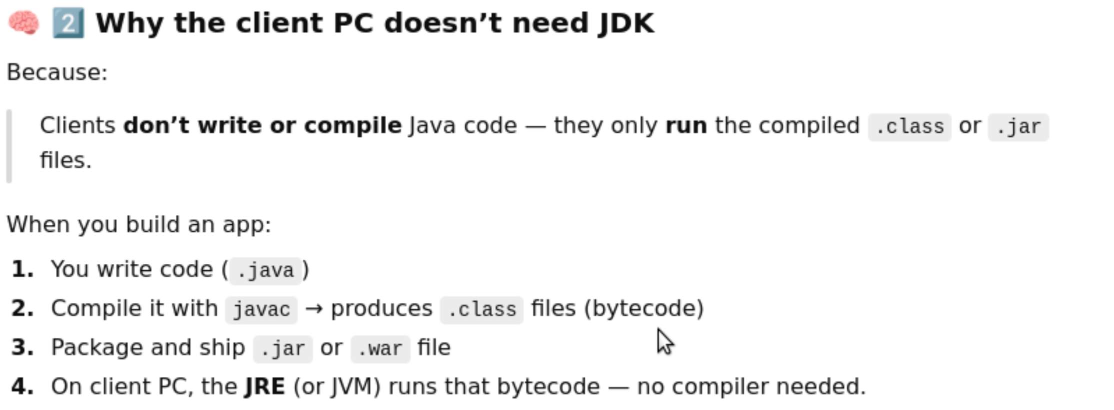
JAVA(Write once and run anywhere)

U need to install JDK(Java dev kit). //sudo dnf install java-21-openjdk-devel

How java works: *Java is platform independent. But that system has to have JVM(Java Virtual machine). Java file can't run without JVM. JVM work onto OS. JVM don't accept Java code. It only execute byte code Compiler(javac) will compile all java code and create a byte code file, which will execute by JVM. Start execute with "Main" method.*
To run java applications we need library's in runtime. These library's are in JRE(Java runtime environment). And JVM is a part of JRE. So JVM can use library's from JRE to run the code.
For development purpose, we need JDK. JDK will have JRE and JRE will have JVM. JDK don't need where we will run our application(client PC).

Component	Purpose
JDK (Java Development Kit)	Used by **developers** to *write, compile, and build* Java programs. Includes `javac`, debugger, etc.
JRE (Java Runtime Environment)	Used by **users (clients)** to *run* Java programs. Includes `java`, JVM, and core libraries.



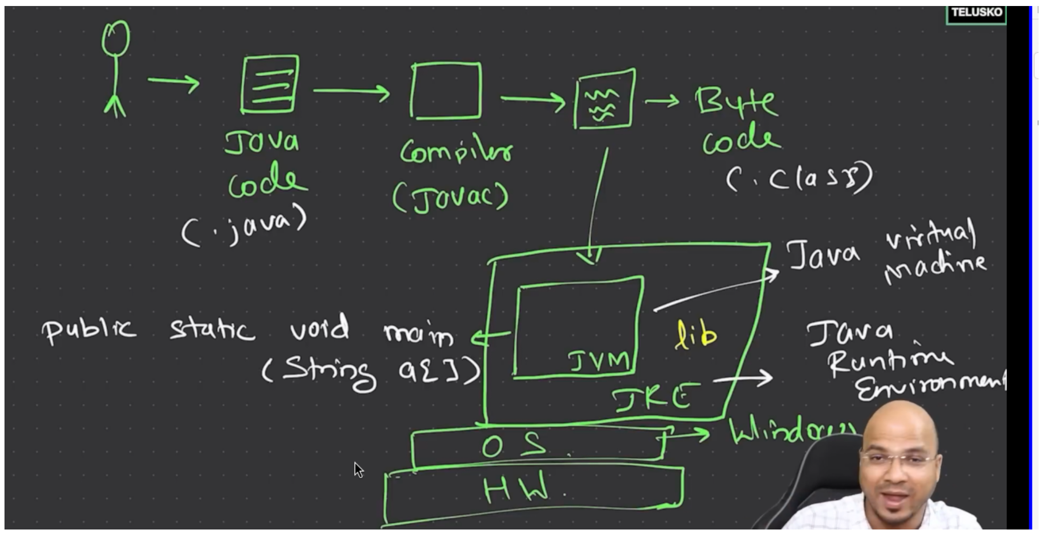
🧠 2️⃣ Why the client PC doesn't need JDK

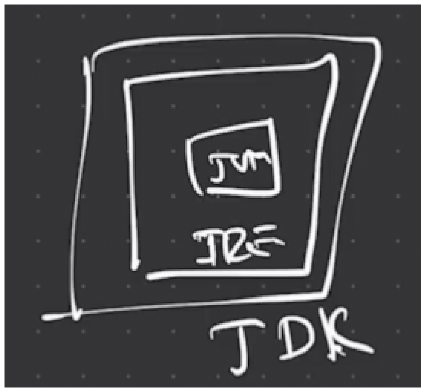
Because:

> Clients **don't write or compile** Java code — they only **run** the compiled `.class` or `.jar` files.

When you build an app:

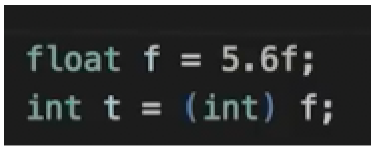
1. You write code (`.java`)
2. Compile it with `javac` → produces `.class` files (bytecode)
3. Package and ship `.jar` or `.war` file
4. On client PC, the **JRE** (or JVM) runs that bytecode — no compiler needed.





-----------x---------

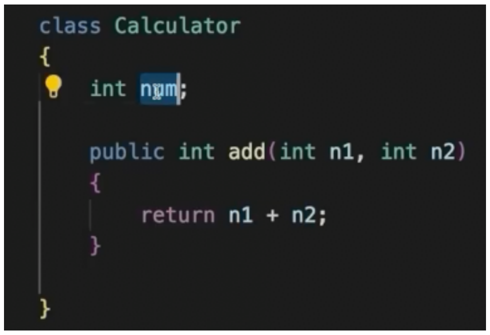
Type Casting:



```java
float f = 5.6f;
int t = (int) f;
```

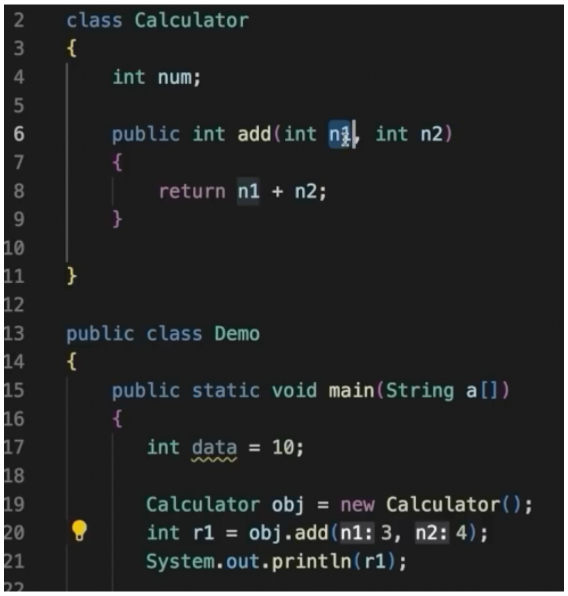
---------x---------

For Every method there will a own stack inside JVM memory. Another memory is Heap.



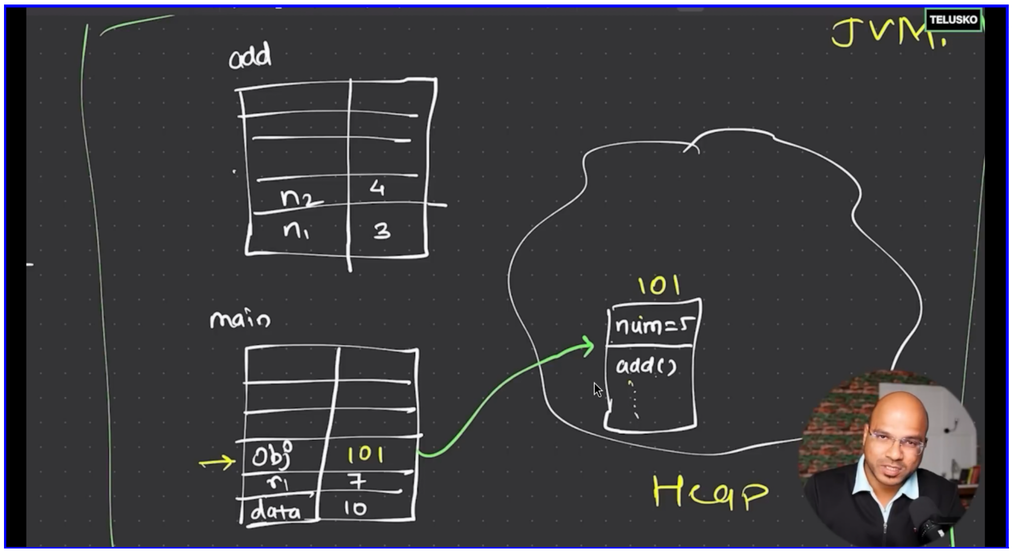
```java
class Calculator
{
    int num;

    public int add(int n1, int n2)
    {
        return n1 + n2;
    }

}
```

Here "Num" variable is instance variable, and "n1,n2" is local variable.(Below)



```java
class Calculator
{
    int num;

    public int add(int n1, int n2)
    {
        return n1 + n2;
    }

}

public class Demo
{
    public static void main(String a[])
    {
        int data = 10;

        Calculator obj = new Calculator();
        int r1 = obj.add(n1: 3, n2: 4);
        System.out.println(r1);
```

There Will be a stack on for main method. Also for add(Lint 6;) method. On line 19; obj is not an object. Its a reference variable of the object. That object memory will be on heap memory. 'obj' will store memory location. On that object memory there are 2 part. 1 aprt will store all instance variables and other store method definition. But that have methods definition only. There will be a stack for that(add) method. And methods local variable will store on that stack.



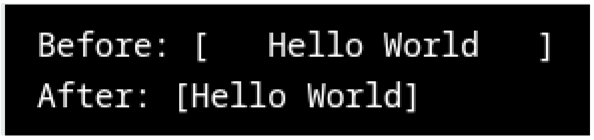
---------x--------

Methods:

Math.random(); *// return double, return range 0>=x<=1, Here "Math" is library*

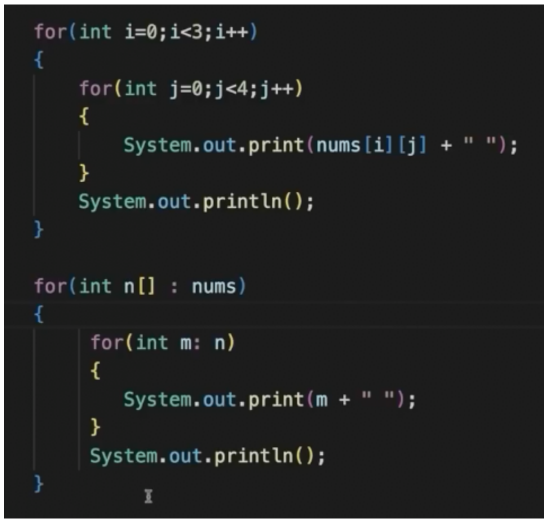
objName.capacity(); *// will give size. For string it will give extra 16 char. If string a="Emon" //4 char. Capacity will be 16+4=20 char. Extra 16 char space will give. This is for* ***stringBuffer****(Mutable string) see: few lecture ahead. Give extra 16 char, cause if no continuous memory available it have to reallocate memory. To reduce reallocation. "Length" and "capacity" diff method.*

Trim(); *// remove space from beginning and end of string.*



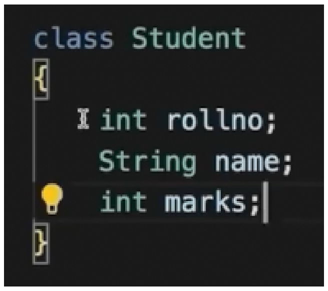
```
Before: [   Hello World   ]
After: [Hello World]
```

---------x---------

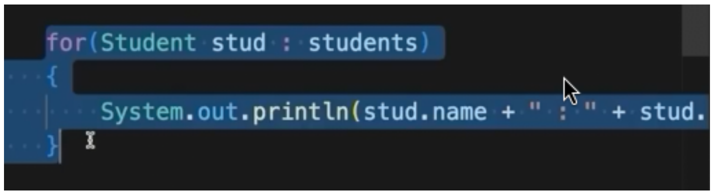


```
for(int i=0;i<3;i++)
{
    for(int j=0;j<4;j++)
    {
        System.out.print(nums[i][j] + " ");
    }
    System.out.println();
}

for(int n[] : nums)
{
     for(int m: n)
     {
        System.out.print(m + " ");
     }
     System.out.println();
}
```

Both are same.(Above) Second is useful when we don't know the array size. Is called "for each loop". This used only for array of any type. Like array of object. (Below)

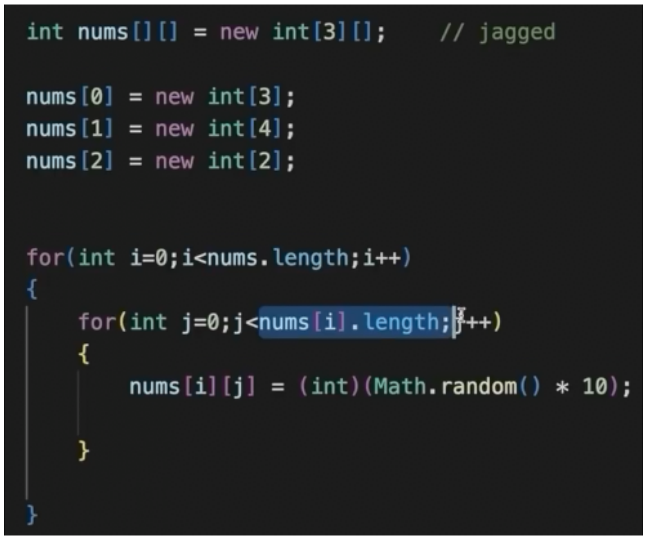


```java
class Student
{
    int rollno;
    String name;
    int marks;
}
```



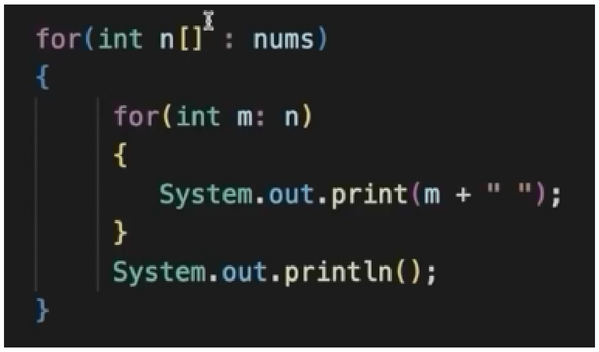
```java
for(Student stud : students)
{
    System.out.println(stud.name + " : " + stud.
}
```

What if we know the row number. But the column size may vary of diff rows for 2D array. This called Jagged Array. (Below)



```java
int nums[][] = new int[3][];    // jagged

nums[0] = new int[3];
nums[1] = new int[4];
nums[2] = new int[2];


for(int i=0;i<nums.length;i++)
{
    for(int j=0;j<nums[i].length;j++)
    {
        nums[i][j] = (int)(Math.random() * 10);

    }

}
```

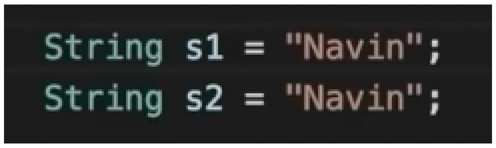
to Print:



```java
for(int n[] : nums)
{
    for(int m: n)
    {
        System.out.print(m + " ");
    }
    System.out.println();
}
```

In Java everything is an object. Like here array is an object.
By default array element value is 0.
** Exceptions are runtime error.*

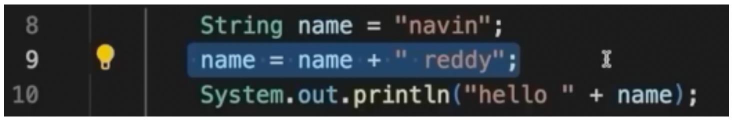
-------------x------------



```java
String s1 = "Navin";
String s2 = "Navin";
```

In heap memory there is a section called "String const pool", where all string stored. Here only 1 object will create. For creating "s2" object, Java will first search whether is present previously or not. Both "s1,s2" variable will contain same reference address(on stack) stored in heap.

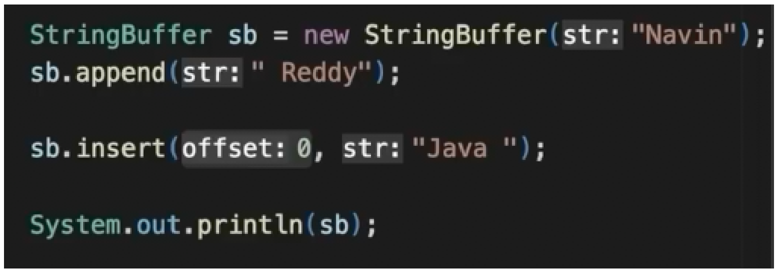
Strings are const.



```java
8       String name = "navin";
9       name = name + " reddy";
10      System.out.println("hello " + name);
```

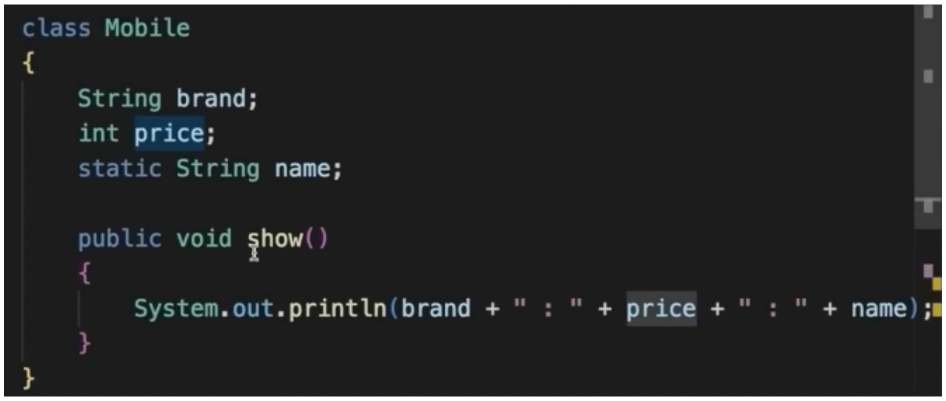
Here, on line 9; we are changing string. But they are const. Suppose name address is 100. when we are adding "reddy" it creates a new object(heap) after concatenating name. Replace name address(on stack) to that new object like 101. Here data is not changed. The "navin" object is eligible for garbage collection.

Strings can be changeable(Mutable). Using StringBuffer, StringBuilder



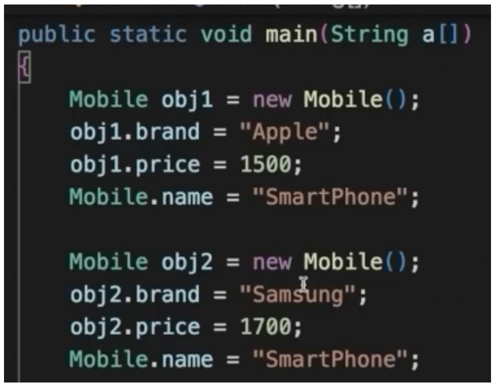
```java
StringBuffer sb = new StringBuffer(str: "Navin");
sb.append(str: " Reddy");

sb.insert(offset: 0, str: "Java ");

System.out.println(sb);
```

StringBuffer is thread safe, StringBuilder is not. *We will learn later. Methods are same for both.*
***** All memory used by code is from JVM. Stack, heap are inside JVM.*
Static:

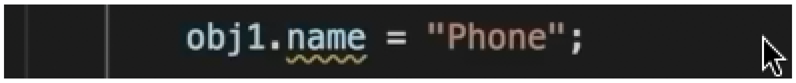


```java
class Mobile
{
    String brand;
    int price;
    static String name;

    public void show()
    {
        System.out.println(brand + " : " + price + " : " + name);
    }
}
```

If we want that for all object 1 property will be same. So we can use static". Static variable should called with class name, but it can be called with object name, but not recommended. If one object property of static variable is changed it effects all. It is class member, not object member. In Java, static means belonging to the class itself, not to any specific object (instance).

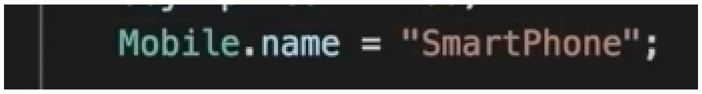


```java
public static void main(String a[])
{
    Mobile obj1 = new Mobile();
    obj1.brand = "Apple";
    obj1.price = 1500;
    Mobile.name = "SmartPhone";

    Mobile obj2 = new Mobile();
    obj2.brand = "Samsung";
    obj2.price = 1700;
    Mobile.name = "SmartPhone";
```



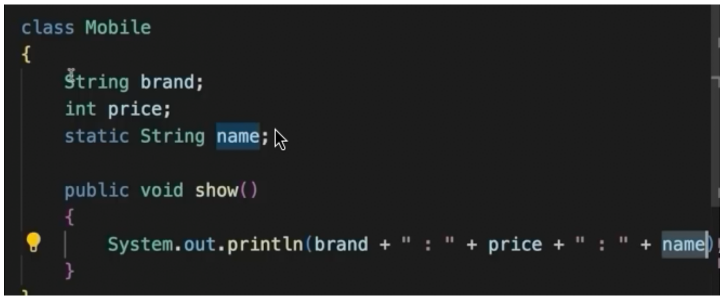
```java
obj1.name = "Phone";
```

Setting one obj name(inside main) changes all obj name properties. While using static



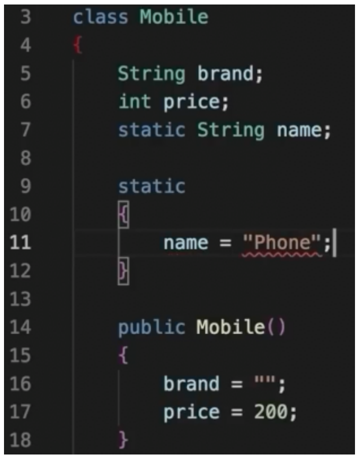
```java
Mobile.name = "SmartPhone";
```

Static variable should be called with class name(above). Static variable shared by diff object. We can us static variable on non-static variable. (below)



```java
class Mobile
{
    String brand;
    int price;
    static String name;

    public void show()
    {
        System.out.println(brand + " : " + price + " : " + name
    }
```

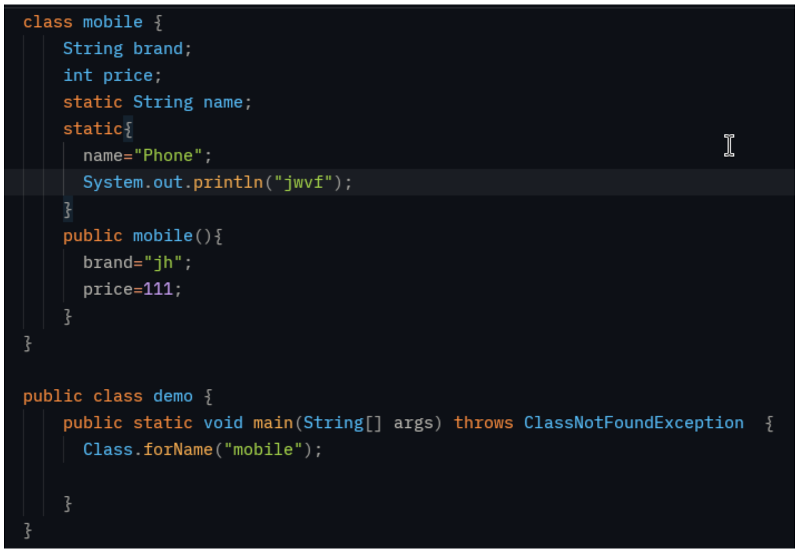
Static Block: *Static block(line 9;) only calls once. No matter how many object created. But constructor will be called every time obj created. Below:*



```java
 3  class Mobile
 4  {
 5      String brand;
 6      int price;
 7      static String name;
 8
 9      static
10      {
11          name = "Phone";
12      }
13
14      public Mobile()
15      {
16          brand = "";
17          price = 200;
18      }
```

Which variable we don't want to change we can set inside static block.
We can set default value on constructor(Line 14;) (above).
Always static block will call firth then constructor.
An obj is called there are 2 steps. Class loads &   object instantiated. Class is only load once(while1st obj created).
Since class loads once, static block loads only once. So if we don't create an obj, no class loaded and not static block will be execute.

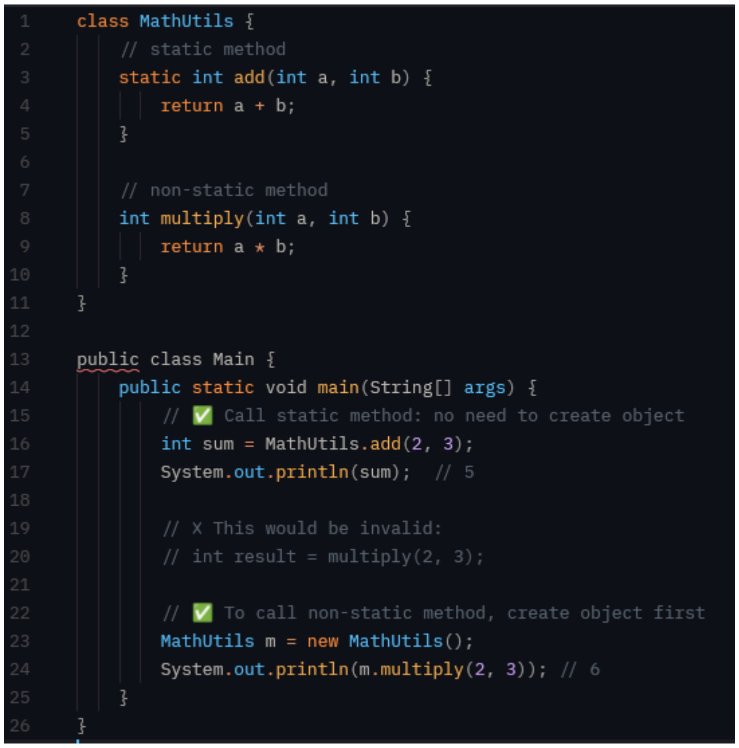
Without loading an obj we can load class. Below:



```java
class mobile {
    String brand;
    int price;
    static String name;
    static{
      name="Phone";
      System.out.println("jwvf");
    }
    public mobile(){
      brand="jh";
      price=111;
    }
}

public class demo {
    public static void main(String[] args) throws ClassNotFoundException  {
      Class.forName("mobile");

    }
}
```

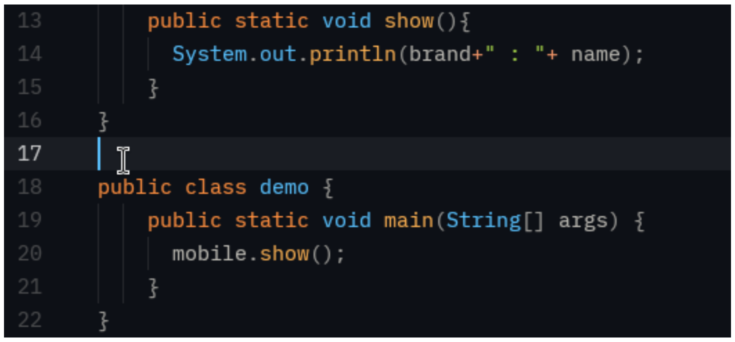
Here there is an exception. We will learn later.

Static Method: *No need to create object to call.*



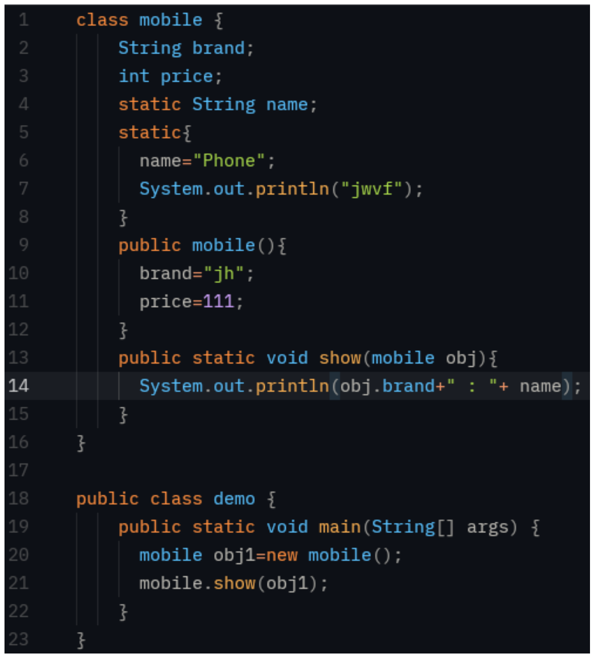
```java
class MathUtils {
    // static method
    static int add(int a, int b) {
        return a + b;
    }

    // non-static method
    int multiply(int a, int b) {
        return a * b;
    }
}

public class Main {
    public static void main(String[] args) {
        // ✅ Call static method: no need to create object
        int sum = MathUtils.add(2, 3);
        System.out.println(sum);   // 5

        // X This would be invalid:
        // int result = multiply(2, 3);

        // ✅ To call non-static method, create object first
        MathUtils m = new MathUtils();
        System.out.println(m.multiply(2, 3)); // 6
    }
}
```

Non-static variable can't use in static variable. Below: line 14;



```java
13      public static void show(){
14        System.out.println(brand+" : "+ name);
15      }
16  }
17
18  public class demo {
19      public static void main(String[] args) {
20        mobile.show();
21      }
22  }
```

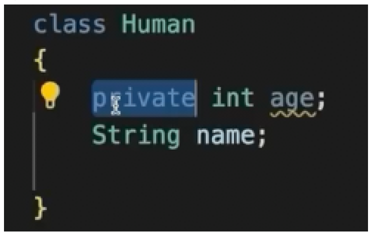
So if we want to use non-static variable in static variable we can pass object as an argument.



```java
class mobile {
    String brand;
    int price;
    static String name;
    static{
      name="Phone";
      System.out.println("jwvf");
    }
    public mobile(){
      brand="jh";
      price=111;
    }
    public static void show(mobile obj){
      System.out.println(obj.brand+" : "+ name);
    }
}

public class demo {
    public static void main(String[] args) {
      mobile obj1=new mobile();
      mobile.show(obj1);
    }
}
```

****Q: So why do main method "static" keyword?*
Ans: if main method is non-static, to call main we need to create a an object of class to call main. So main is not executed how can we create an object of demo. That's why static.

Encapsulation: *Encapsulation(data hiding) means somethings we don't want someone to use it from outside.*



```java
class Human
{
    private int age;
    String name;

}
```

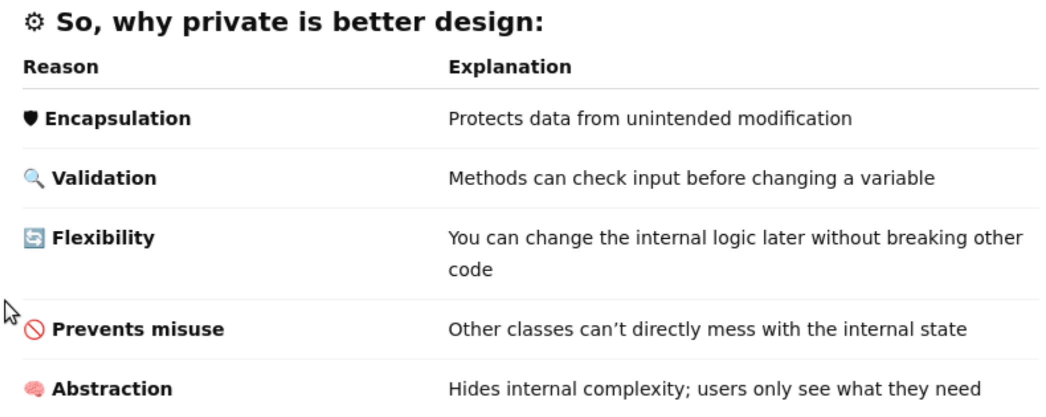
Using "Private" this variable is accessible inside class. Instance variable should be private. This type of date can be accessible with method inside that class.

🧩 Think of it like this (real-world example):

Imagine your class as a **vending machine**:

- The **snacks** (data) are *inside* — you can't just grab them directly (private).
- You use **buttons** (methods) — "Get Snack A", "Insert Coin".
- The machine decides whether you can get it (maybe if you pay enough).

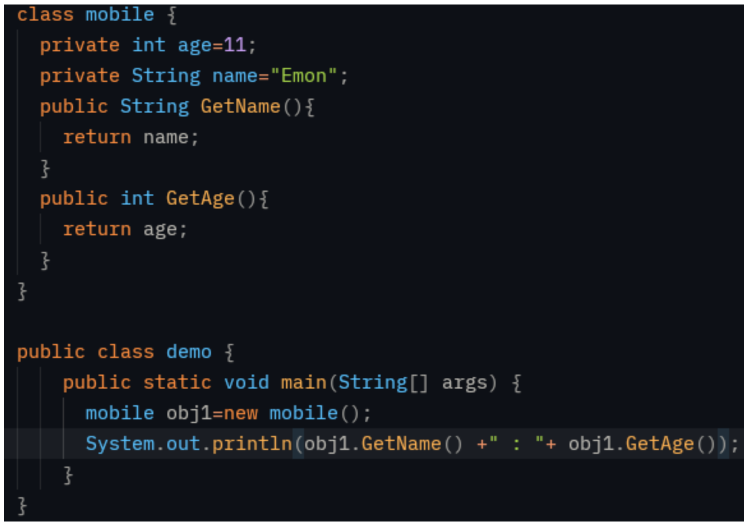
You *access* the snacks indirectly, but in a **safe, controlled way**.



⚙ So, why private is better design:

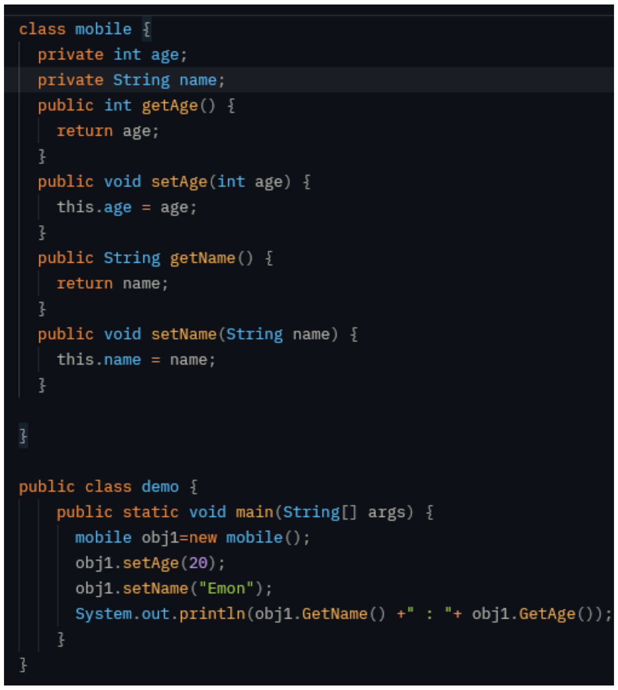
Reason	Explanation
🛡 **Encapsulation**	Protects data from unintended modification
🔍 **Validation**	Methods can check input before changing a variable
🔄 **Flexibility**	You can change the internal logic later without breaking other code
🚫 **Prevents misuse**	Other classes can't directly mess with the internal state
🧠 **Abstraction**	Hides internal complexity; users only see what they need

To control access we can use private variable and access with methods.



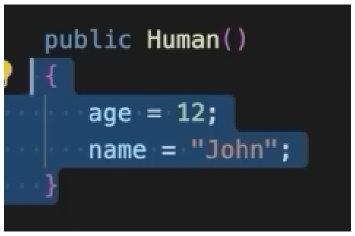
```java
class mobile {
  private int age=11;
  private String name="Emon";
  public String GetName(){
    return name;
  }
  public int GetAge(){
    return age;
  }
}

public class demo {
    public static void main(String[] args) {
      mobile obj1=new mobile();
      System.out.println(obj1.GetName() +" : "+ obj1.GetAge());
    }
}
```

To getting these data creating a method called "getter".
To set the data of a private variable by a method called "Setter". All same just create a method which accept data as argument and assign it to the variable. "This" is a keyword represents current object.



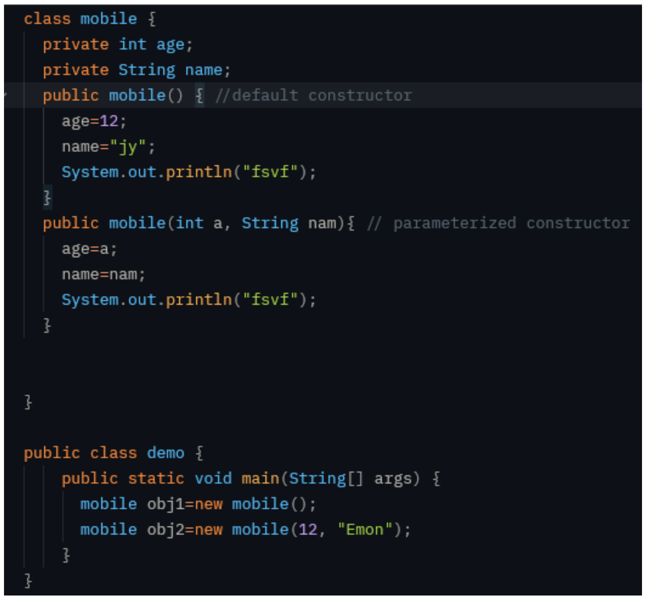
```java
class mobile {
  private int age;
  private String name;
  public int getAge() {
    return age;
  }
  public void setAge(int age) {
    this.age = age;
  }
  public String getName() {
    return name;
  }
  public void setName(String name) {
    this.name = name;
  }

}

public class demo {
    public static void main(String[] args) {
      mobile obj1=new mobile();
      obj1.setAge(20);
      obj1.setName("Emon");
      System.out.println(obj1.GetName() +" : "+ obj1.GetAge());
    }
}
```

**** Every method has its own stack.*
Constructor: *Is a special kind of method which is being called during obj is created. Method with class name. No return value, not even "void". We can use to set default value.*



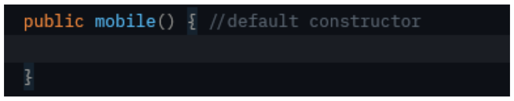
```
public Human()
{
    age = 12;
    name = "John";
}
```

We can create another constructor which will accept some value. Called Parameterized constructor. (above is default constructor).



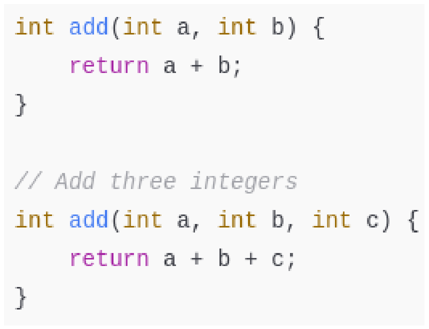
```
class mobile {
  private int age;
  private String name;
  public mobile() { //default constructor
    age=12;
    name="jy";
    System.out.println("fsvf");
  }
  public mobile(int a, String nam){ // parameterized constructor
    age=a;
    name=nam;
    System.out.println("fsvf");
  }



}

public class demo {
    public static void main(String[] args) {
      mobile obj1=new mobile();
      mobile obj2=new mobile(12, "Emon");
    }
}
```

If no constructor mentioned implicitly java will create a blank default constructor.
Use: set connection bet^n application and database.



```
public mobile() { //default constructor

}
```

Method overloading: *Method can be same name but parameter type/size/order must be diff. Above constructors are example of method overloading. Usually it mention for normal method/function.*



```
int add(int a, int b) {
    return a + b;
}

// Add three integers
int add(int a, int b, int c) {
    return a + b + c;
}
```