **Single Inheritance**

✅ **Java Supported**

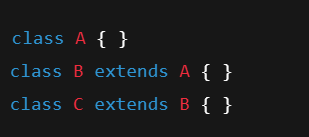
👉 Isme **ek class**, dusri **ek hi class** se inherit karti hai.  
Ek **child class**, ek **parent class** ke methods/properties use karti hai.



### 2️⃣ **Multilevel Inheritance**

✅ **Java Supported**

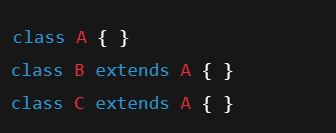
👉 Ek class **dusri class se inherit** karti hai, aur fir teesri class **usse inherit** karti hai.  
Yani **chain of inheritance** banti hai.



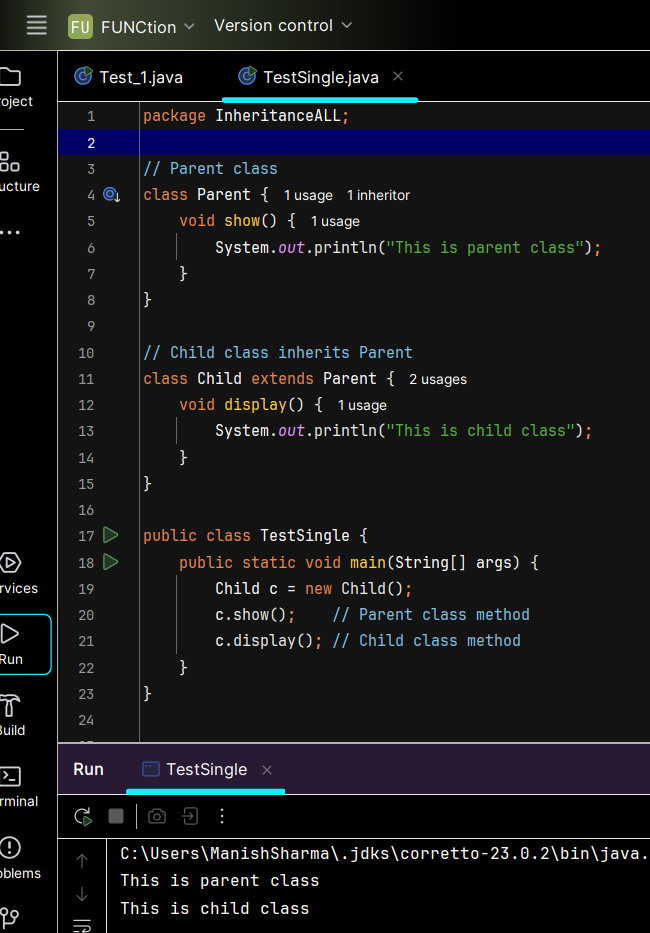
### 3️⃣ **Hierarchical Inheritance**

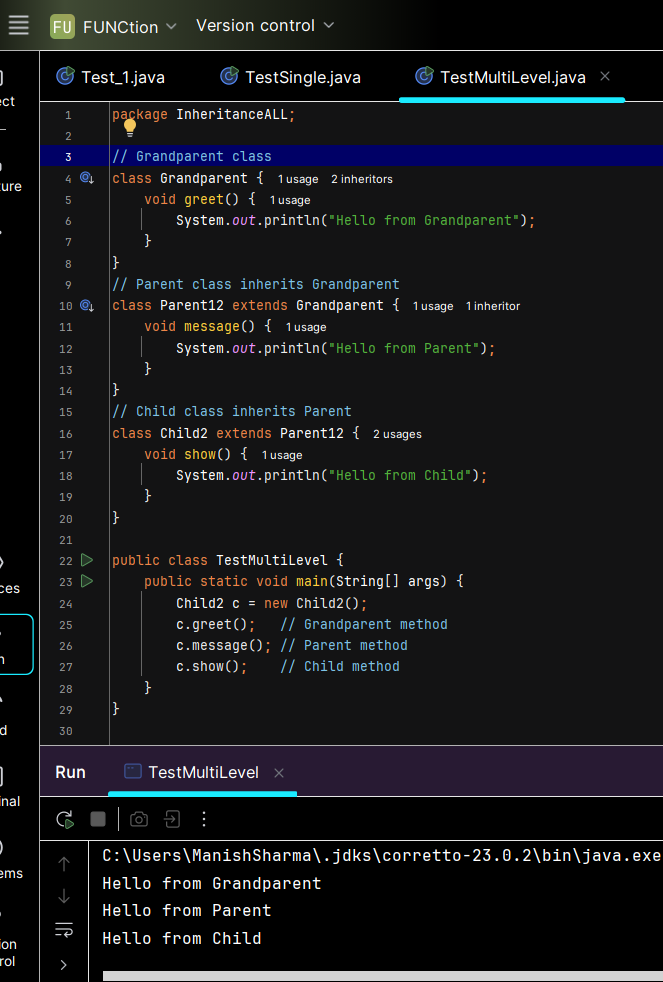
✅ **Java Supported**

👉 **Ek parent class** se **multiple child classes** inherit karti hain.  
Sabhi child classes **same parent ke method** ko access kar sakti hain

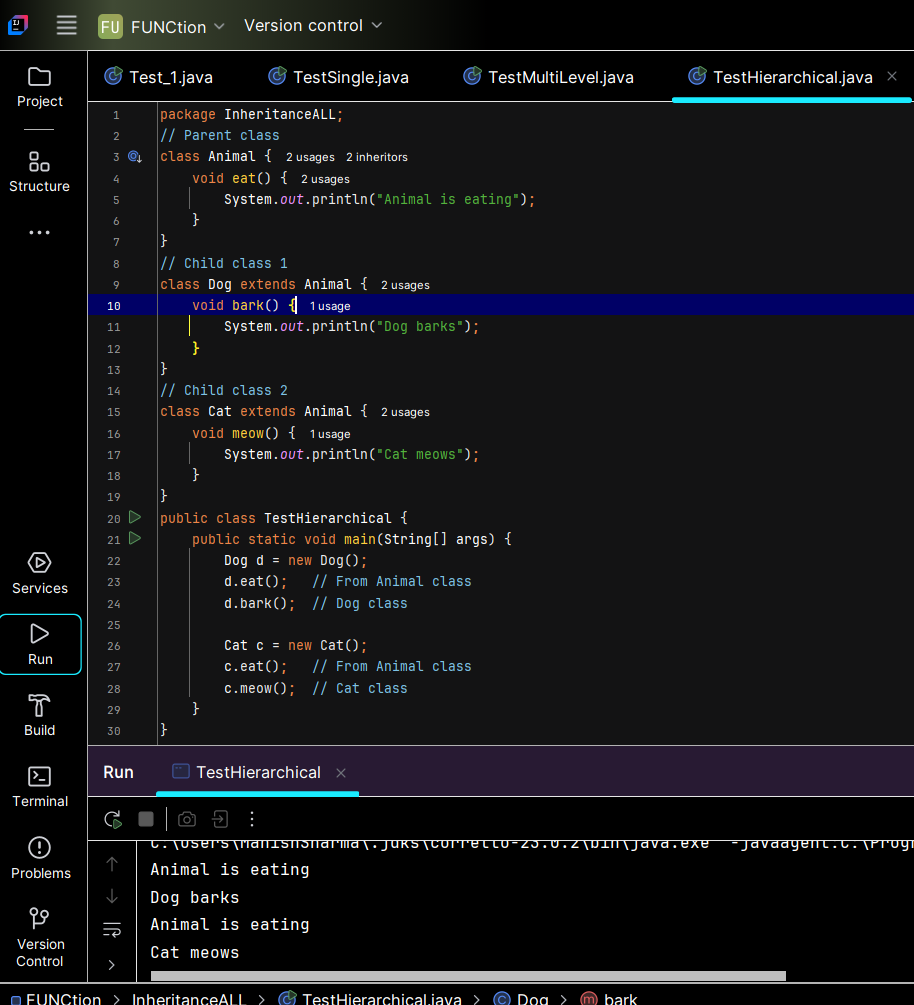


| **Inheritance Type** | **Supported in Java** | **Description** |
| --- | --- | --- |
| Single Inheritance | ✅ Yes | One child class inherits one parent class |
| Multilevel Inheritance | ✅ Yes | Inheritance in chain (A → B → C) |
| Hierarchical Inheritance | ✅ Yes | One parent class, many child classes |
| Multiple Inheritance | ❌ No (class) | Allowed via interface only |
| Hybrid Inheritance | ❌ No (class) | Achieved via interface combination |

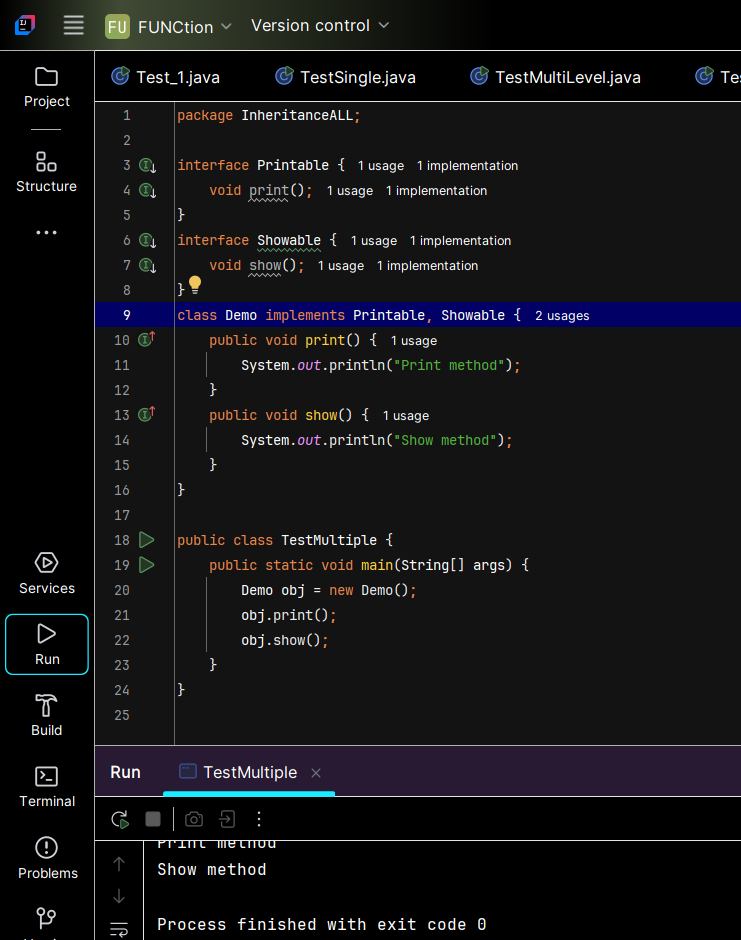
**CODE : : 🡪**  
  


🡺MULTILEVEL 🡺  
  


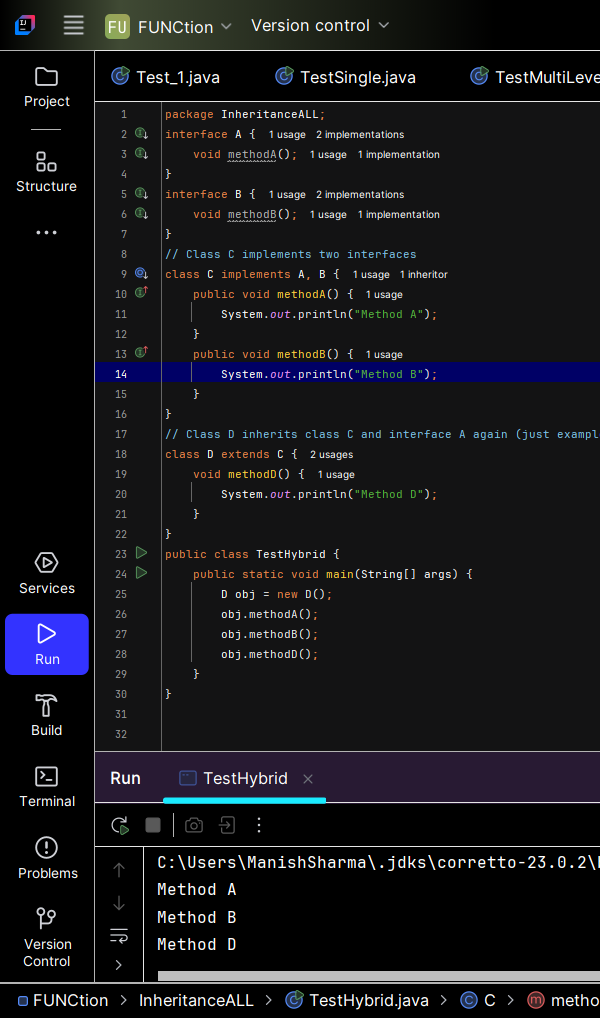
**3️⃣ Hierarchical Inheritance**



**4️⃣ Multiple Inheritance (Using Interfaces)**



**5️⃣ Hybrid Inheritance (Using Interfaces + Classes)**

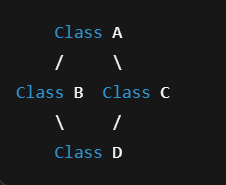
****

## 🧐 **Why Multiple Inheritance with Interfaces only?**

### 1. **Java mein Multiple Inheritance (Class ke through) allowed nahi kyun?**

* Agar ek class **do ya zyada classes** se directly inherit kare (multiple inheritance), to **"Diamond Problem"** create ho sakta hai.

🔍 **Diamond Problem kya hota hai?**

****

### 2. **Interface kaise solve karta hai?**

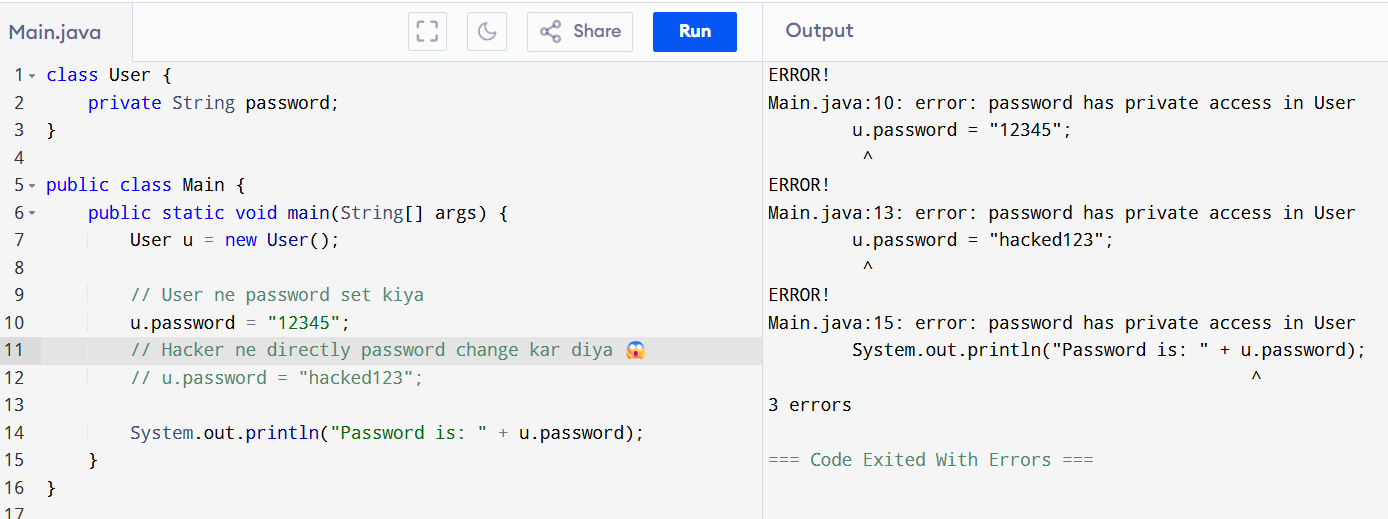
* Interfaces mein **sirf method declarations** hote hain (method body nahi hoti — ya phir Java 8 se default methods hote hain, jo alag handle hote hain).
* Isliye **ambiguity kam ho jati hai**, kyunki class ko clearly methods implement karne hote hain.
* Interfaces multiple ho sakte hain, kyunki koi method body nahi hoti ya developer khud clearly define karta hai.

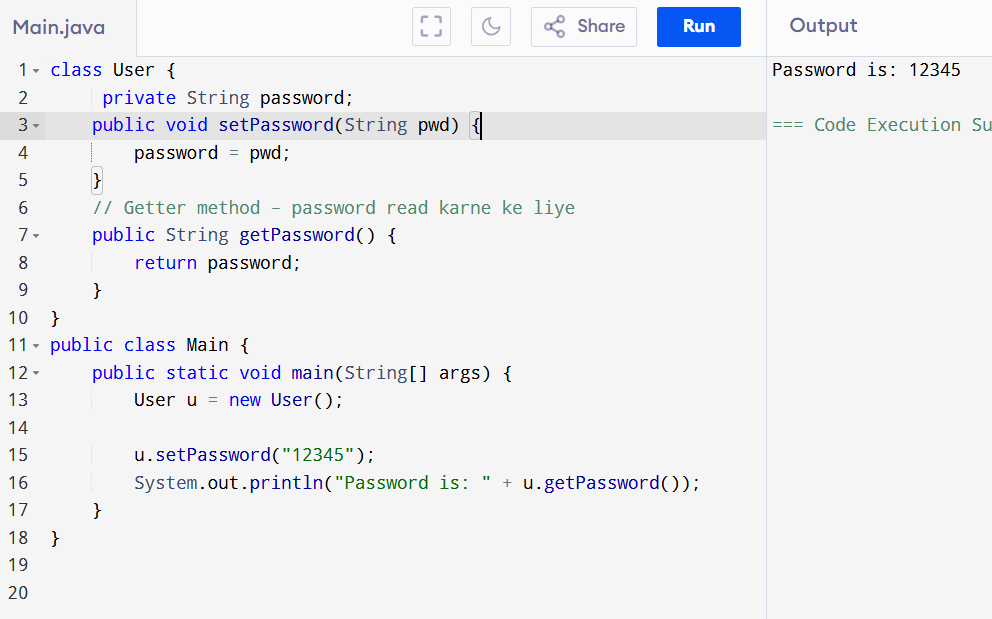
| **Point** | **Class Inheritance** | **Interface Inheritance** |
| --- | --- | --- |
| Multiple inheritance | **Not allowed** | **Allowed** |
| Diamond problem | Ho sakta hai (ambiguity) | Nahi hota (method body nahi) |
| Method implementation | Parent class se directly milti | Implement karni hoti hai child class ko |
| Flexibility | Limited | Zyada flexibility deta hai |

**Encapsulation**

is a concept in object-oriented programming where we **wrap the data (variables)** and **code (methods)** together into a single unit (class), and **restrict direct access** to some of the object’s components by making them private and accessing them through getter and setter methods.

"**Encapsulation** ka matlab hota hai — data ko **private** rakhna, aur **getter-setter** ke through access dena."



Solution 🡺:  
  


## 🔁 **Polymorphism – Definition**

**Polymorphism** means "**many forms**". In Java, it allows **one function, method, or object to behave differently in different situations.**

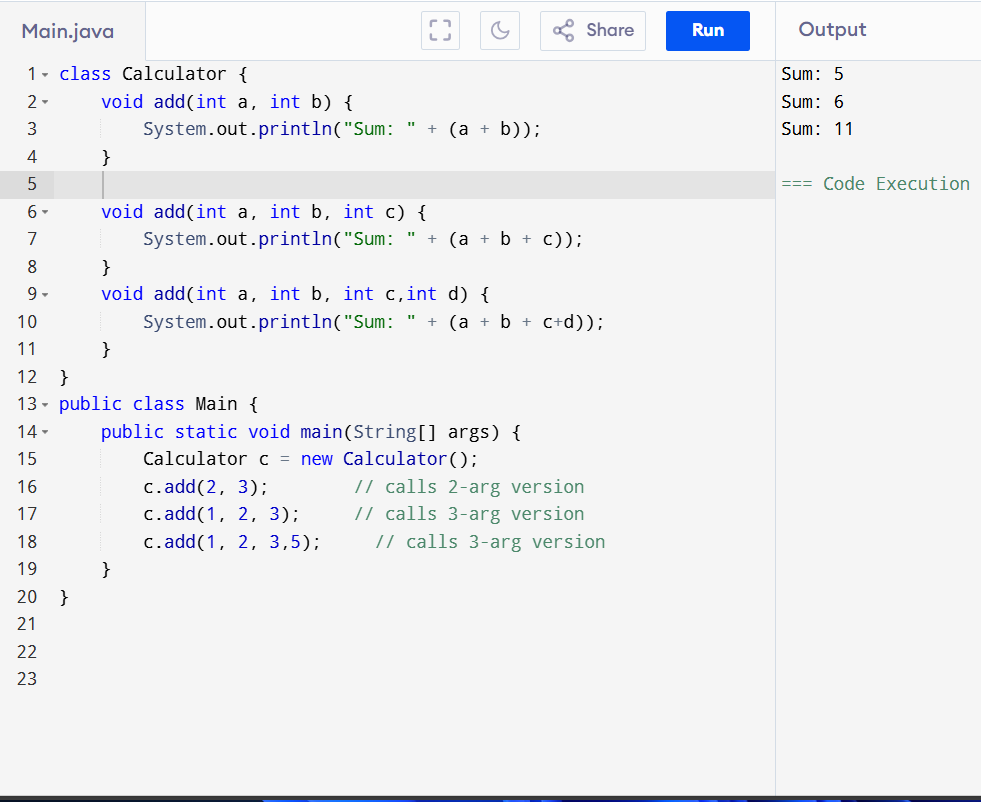
### 📚 Simple Definition (Easy Words):

Jab **ek hi method ya object** ka **multiple forms me use** hota hai — usko **Polymorphism** kehte hain.

| **Type** | **How it happens** | **Also called as** |
| --- | --- | --- |
| 1. Compile-time | Method Overloading | Static Polymorphism |
| 2. Run-time | Method Overriding | Dynamic Polymorphism |

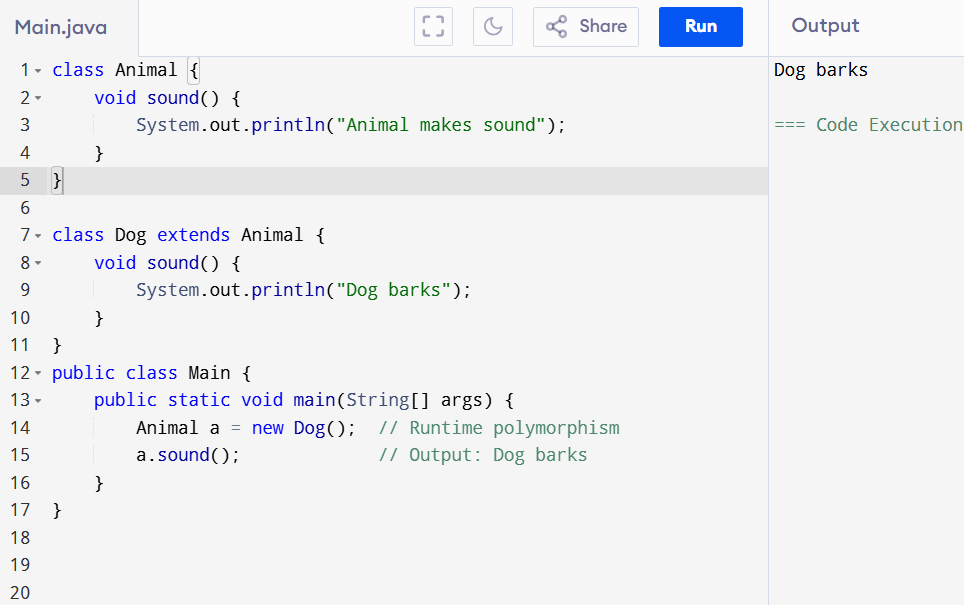
## **🔷 1. Method Overloading (Compile-Time)**

Same method name, **different parameters** — compiler decides which one to run.



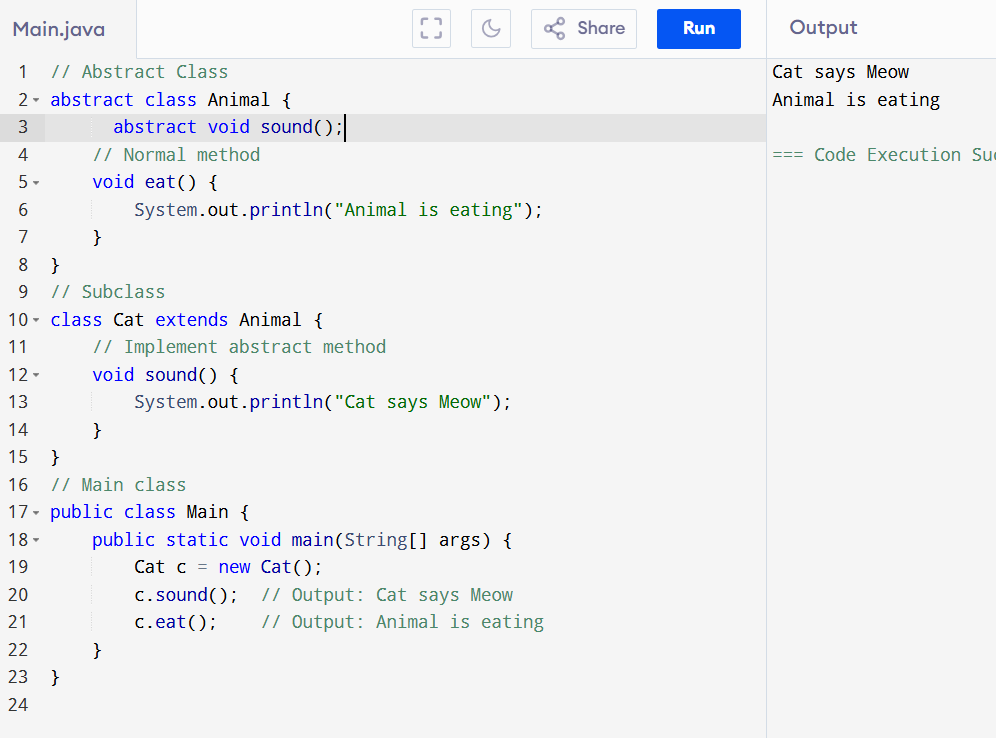
## **🔷 2. Method Overriding (Run-Time)**

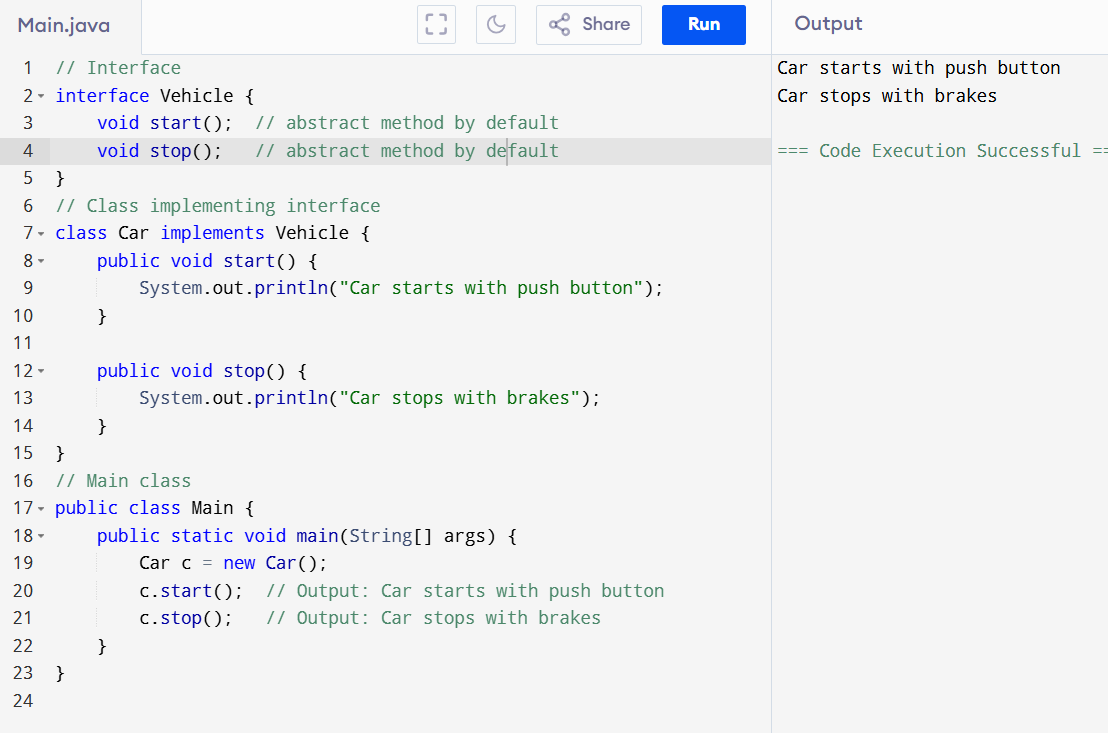
Same method name and parameters — **but in child class**, different behavior.



## 🚀 **Abstraction – Definition**

**Abstraction** means **hiding the internal implementation** and **showing only the essential details** to the user.



🡺  
Interface   
  


| **Abstract Class** | **Interface** |
| --- | --- |
| abstract class Animal {} | interface Vehicle {} |
| Can have normal + abstract methods | Only abstract methods (Java 8+ allows default too) |
| Use extends to inherit | Use implements to inherit |
| One class can extend 1 abstract class | One class can implement multiple interfaces |