**JAVA NOTES**

**PROGRAMMING LANGUAGE**

* Medium of communication between human and machines

**TYPES**

1. **Low Level ->** That can be understandable by machine e.g.->binary.
2. **Mid Level->** Which has a few keywords called mnemonics, we need assembler to convert mnemonics into machine understandable language.
3. **High Level->**Easily understandable by humans JDK

JRE

Src.java 🡪compiler 🡪src.class (bytecode) 🡪 JVM (java virtual machine)

JVM

compiler

* JVM (java virtual machine) -> converts bytecode into machine understandable language
* JRE (java runtime environment) -> contains JVM and libraries
* JDK (java development kit) -> JRE and compiler
* Compiler-> Runs the whole code at once and givers all errors during compile time
* Interpreter->Runs the code line by line and gives one error at a time during runtime

**Features of java**

1. Platform Independent
2. Secure (we cannot get the actual memory address)
3. Muti-threaded
4. Supports OOPs
5. Open source

**Structure of Java Program**

class <classname>

* We can either declare or initialise something in a class

**To execute any java program**

1. Javac filename.java (src to compile)
2. Java classname (bytecode to jvm)
3. Java <filename>.java ->can directly run program

* To create a class file, it is not mandatory to have a main method
* Class name and file name should be same (conventional not rule)
* Compiler will create one class file whose name will be same as class name
* To run any java program MAIN METHOD is required

**Introduction to main method**

class Demo {

public static void main(String [] args){

System.out.println(“haha”)

}

}

**TOKENS**

* Smallest unit in any programming language

1. Keyword
2. Identifier
3. Literals
4. Variables
5. Operators

**1.KEYWORD**

* Predefined values that are understandable by java compiler.(50+)

**2.IDENTIFIER**

* Name given to the java components
* Class, variable, method, interface, enum

**Naming convention**

* Identifiers cannot start with numeric value
* Only \_ and $ these 2 special characters are allowed while giving the identifier (they don’t have any specific meaning)
* We cannot use a keyword to create an identifier

**PASCAL CASE CONVENTION**

* Class name should always start with capital letter
* If your class name is combination of more than 2 words then first letter of each word should be capital (TextBook)

**3.LITERALS**

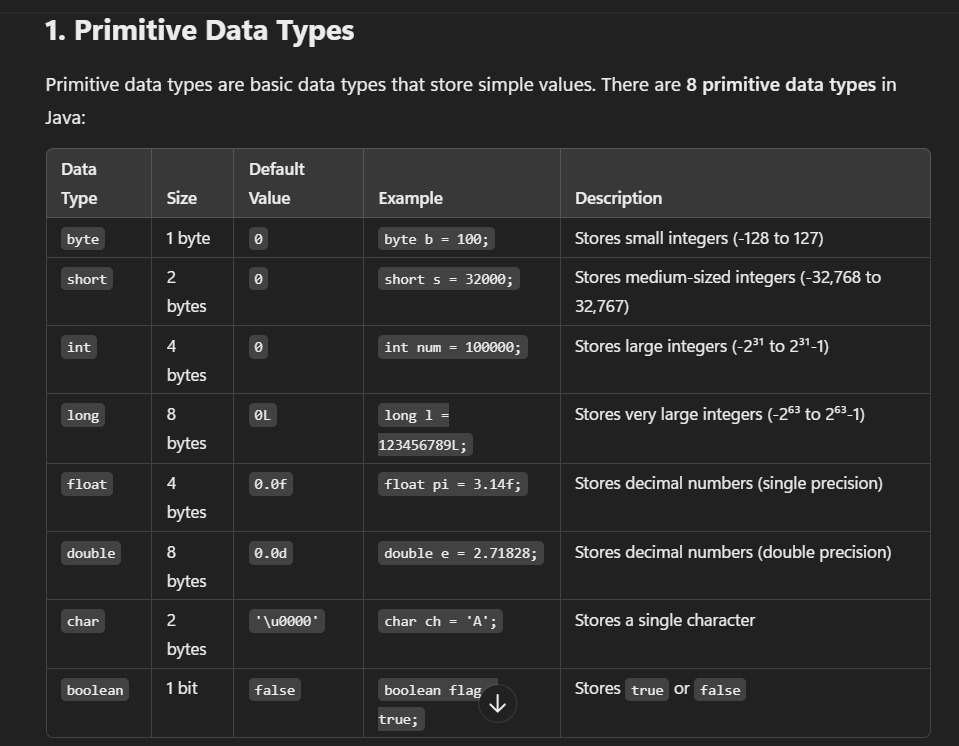
* Data used inside a program e.g. -> int a=10(here 10 is literal)

1. Number (10, 10.21)
2. Char (‘A’, length is always 1)
3. String(“hello”)
4. Boolean (true or false)

**4.VARIABLES**

* int num = 10
* Named block of memory
* Syntax🡪 datatype <variable name>;
* We can do initialisation, declaration and reinitialization (int a=10; a=20)
* We cannot declare 2 variables with same name (int num=10; int num=20; )

**#Datatypes:**

****

* They are predefined datatypes.
* Java doesn’t work with ASCII, it works with unicode(‘\u0000’).
* We can use 65535 characters(2 bytes) in java whereas in C/C++ we can only use 256 characters (1 byte).
* We cannot store negative value in char.
* In java size of Boolean is not precisely defined.
* We can only assign Boolean values as true/false, they are not keywords rather Boolean literals.

**4.2.Non-Primitive Datatype**

* It is a user defined datatype
* Class(string), interface, array
* Use of non-primitive datatype is to create a non-primitive variable to store a non-primitive value

**Ques🡪Can we create a variable with non-primitive datatype?**

* YES, class Demo{} Demo bag; we can store 1.null 2.object of bag

Non-primitive Non-primitive variable

**#Scope:**

**Local Variable**

* Variable created in any other block than class block

**Characteristics of local variable**

* We can access local variable only inside its own block.
* We cannot create 2 same local variables inside the same locality or inside the same scope.
* For local variable default values are not there.

**5.OPERATORS**

* Operators are special symbols that have predefined task.

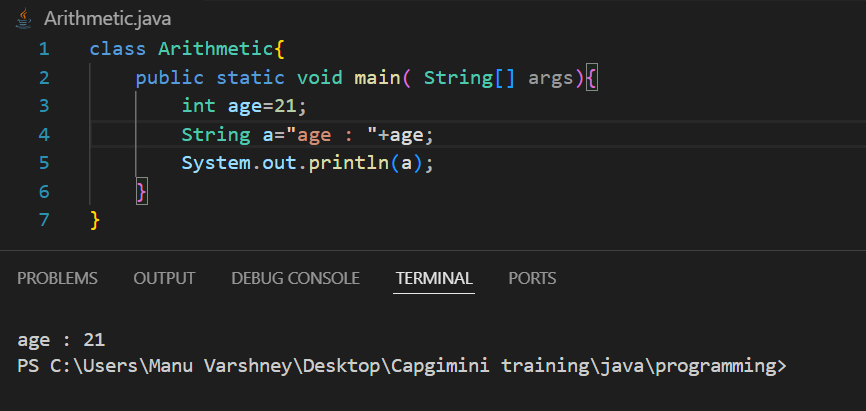
**Expression🡪**Combination of operator and operand.

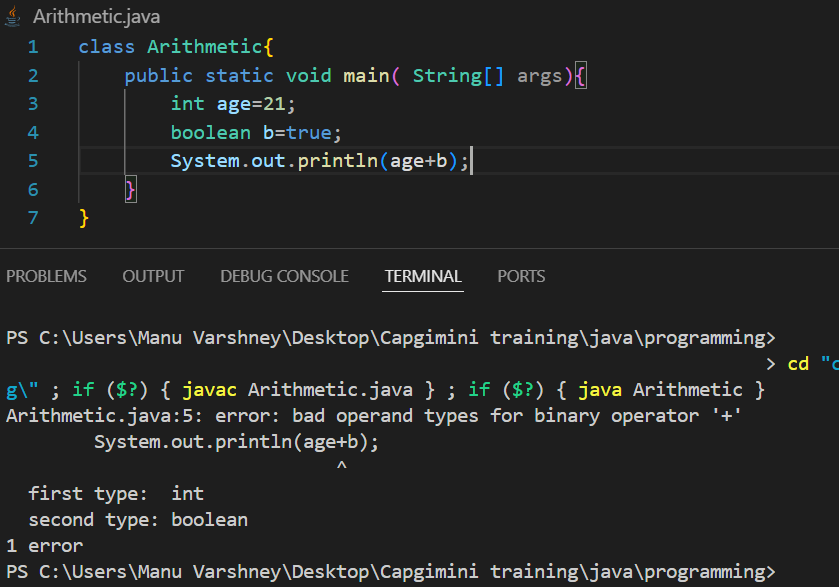
**Precedence 🡪** Priority of execution.

**Associativity 🡪** Direction of execution.

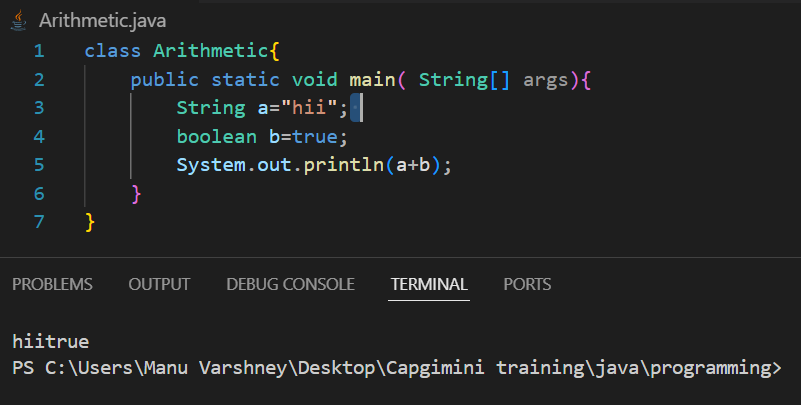
**5.1.Arithmetic Operators**

* +, -, \*, /, %
* ‘+’, It will perform two tasks
  + - When both operands are numbers then it will perform addition.
    - If one of the operands is string it will perform concatenation.

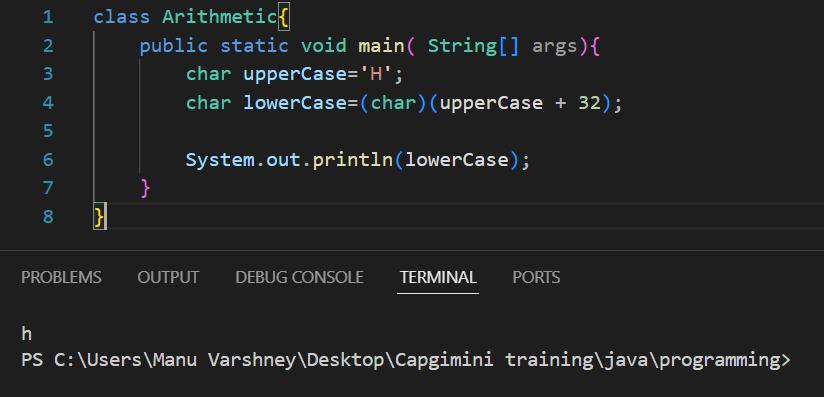


**Number + Boolean = compile time error**  

**String + Boolean = Concatenation (Boolean will be converted into string)**



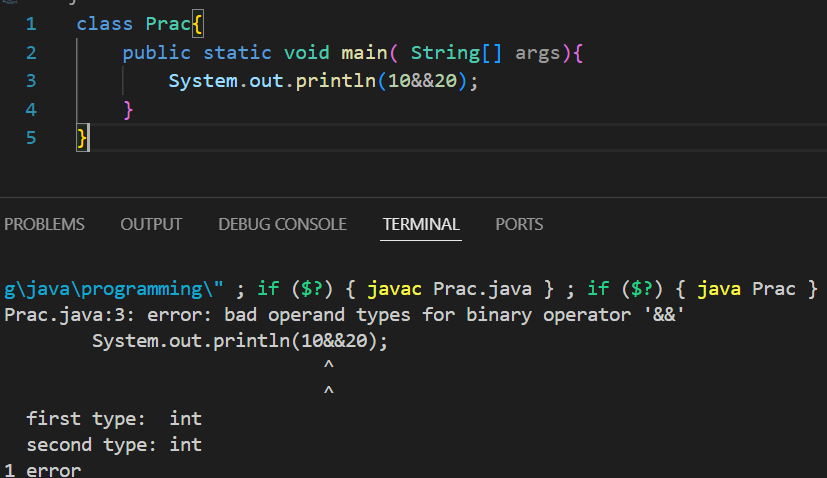
* **From uppercase to lower case +32 and vice versa.**

****

**5.2.Realational Operators**

* >, <, >=, <=, ==, !=.
* They always return Boolean value.

**5.3.Logical Operators**

* &&, ||, !
* They only work with Boolean value. 

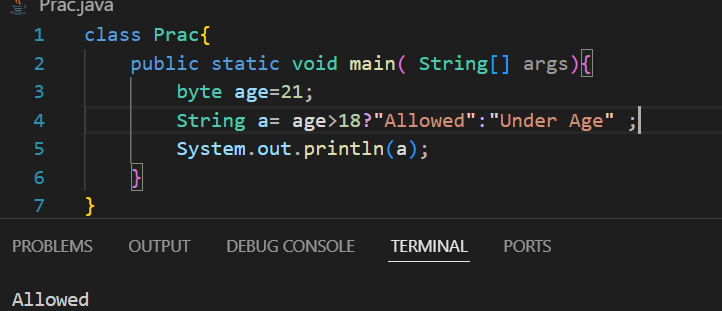
**5.4.Conditional Operators**

* ?, ; (ternary operator)

false

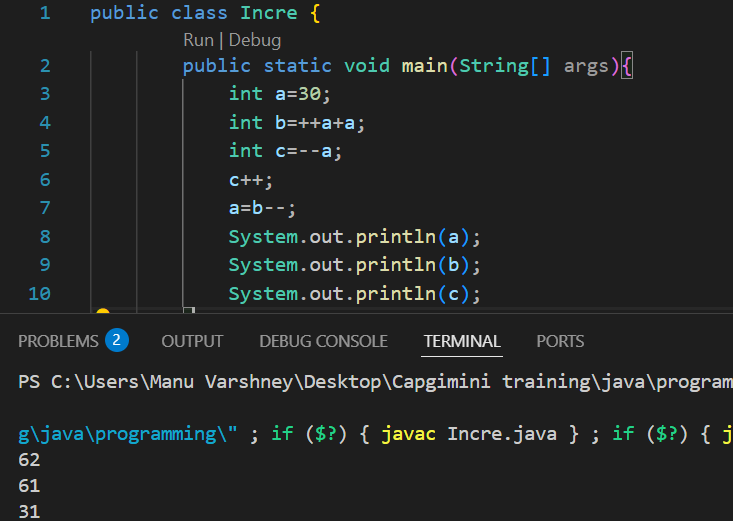
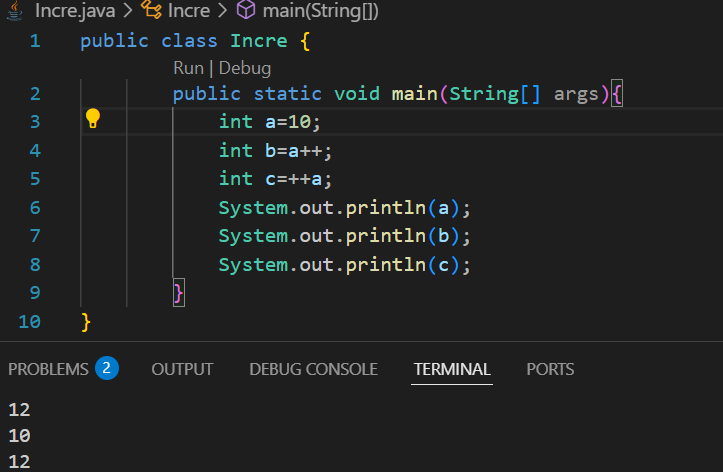
* Condition ? exp/value : exp/value

true

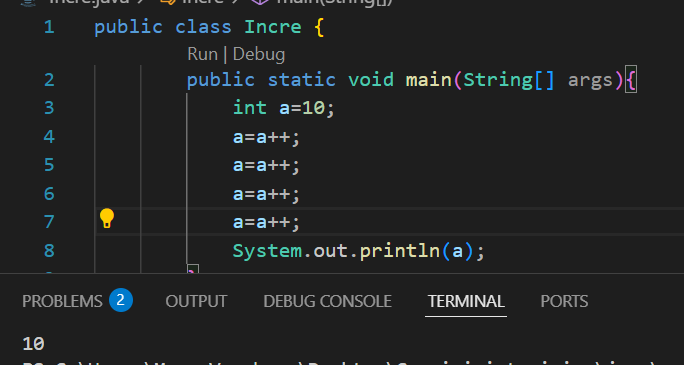


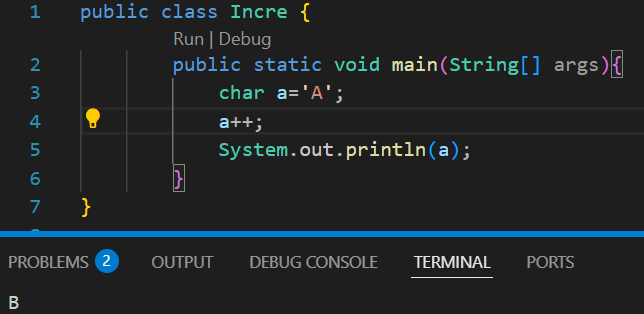
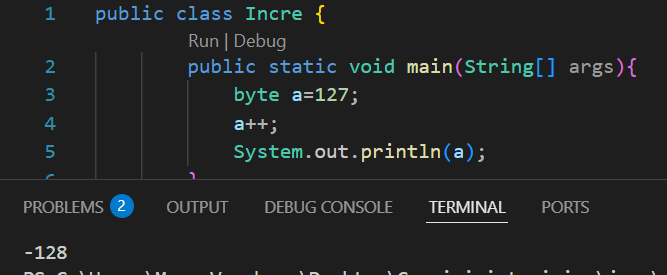
**5.5.Increment & Decrement Operators**

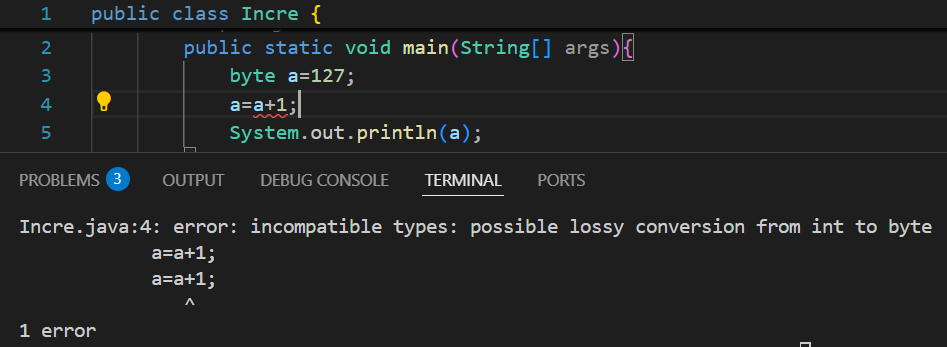
* ++, --
* We can only use them with variables(not with constant values).

****

* **Special case🡪 If we post increment or decrement a variable and store it in the same variable then its value will remain the same .**

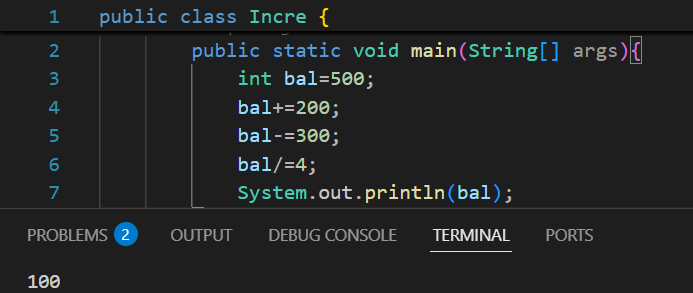


****

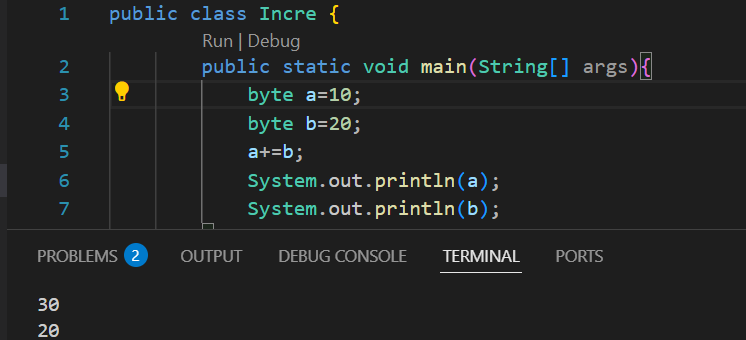


**5.6.Compound Assignment Operators**

* +=, -=, \*=, /=
* Combination of arithmetic and assignment operators.



* **In compound Assignment operator we will not get error while performing any arithmetic operation on byte as it converts the result into its respective form**

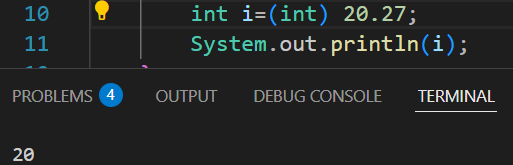
****

**Widening🡪**

* It is performed to convert smaller datatype into larger e.g.->int to double.
* It can be performed implicitly as well as explicitly.

**Narrowing**🡪

* Converting large datatype into smaller datatype.
* Compiler is not allowed to do narrowing **because of data loss**.
* We use Cast Operator to do explicitly type casting, <datatype>.



**METHOD(Function)**

* It is **set of code or block of information** that is used to perform a task.
* Modifiers return\_type name(formal arguments){

----------------------

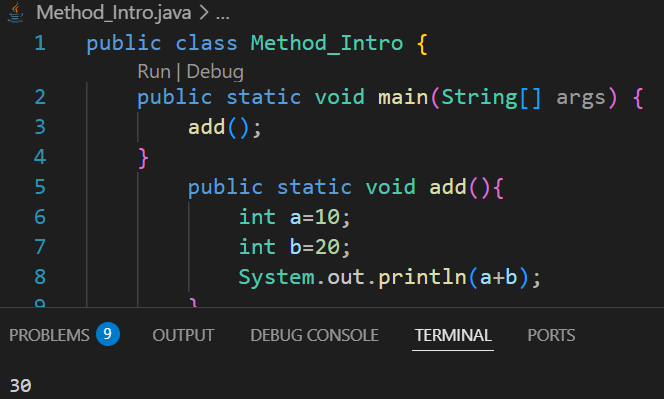
----------------------

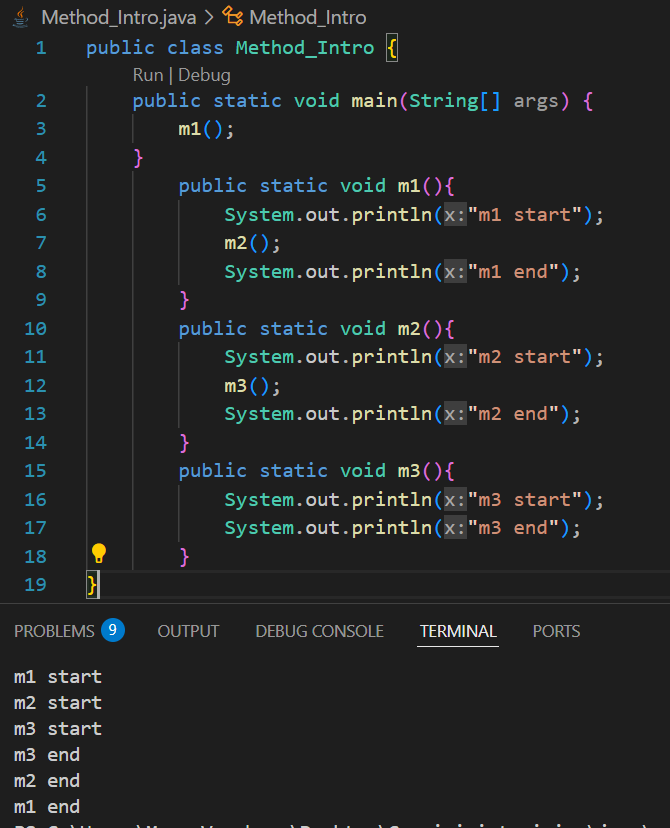
Return statement

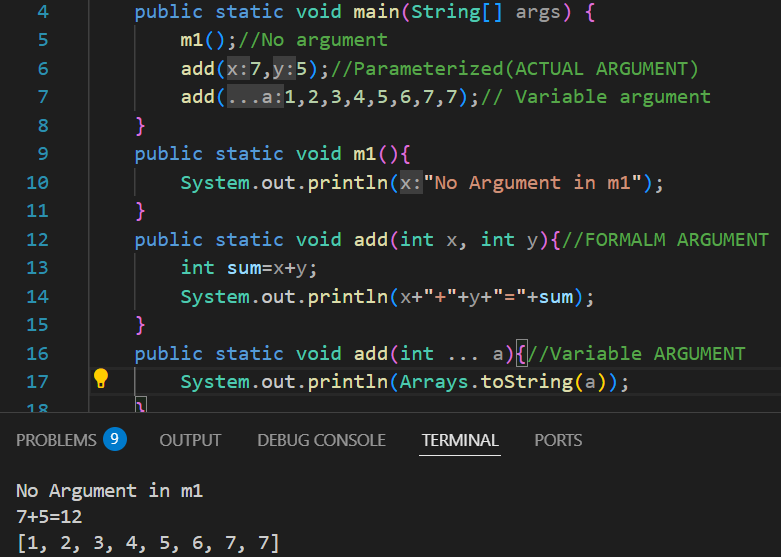
}

Modifiers, formal arguments and return statement are optional

* If we are not using any of the access modifier, its scope will be Default🡪Package.
* If access modifier is **Public**, the member (variable, method, or class) can be accessed from anywhere in the program.
* If the access modifier is **Private**, the member (variable, method, or class) can only be accessed within the same class in which it is declared. (We can access protected member outside the class with the help of reflection API)
* If the access modifier is **Protected**, we can only access inside the package. (We can access protected member outside the package, for that we need to use inheritance)
* We cannot create a method with a default keyword inside a class.
* If at the place of **Return type** there is **void**, method will not return anything, it will return the control.
* **Name+(Formal arguments)** it is called as **Method Signature**.
* **Modifiers + Return type + Method signature,** it is called as **Method Declaration.**
* **{}** are called **Method body/block/implementation.**
* **Method Declaration+ Method Block,** it is called as **Method Definition.**

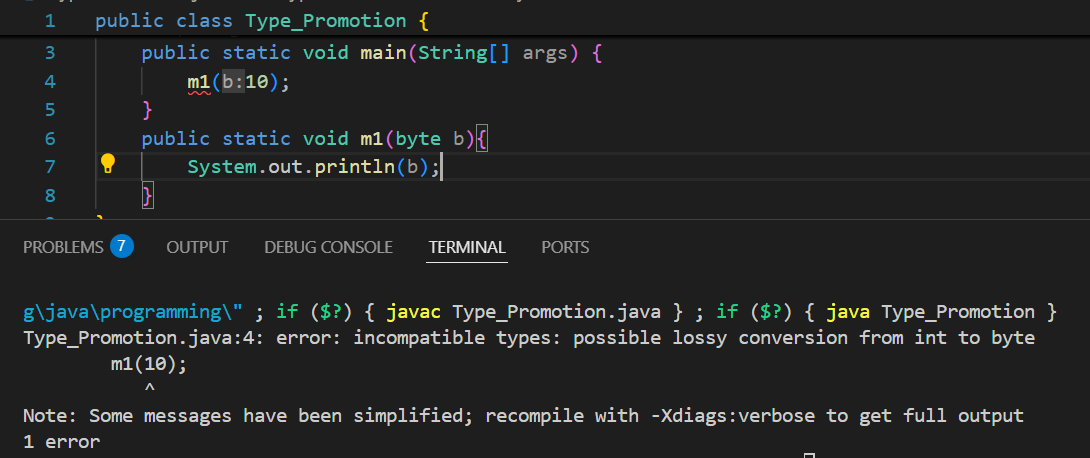


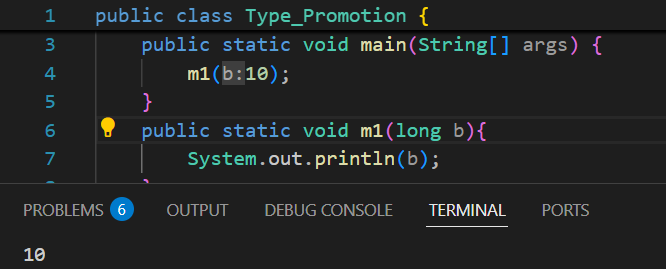




Variable argument🡪 It creates array when used in formal arguments, we can enter n number of values as per our requirements (e.g.--> public static int add(int … num))

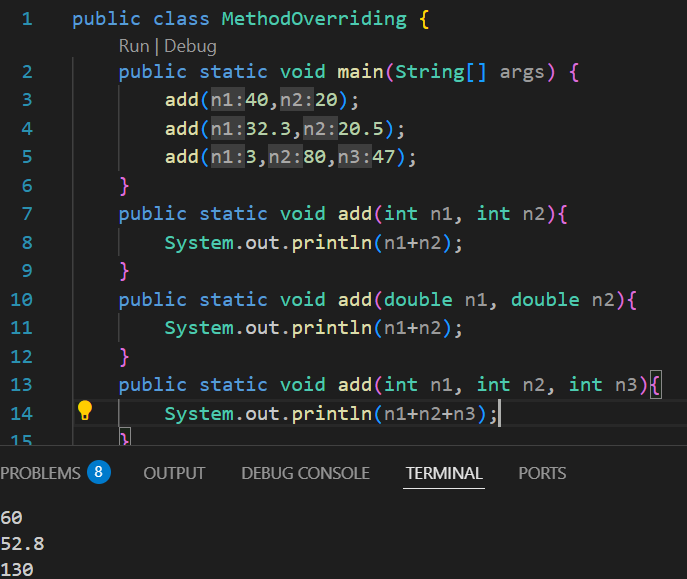
* **Actual Arguments🡪**The data that we are going to pass while calling the method.
* Length and datatype of formal and actual argument should be same.
* Whenever method with the same datatype is not present in the class, then we go for type promotion (searching for data which have higher datatype).





**METHOD OVERLOADING**

* If inside a class we have method with same name and different formal arguments.
* It is also called compile time or early binding because it will be done at compile time.



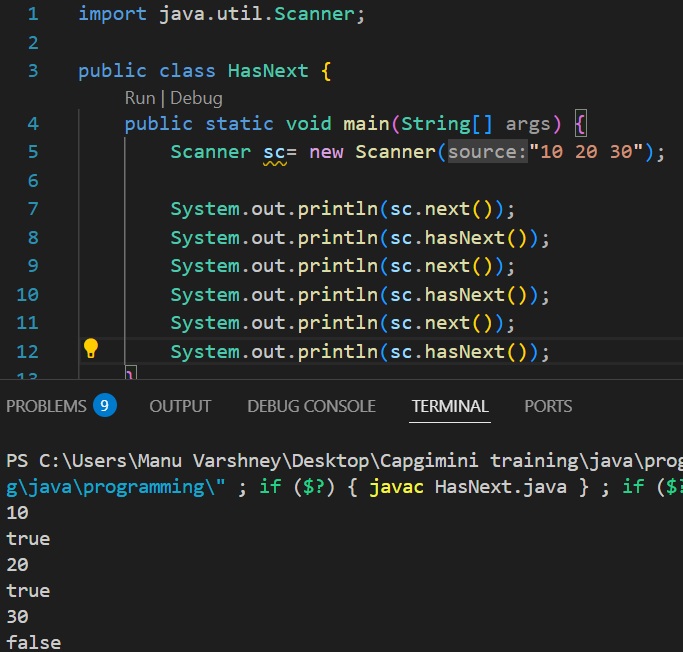
* **Println is one of the best example of Method Overloading**

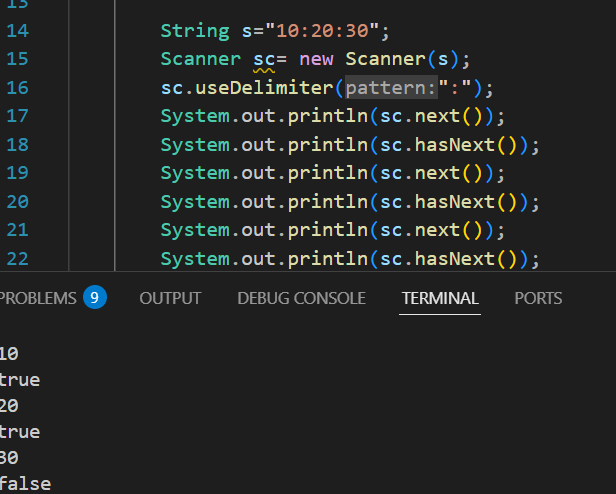
**Ques🡪How to perform method overloading?**

* Method name should be same
* Formal Arguments should be different
  + Length
  + Type
  + Sequence

**Taking String Input**

* **Scanner.next()** will take input before space only(e.g.name is, only name will be taken)
* **Scanner.nextLine()** will take the whole string (but if the carriage is empty it will take empty only ass imput).
* We need to add one more Scanner.nextLine() before taking input as string**.**
* **hasNext()** will return Boolean data, it will check if data is present or not.
* Default delimiter is **“Space”.**
* We can change the delimiter by using **Scanner.useDelimiter(“:”);**





**Static Method**

* A method which has static keyword in the method declaration.

**Ques🡪How to call static method inside the same class.**

🡪1.We can call directly m1();

2.By using class name as a reference myclass.m1();

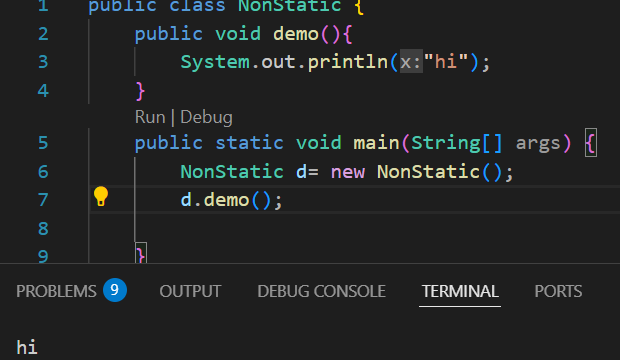
3. By object creation.

**Ques🡪How to call static method from one class to another class.**

🡪 By using class name as a reference myclass.m1();

**Non-Static Method**

* If we want to call a non static class then we have to create a class object.



**Decision Making Statements**

* With the help of decision making statements, programmer can decide which part of code is to be executed and which part should be skipped.
* Inside **Switch** certain datatypes are not allowed.

1. Boolean
2. Double
3. Float
4. Long

**Break Statement**

* Break is a control transfer keyword which will transfer the control out of the block.
* We can use break keyword only **inside switch and looping statement**.

**Enhanced Switch (Switch as an expression)**

* There is no need of break statement.
* Default is mandatory.
* If we have multiple lines in a case then we cannot use {}, we need to use **yield** <statement>.
* Writing default is not mandatory in some cases, these scenarios are related to enum and sealed classes.
* Case value should be constant, it cannot be a variable.
* Switch is faster than else if ladder.
* Final variable is allowed as a case value
* If we have multiple cases that perform same task then we can use “,” e.g. case 1,2,3

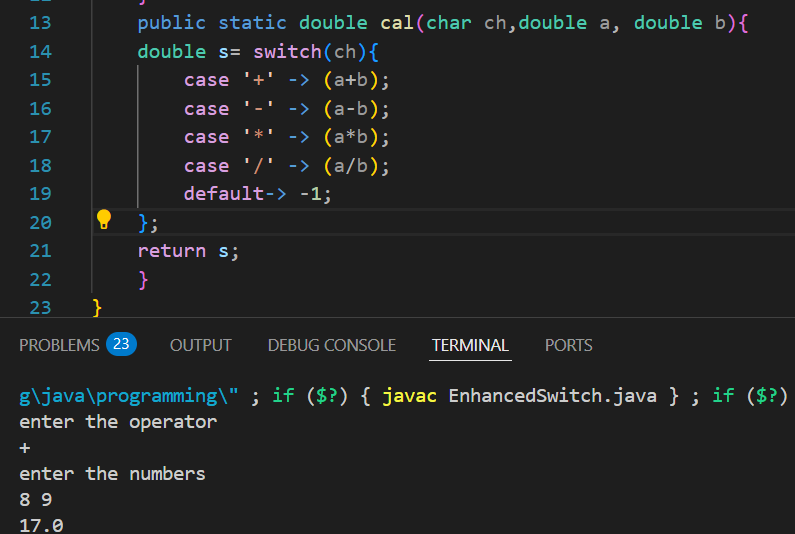
String s= switch( ){

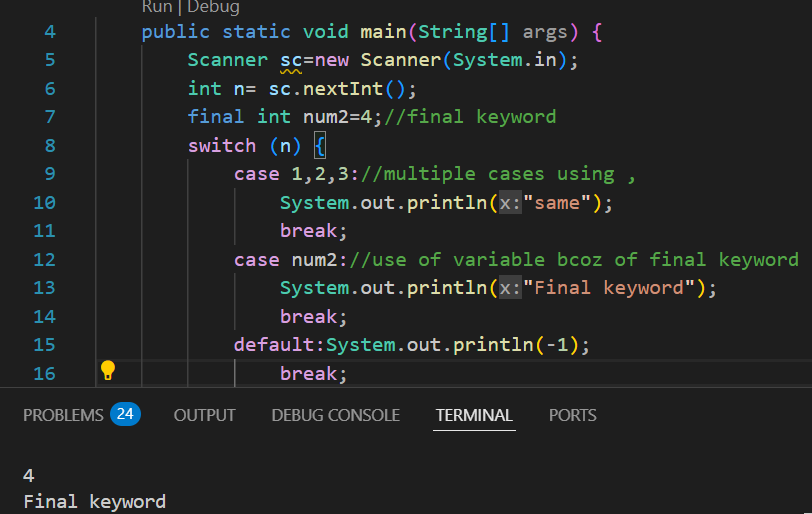
Case v1 -> “1”;

Case v1 -> “1”;

Default-> “0”;

**};**

****

****

**LOOPS**

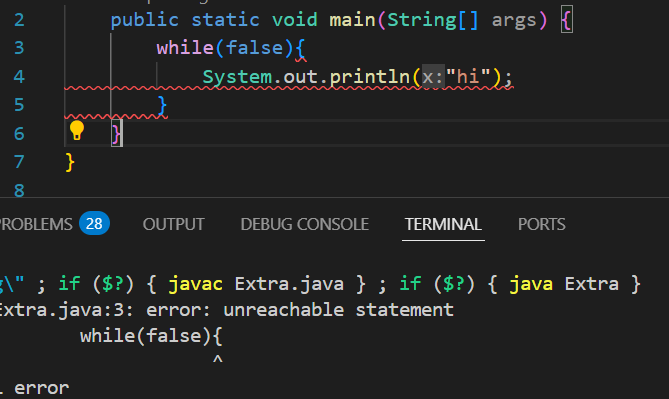
1. **While Loop**

**While (condition){**

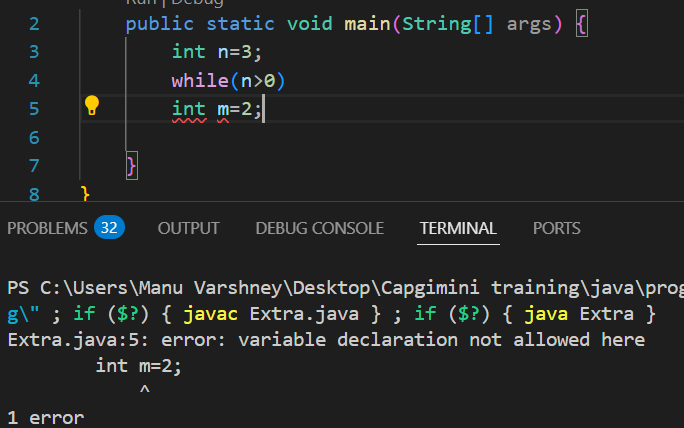
**Statements**

**// update (not a part of syntax)**

}



* **If we are not using any curly{} braces then we cannot declare inside a while loop**.



1. **For each Loop/Enhanced for loop**

**for(datatype name : array/collection){**

**Statements** }

e.g.--> int[] arr={1,2,3};

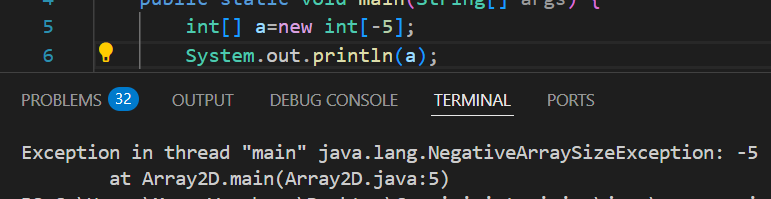
for(int a: arr){

sout(a);}

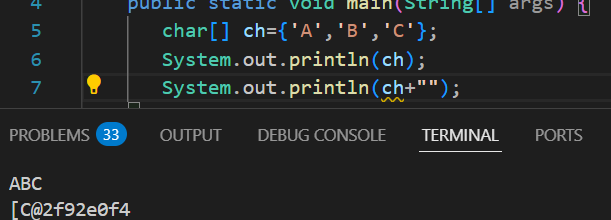
* We cannot traverse in return.

**ARRAYS**

* They are homogeneous i.e. we can only store one type of datatype.
* We cannot increase or decrease the size of array at runtime.
* Syntax 🡪 <datatype> [] <variable name> = new <datatype>[size] (int[] arr =new int[5]).
* int[] arr/ int arr[] both are correct.
* **New keyword** will create a block of memory inside heap area.
* **Arrays.equals(a1,a2)** to check whether 2 arrays are exactly same.
* **When we pass the length of array as -ve, it will give runtime error (size exception).**



* **When we will print the address of char array it will return the array only (because println(char) is overloaded).**



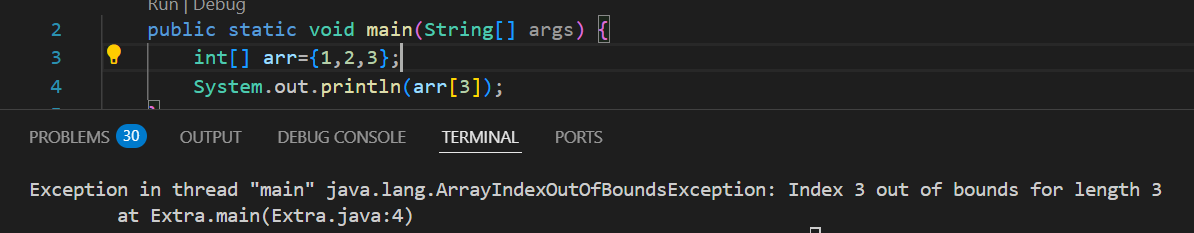
**Int[] array={1,2,3};**

**Int[] ball=array;**

**Ball[0]=1000;**

**Sout(array[0]);**

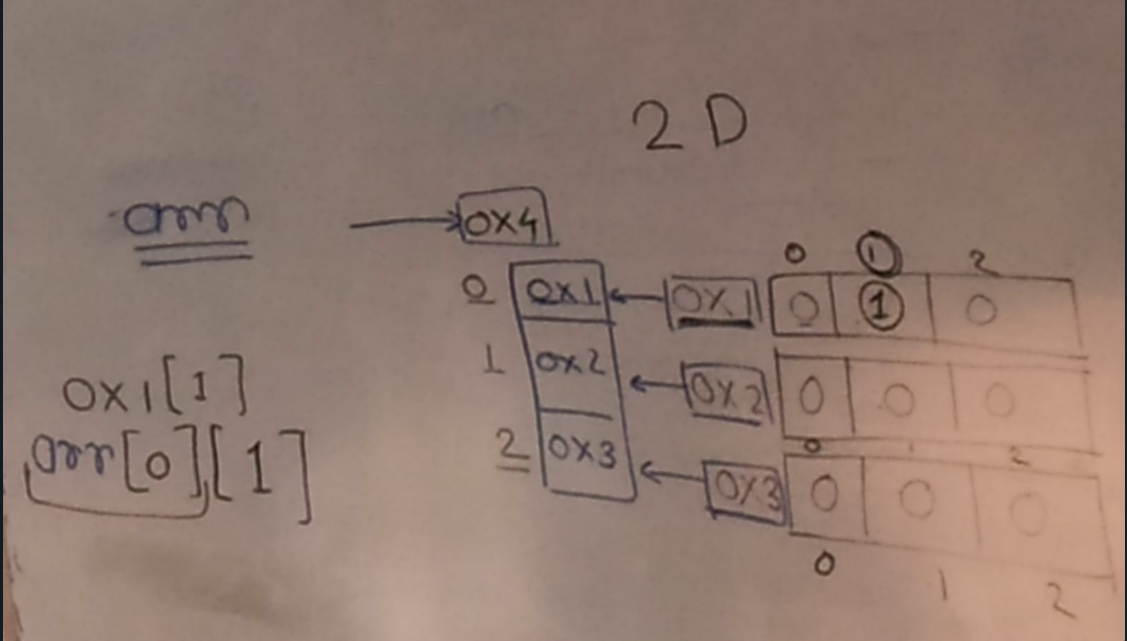
* **It will return 1000 even we have changed the value in bal array bcoz here we have copied by reference.**

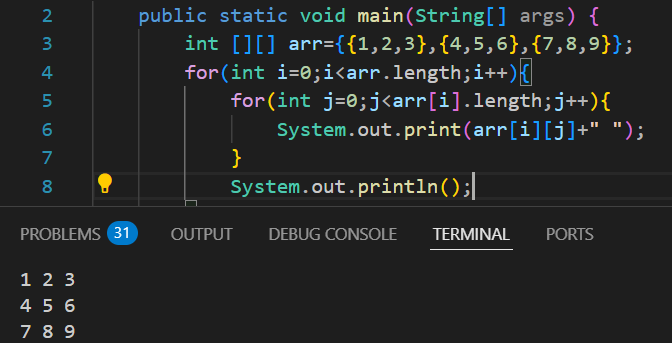


* **If we want to print array without using loops** 🡪 **sout(Arrays.toString(arr));**

**2D array**

* **Syntax 🡪 datatype [] [] variable = new datatype [rows] [cols];**
* Datatype[] [] name = { {arr1}, {arr2}, {arr3} };
* Printing directly🡪 **System.out.println(Arrays.deepToString(arr));**
* If the arrays inside 2D array are of different sizes, it is called **Jacked array**.

****

****

**OBJECT**

Variables

* When we want to convert real world entity into virtual world.
* In general object is a real world entity and in technical terms it is a block of memory.
* We can store heterogenous type of data (variables and methods).

Method

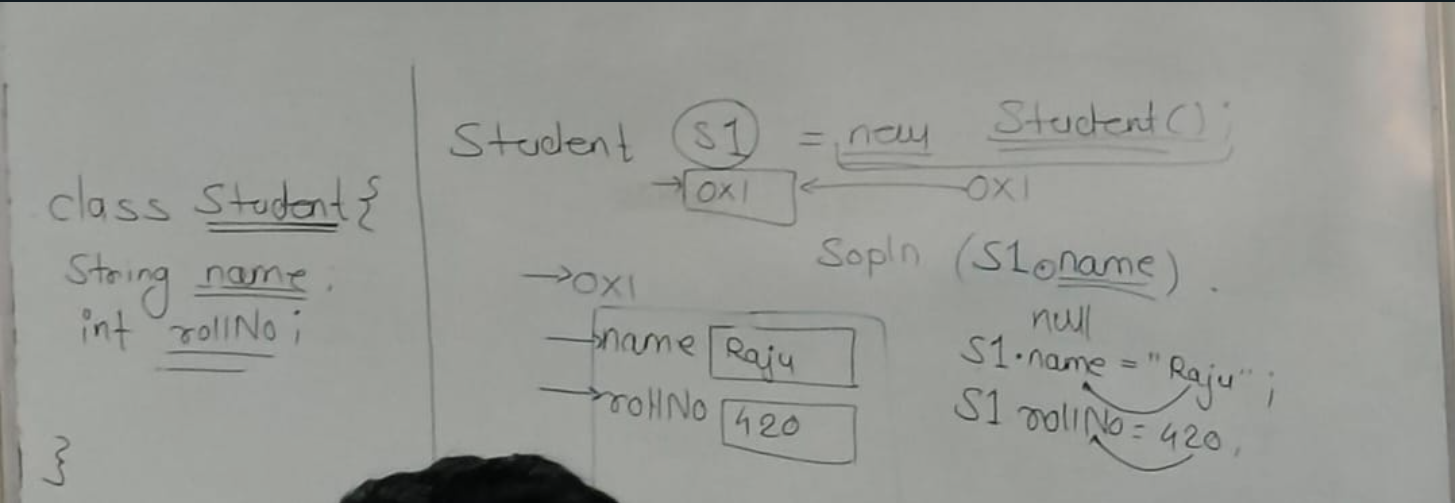
* Every object has two parts🡪

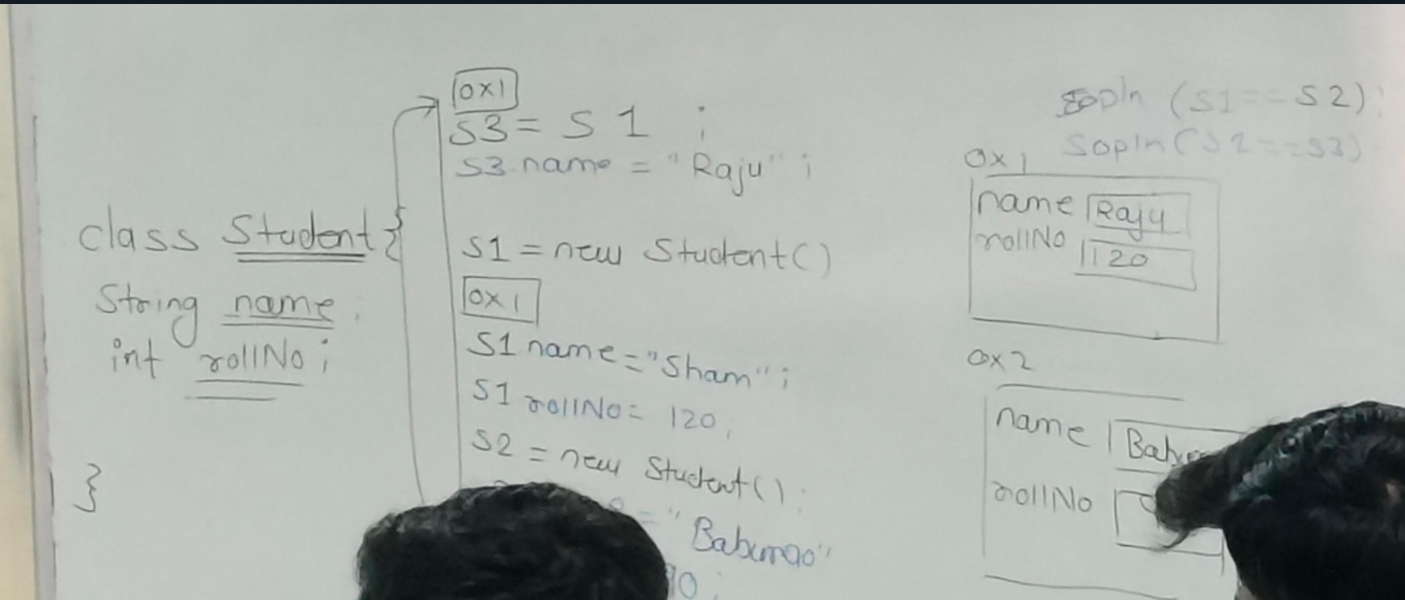
1. Information related to object (properties, fields and states.)
2. Action performed by the object

* If we want to create an object, we need to first create a class.
* **Class**

1. To execute a program
2. To create an object

* Syntax🡪 new constructor();
* Student s1=new student();🡪 This s1 is non-primitive variable also called **object reference variable.** Inside this s1 there will be a address of the object.





**Non-Static Variables**

* Variable created inside a class block and not prefixed by the static keyword.
* To store the properties of the object we are going to use non-static variable.
* It will get loaded for every object creation.

**Constructors**

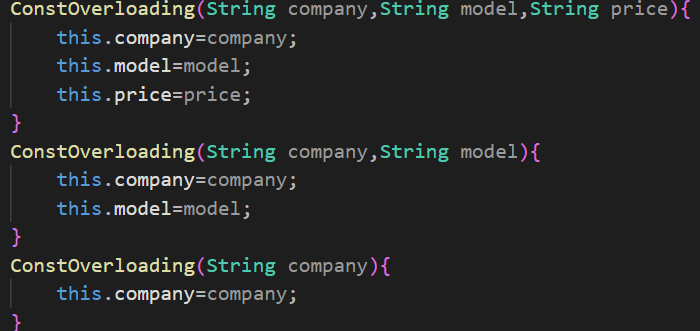
* It is a special member of a class which will load all the non-static variables and methods into the object.
* Syntax🡪 <**access modifiers> <class name> (parameters){}.**
* There is **no return type** in constructor otherwise it will become a method.
* **Default Constructor🡪**Wherever a programmer fails to declare a constructor inside a class at that time compiler will add one no argument constructor.
* **This** is a keyword which will hold the address of currently executing object (We cannot use this keyword in static context, static method).

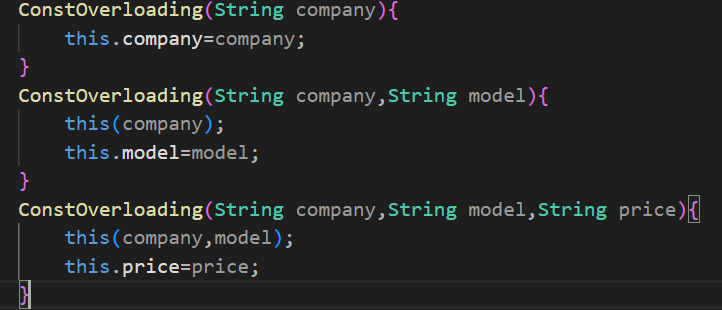
**Constructors Overloading**

* Creating different constructors inside a same class with different parameters.

**Constructors Call Statement**

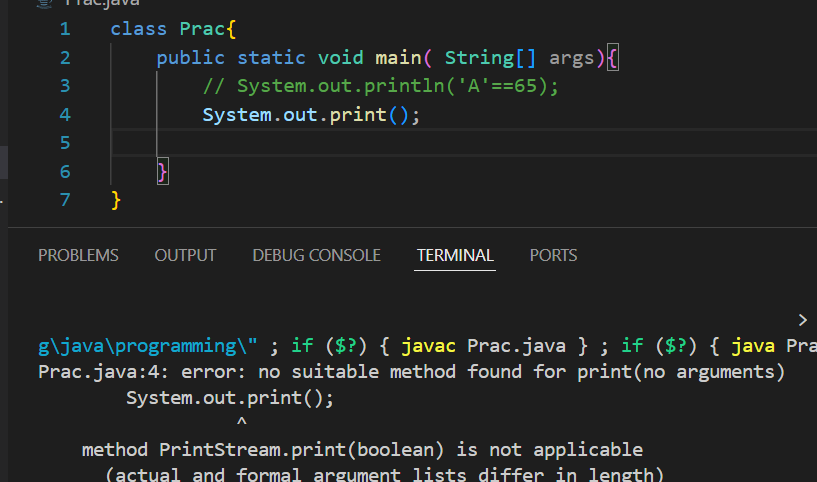
* this()🡪We can call a constructor in the same class.

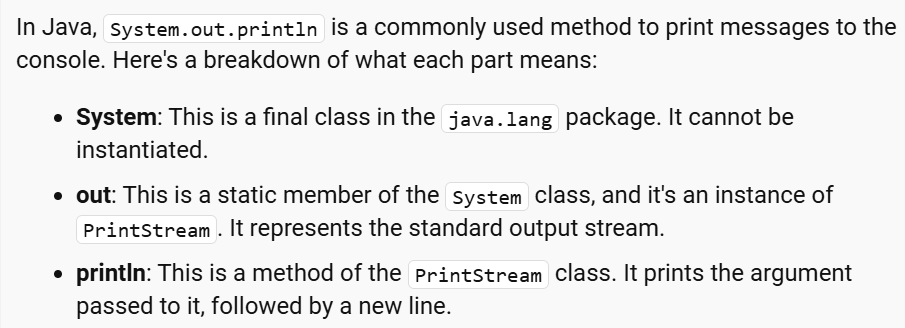


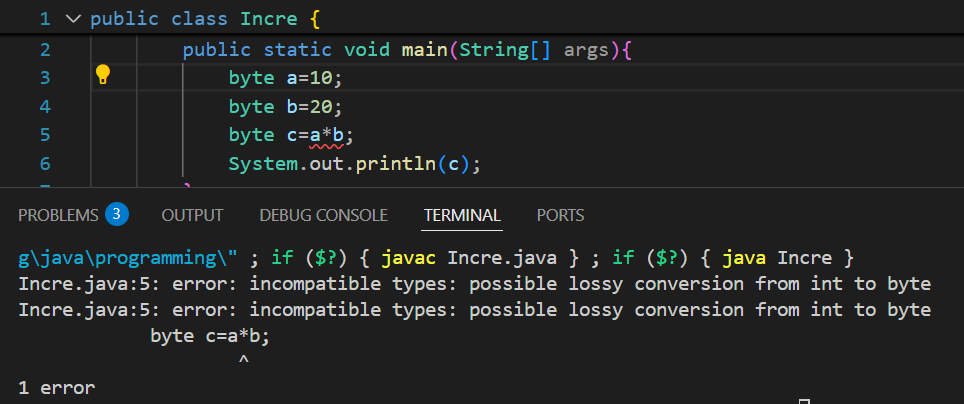


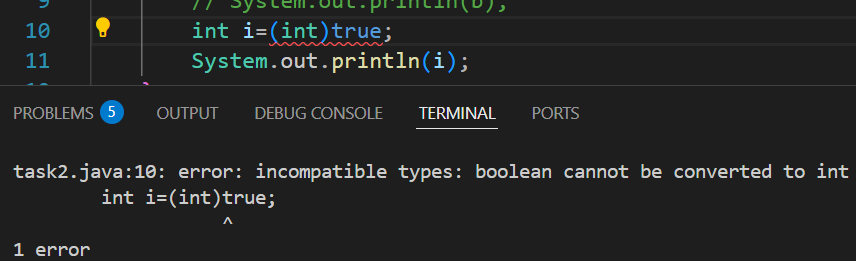
NOTES

1. From uppercase to lower case +32 and vice versa.
2. Print is used to print the output in single line
3. Print cannot work when there is no argument e.g. System.out.print();



1. System.out.println() 
2. If we perform arithmetic operation on byte datatype then it will show error as arithmetic operations returns integer values.



1. We cannot type caste Boolean to any other datatype or vise versa. 
2. We can convert primitive datatype into non-primitive and vise versa using Rapper classes.
3. char ch=97; will not give error as it lies in the range of char(65535), no loss of data.
4. Inside printing statement we can call a method which will return some data (if the return type is void we cannot call it in printing statement).
5. If return type is other than void then return statement is mandatory.
6. Return statement should be the last statement inside the block
7. Whatever the classes are presented in lang package, they are automatically added.

