OBJECT ORIENTED PROGRAMMING USING

Java

Let's explore technology
together to live in the future

Checkout more on
https://github.com/Sy-hash-collab

Sy-hash-collab

6/9/23

Printing in Java:

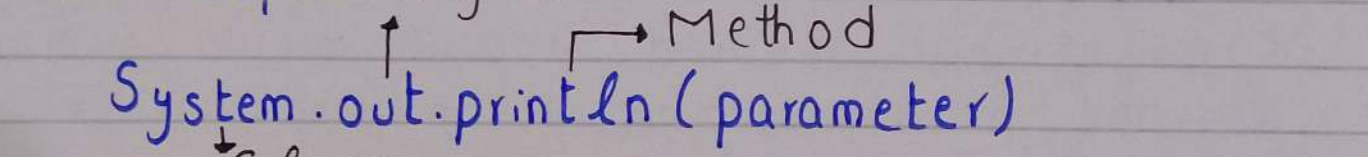
- System.out.println in Java:

System.out.println is used to print an argument that's passed to it.

System: It is a final class defined in the java.lang.package.

out: This is an instance of PrintStream type, which is a public and static member field of the System class.

println(): All instances of PrintStream class hav a public method println(), it prints any argument passed to it and adds a newline to the output.



* The parameter might be anything that the user wishes to print on output screen.

```
System.out.println("Hello")
```

* To print multiple variables in Java, we us '+' (concatenation)

```
System.out.println("Hello" + a + "ac")
```

* '+' has multiple forms in Java (Polymorphic).
* The name of file of program and class name is same.

Java main() Method:

Keyword ↑ Method Name ↑

public static void main (String args[])

↓ Access Specifier ↓ Return Type ↓ Array of string type

1. **Public:** It is an "access modifier", which specifies from where and who can access the method. Making the main() method public makes it globally available.
2. **Static:** It is a keyword that is when associated with a method, making it a class-related method.
3. **Void:** It is a keyword and is used to specify that method doesn't return anything. As the main() method doesn't return, its returntype is void.
4. **main:** It is the name of Java main method. It is the identifiers, as the startingpoint of java program. It's not a keyword.
5. **String [] args:** It stores Java command-line arguments and is an array of type "java.lang. String class. Here, the name of String array is "args" but it's not fixed, the user can use any name in place of it.

```
class GeeksforGeeks
{
    public static void main (String[] args)
    {
        System.out.println ("I am Geek");
    }
}
```

```
class Student
{
    int rollno. ;
    Static String deptName ;
    public static void main (String [] x)
        {
            System.out.println("Hello")
        }
}
```

- Main is a method, member of Student class main has one parameter 'x' whose type is "array of String".
- Student is a class, which has three members i-e
- Two data members : rollno. , deptName
- One Method : public static void main (String [] x)

12/9/23

Non-Primitive DataType:

Non-primitive datatypes are called reference types because they refer to objects.
Created by programmer, mostly user-defined.
All datatypes whose class exist which can be built in such as String either can be user-defined i-e Student.
It will have first letter in uppercase.
User-defined datatype is also called abstract
The variable of non-primitive datatype is called Object.

```
class Student
{
}
Student s = new Student()
        └ object
```

Function / Method call:

Non-primitive datatypes can be used to call methods to perform certain operations because the variable of non-primitive datatype is called object and object can be used to call the method or function in java.

```
void show()  ──→ method / function
{
  Integer x;  ──→ non-primitive datatype
              └─→ Object
  x.show()  ──→ Method call
}
```

TYPE CASTING:

The process of converting the value of one datatype (int, float, byte) to another datatype is typecasting.

- Type casting is when you assign a value of one - primitive datatype to another type.

=> Widening Casting:

In Widening Type Casting java automatically converts one datatype to another. Converting a smaller datatype to larger typesize

byte → short → char → int → long → float → double
1 2 2 4 8 4 8

- Lossless type casting. Safe casting.

```
print class Main{

    public static void main(String[] args){
        int myInt = 9;
        double myDouble = myInt  // automatic casting

        System.out.println(myInt);  // outputs 9
        System.out.println(myDouble); // outputs 9.0
```

In above example, we are assigning the **int** type variable named **myInt** to **double** type variable named **myDouble**.

Hence, java first converts the **int** type data into the **double** type. And then assign it to **double** variable.

The lower datatype (smaller size) converted to higher datatype (larger size). This is why this conversion happen automatically.

=> Narrowing Casting:

In Narrowing Type Casting, we manually convert one datatype into another using the parenthesis.

- Converting a larger type to a smaller size type

double → float → long → int → char → short → byte
8 4 8 4 2 2 1

- Narrowing casting must be done manually by placing the type in parentheses infront of value

```
public class Main{
  public static void main (String []args)
  {
    double myDouble = 9.78d;
    int myInt = (int)myDouble; // manual casting

    System.out.println (myDouble); // Output 9.78
    System.out.println (myInt);    // Output 9
```

In above example, we are assigning the **double** type variable named **myDouble** to an **int** type variable named **myInt**

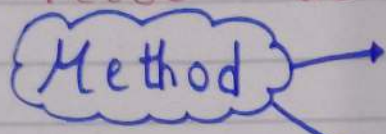
Notice the line; int myInt=(int)myDouble;

Hence, the **int** keyword inside parentheses indicates that the **myDouble** variable is converted into the **int** type

In case of this casting, the higher data types (having larger size) are converted into lower datatypes (having smaller size). Hence, there's the loss of data, can't happen automatically.

JAVA METHODS

Methods of java is a collection of statements that perform some specific task and return the result to the caller. A java method can perform some specific task without returning anything. * Java methods allow us to reuse the code without retyping the code.



- .is like a function i-e used to express bhaviour of object.
- .is a set of codes to perform a particular task.

Syntax:

```
<access_modifier> <return_type> <method_name>
                        (list_of_parameters)
{
    //body of code
}
```

Method decleration:

1. MethodsModifier: It defines access type of methodi-e from where it can be accessed in your application.
   - public - accessible in all classes.
   - protected - accessible within class in whithin which it's defined, in subclasses.
   - private - accessible within class in which it's defined.
   - default - It is defined without using any modifier, accessible within same class in which it's defined

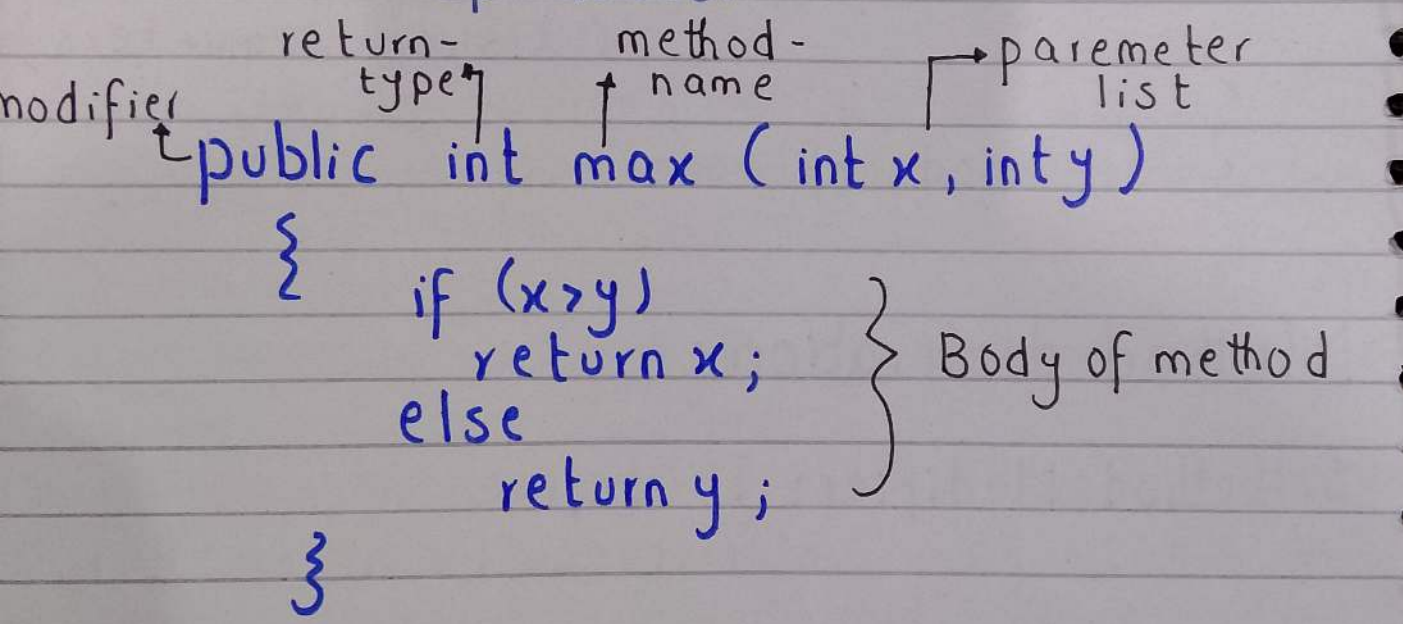
2- The returntype: The datatype of the value returned by the method or void if doesn't return a value.

3- Method name: The rules for field names apply to method names as well.

4- Parameter list: Comma-separated list of the input parameters is defined preceeded by their datatype, within the enclosed parentheses. If no parameters use empty ().

5- Exceptions list: The exceptions you expect by the method can throw, you can specify these exceptions.

6- Method body: It's enclosed between braces. The code you need to be executed to perform intended operations.



Method Calling

Method needs to be called for use its functionality. There can be three situations when method is called. A method returns to code that invoked it when:

- It completes all statements in a method.
- reaches return statment.
- Throws an exception.

14/9/23

String Concatenation:

The '+' operator can be used between strings to combine them. This is called concatenation.

```
String firstName = "John";
String lastName = "Doe";
System.out.println(firstName + " " + lastName);
```

* we've added an empty text (" ") to create a space between firstName and lastName.

You can also use the concat() method to concatenate two strings.

```
String firstName = "John";
String lastName = "Doe";
System.out.println(firstName.concat(lastName));
```

The String concatenation consists of the operands and concatenation is the operation performed on these operands. The '+' operator is used for concatenation but it has polymorphic forms in java, it can be used for concatenation and addition as well. So, whether to do concatenation / addition To decide this '+' uses the operands.

Addition: if the operands are of integer datatype then '+' will undergo addition.

Concatenation: if one operand is string and other is of integer type or if both operands are of string datatype, '+' will undergo concatenation.

Addition

```
int x = 20;
int y = 30;
z = x+y;
```

Output : 50

Concatenation

```
String n = "20";
String m = "30";
String o = n+m;
```

Output : 2030

```
int n1 = 20;
String n2 = "30";
String n3 = n1+n2
```

Output : 2030

1) Java String length() Method:

Def:
The length() method returns the length of a specified string.

Syntax: public int length()

Example:
Return the number of charachters in a string:

```
String txt = "ABCD";
System.out.println (txt.length());
```

Returns: An int value, representing the length of the string.

2) toUpperCase():

The toUpperCase() method converts a string to upper case letters.

Syntax: public String toUpperCase().

3) toLowerCase():

The toLowerCase() method converts a string to lowercase letters.

Syntax: public String toLowerCase().

Example: Convert a string to upper/lowercase.

```
String txt = "Hello, World";
System.out.println(txt.toUpperCase());
System.out.println(txt.toLowerCase());
```

Java String indexOf() method:

Def: The indexOf() method returns the position of first occurence of specified charachter(s) in a string.

Syntax:

- public int indexOf (String str)
- public int indexOf (String str, int start
- public int indexOf (int char)
- public int indexOf (int ch, int start

in case of the unicode

Parameter Values:

str - A string value, representing string to search for
start - An int value, rep. the index to start the search from
ch - An int value, representing a single value/charachter 'A', or a unicode value.

Returns: An int value, representing the index of first occurence of charachter in string

Example: Find first occurence of letter 'e' in String starting search at position 5

```
public class Main {
    public static void main(String[] args){

    String myStr = "Hello planet earth";
    System.out.println(myStr.indexOf("e", 5));
```

Java - max() method:

This method gives the maximum of the two arguments (int, float, long double).

Syntax: `int x = Math.max(39, 49);`

Parameters: This method accepts any primitive datatype as parameter.

Return : This method returns maximum of two arguments.

```
public class Test{
  public static void main(String[] args){
    System.out.println(Math.max(12, 123));
  }
```

Output : 123

Java - sqrt() method:

This method returns the square root of the argument.

Syntax: `int x = Math.sqrt();`

↳ `public static double sqrt(double a)`

Parameter: The value whose square root is to be returned (e.g a).

Return: Positive square root value of argument.

Example:

```
public class Test{
  public static void main(String[] args){
    double x = 81.0;
    System.out.println(Math.sqrt(x));
  }}
```

Output : 9.0

Java - abs() method:

Absolute value refers to the positive value corresponding to the number passed in as in arguments.

Math.abs() - returns the absolute value of a given argument.
If the argument is not negative, the argument is returned.
If the argument is negative, the negation of argument is returned.

Syntax:

public static DataType abs (Datatype a)

Parameters: Int, long, float, double whose absolute value is to be determined.

Returns: Absolute value of the argument. (Positive)

```
public class Main{
public static void main (String[] args){
System.out.println (Math.abs (-4.7));
System.out.println (Math.abs (4.7));
System.out.println (Math.abs (3));
}
```

Output: 4.7
4.7
3

Java - random() method:

This method return a pseudorandom double type number greater than or equal to 0.0 and less than 1.0. The default random number always generated between 0 and 1.

Syntax:

public static double random()

```
int x = int (Math.random()* 100);
```

We give a range of 1-100 to random meth and it will give any number in this range
Output will be integer (converted from float to int).

If you want to specify range of values, you have to multiply the returned value with the magnitude of the range. For example, if you want to get the random number between 0 to 20, the resultant address has to be multiplied by 20 to get result.

```
public class Main{
public static void main(String[]args)
{
    double a = Math.random * 20;
    System.out.print(a);
} // Output is different each time this code
    is executed.
```

Operators In Java

1. Arithmetic Operators - used to perform mathematical operations.

   +, -, *, /, %, ++, --

2. Assignment Operators - used to assign values to variables.

   += , for adding left operand with right operand and then assigning to variable.

   -= , for subtracting left operand with right operand and then assigning it to variable on left.

   *= , for multiplying " " " "

   /= , for dividing " " " "

   %= , for assigning the modulo of the left operand by right operand then assigning it to variable on left.

3. Relational Operators:

These operators are used to check for relations like equality, greater than, less than. They return boolean result after the comparison and extensively used in looping statements, if-else statements.

variable relational_operator value

==, Eequal to	!=, Not equal to
<, less than	<=, less than or equal to
>, greater than	>=, greater than or equal to

4- Logical Operators:

Logical operators are used to determine logic b/w variables/valu

&&, Logical AND - returns true when both conditions are true.

||, Logical OR - returns true if atleast one condition is true.

!, Logical NOT - returns true when a condition is false.

5- Unary Operators:

Unary operator need only one operand.

+ Unary plus	- Unary minus
++ increment operator	-- decrement operato

6- Ternary Operator:

The ternary operator is shorthand version of if-else statement It has three operands.
Also known as conditional operator.

```
condition ? if true : if false
```

The above statement means that if the condition evaluates to true, then execute statements after '?' else execute after ':'

LOOPS IN JAVA

Loops can execute a block of code as long as the specified condition is reached.

Java While Loope: The while loop loops through block of code as long as condition is true.

```
while (condition)
  {
  }
```

Example:

```
int i=0;
while(i<5){
  System.out.println(i);
  i++;
}
```

Java Do/While Loop: The do/while loop is variant of while loop. This loop will execute code block once, before checking if condition is true, then it will repeat the loop as long as condition is true.

Example:

```
int i=0;
do {
 System.out.println(i);
 i++;
}
 while (i<5);
```

Syntax:

```
do {
    //code
}
while(condition);
```

Java For Loop: When you know exactly how many times you want to loop through block of code, use the for loop.

```
for (statement 1 ; statement 2; statment 3)
{
}
```

- statement 1 → executed (one time) before execution of code block.
- statement 2 → defines condition for executing the code block.
- statment 3 → is executed (everytime) after the code block has been executed.

```
for (int i=0 ; i<5 ; i++ ){
    System.out.println(i);
}
```

Nested Loops: It is possible to place a loop inside another loop.
"inner loop" will be executed one time for each iteration of "outerloop".

```
for (int i=1; i<=2; i++){
    System.out.println ("Outer: "+i);

for (int j=1; j<=3; j++){
    System.out.println ("Inner: "+j);
```

Java Break:

The break statement can also be used to jump out of loop.

```
for (int i=0; i<10; i++) {
    if (i==4) {
        break;
    }
    System.out.println(i);
}
```

Output:
0
1
2
3

Java Continue:

The continue statement breaks one iteration (in the loop), if a specified condition occurs, and continues with next iteration in the loop.

```
for (int i=0; i<10; i++) {
    if (i==4) {
        continue;
    }
    System.out.println(i);
}
```

(if i==4)

Output:
0
1
2
3
5
6
7
8

When a continue statement is encountered the control directly jumps to the beginning of the loop for next iteration instead of executing statements of current iteration.

- In case of for loop, the continue keyword force control to jump immidiately to the update statement.
- In case of while / do-while control jumps to Boolean expression.

VARIABLES

=> Used to store values in computer's memory
=> Each variable has a specific type. It's called variable because value can be changed

- **Decleration:** Allocating space inside memory.
  TYPE NAME;
  ↳ a variable must be declared before use.
- **Assignment:** Used to store/put a value inside a variable.
  variableName = expression
- **Initialization:** Assigning a value to a variable when declaring it.
  String myJob = "Programmer";

CONSTANTS

"A variable whose value cannot be changed"
=> To define a constant we use "final" word
Contants names are written in uppercase and using the snake case convention.
You'll get syntax error if you change the value
final TYPENAME = VALUE;

IDENTIFIERS

Identifiers are the names that identify the elements in a program.

- Name of variables
- Name of classes, methods

Rules

- Can contain letters, digits, underscore and dollar sign.
- Must start with letter, underscore or dollar sign.
- Cannot start with a digit.

DATA TYPES

Integers: Numbers without a decimal part.
1, 2, 100, -4, -9, 0

RealNumbers: Numbers with a decimal part.
1.5, 2.5, -4.7, 1.0, 0.0

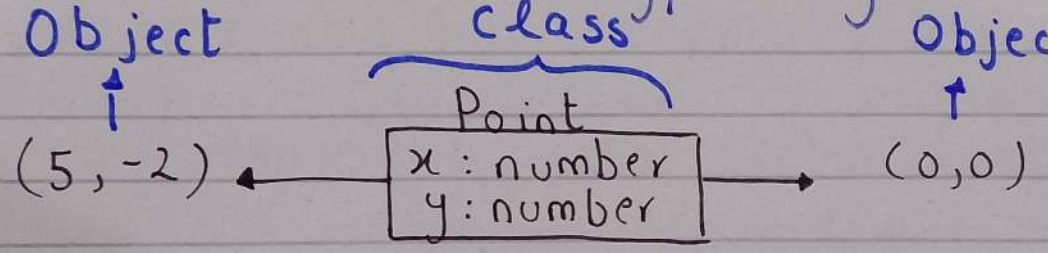
Charachters: All charachters on the keyboard
'a', '5', '-', '*', '?'

Strings: Group of charachters - Text
"abc123,*()", "523", "hello"

Booleans: Represents True & False
true, false

USER-DEFINED TYPES

Create a custom type using Classes, Objects



INT - DATATYPE

An integer is a number that doesnot have a decimal part. 4, 1, 100, -23.

Range of int variable:
Interval of values that can be stored in an int variable.

Declare an int variable.
-> int number;

All numbers in the interval [-2147483648, 2147483647] canbe stored in the variable number.

BYTE

A type used with integers
All numbers in the interval [-128,127] can be stored in byte variable.

SHORT

All numbers in interval [-32768,32767] can be stored in a short variable.

LONG

All numbers in interval [-9223372036547780 L, 9223370368547758O7L] stored in long var.
A letter 'L' should be added to tell the compiler that a number is long not an int.

TYPE CONVERSION

byte -> short -> int -> long

A long can store an int, a short, a byte.
An int can store a short and a byte but it cannot store long.
A short can store a byte.

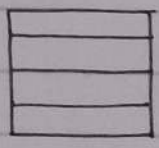
BITS & MEMORY

A bit is the smallest unit to measure memory.
A byte is equal to 8 bits. 1 byte = 8 bits
Our memory is divided into bytes.
Each variable reserves certain number of bytes in memory.

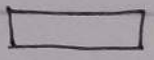
Calculating the Range :

$$[-2^{\text{no. of bits}-1}, (2^{\text{no. of bits}-1})-1]$$

Short = 2 Byte = 16 bits
Range = $[-2^{16-1}, 2^{16-1}-1]$
= [-32,768 , 32,767]



An int variable = 4 byte



A byte variable = 1 byte

String DataType in Java

STRING IS A CLASS:

This class contains some "static methods".

```
public class Main{
  public static void main (String[]args){
    String.
```

Static method { valueOf(int i) → String; valueOf(char c) → String; valueOf(long l) → String } Return Type

First of all, i'm using the name String, written with capital 'S' so this means that String is a class. Then i'm using dot operator and then there's list of methods. All of these are static methods because we're calling them using the "name of class".

STRING OBJECTS:

Variables (Objects) of type String, also have some methods

```
public static void main (String[]args){
  ① String name;
  ② name.
```

Not static method { concat(String str) → String

① We created a variable of type String so name is an "object" of String class.

② We're using name and dot operator and a list of methods. These methods are not static methods

Calling Some Methods

```
public static void main (String [] args){
    String text = " This is some text ";
    text . toUpperCase ()
```

-> Creates a new String "THIS IS SOME TEXT".
-> The method will return the new String
-> The new String canbe stored inside var.

Note: A new String is created and original String is not modified.

```
System.out.println (upperCaseText );
```
THIS IS SOME TEXT

```
System.out.println (text);
```
This is some text

CONCATENATION

Adding strings to eachother

```
public static void main (String [] args) {
    String s1 = "Neso ";
    String s2 = "Academy";
    System.out.println ( s1 + s2);
```
Output : Neso Academy

Concatenating Strings and Numbers

```
System.out.println ("My favourite number is "
        + 5 );
```
↓ Automatically convert 5 to "5".

```
System.out.println ("My favourite number is " +
        "5 ");
```
Output: My favourite number is 5.

INSTANTIATING AN OBJECT

Creating an object of a class.

```
ClassName objectName = new ClassName (
                              Parameters );
```

Parameters → a special method called constructor.

Creating a variable (object) of type String (class)

```
public static void main (String [] args){
    String s1 = new String ("Neso Academy");
    System.out.println (s1);
}
```

CONSTRUCTORS IN JAVA

Constructors are "block of codes smilar to method". It's called when an instance of class is created.

Special Type of Method used to initialize object.

Everytime, an object is created using new() keyword, at least one constructor is called.

```
import java.io.*;

class Geeks{
    Geeks ()  // constructor
    {   super ();
        System.out.println ("Constructor Call");
    }
    public static void main (String[] args)
    {
        Greeks geek = new Geeks();
```

19/9/23

For-each loop in Java:

- Powerful loop in which the developer doesn't know the start and end of loop.
It's commonly used to iterate over an array or Class Collections
Instead of declaring and initializing a loop counter variable, you declare a variable that's same type as base type of array name

```
for ( type variableName : arrayName)
  {
  //statements using variableName
  }
```

In the loop body, you can use the loop variable you created, rather than using an indexed array element.

```
String [] x = {"ali", "nasir", "kashif"};
  for (String p : x )
      {
        System.out.println (p);
      }
```

Here; x is an array, p is a variable and is of same datatype as that of array.
Each item of x is assigned to p.
(array) (variable)
p will run the loop from first to last member iterate from first to last member.

Note: → you can't print any random number
→ you can't print reverse of loop.