JAVA

LEARNING

Ayushi Dixit | NOTES | 25-09-2024

Primitives :

- The most basic fundamental data types in JAVA.
- The primitive types represent single values—not complex objects.
- Java defines eight primitive types of data**: byte, short, int, long, char, float, double, and boolean.**
- sin( ), cos( ), and sqrt( ), return double values.

```
public class primitives {
    public static void main(String[] args) {
        int marks = 98;
        char section = 'A';
        byte age = 100; // 8 bit
        short roll_no = 7484; //16 bit type
        long phone_no = 987654321; //64 bit type
        float weight = 10.56f; //real numbers 32 bits
        double size_of_celestrial_body = 345678228754288664421436772 2.645; //64
bits
        boolean check = true;

    }
}
```

In some cases float throws error (Floating point error) when we use decimal in it so, we use 'f' for float to represent decimal values in Float.

IMP:

- String is not in Primitives
- In JAVA strings are written in (" ") while char in (' ')

INPUTS

```
Scanner input = new Scanner(System.in);
int roll_no = input.nextInt();
System.out.println("Your roll no is" + roll_no);
```

```
  Inputs
23
Your roll no is23
```

IMP:

- print() :- the next output will be printed on the same line, immediately.
- println() :- adds a new line character (\n) at the end. The next output will be printed on a new line.

Floating point error:-

```
float marks = input.nextFloat();
System.out.println("my marks are" + marks);
```

```
  Inputs
73633.38366333
my marks are73633.38
```

The digits got round off, this is the floating point error.

Type conversion & casting

Assigning a value of one type to a variable of another type.

Java performs the conversion automatically if the types are compatible

e.g. int value can be assigned to a float var BUT double can't be assigned to byte.

An automatic conversion;

- Types are compatible
- Destination type is larger than the source type
- No automatic conversion of num to char or Boolean.

```
int num = 'A';
System.out.println(num);
```

```
65
```

here, we r giving a string in integer and it results in an integer
65 is the ASCII value is A.
Java follows the Unicode rules, hence any language can be used.

TRUNCATION
When floating point value is assigned to an integer type, the fractional component is lost.

```
float a = 43.5f;
int s = (int) (a);
System.out.println(s);
```

```
43
```

Casting 257 to a byte variable.
As the range of byte is 256, if we assign a larger value it results as [257 % 256 == 1]

```
int a = 259;
byte b = (byte) (a);
System.out.println(b);
```

```
3
```

```
byte x = 60;
int y = x;
System.out.println(y);
```

```
60
```

here a byte x is assigned to an int variable, hence destination type is larger than a source type. But the reverse is not true.

Here comes the role of CASTING
Assigning an int to a byte variable is narrowing conversion.

```
int a = 23;
int b = 34;

byte c = (byte) ((byte) (a) + (byte) (b));
System.out.println(c);
```

```
57
```

Automatic Type Promotion in Expressions

Java automatically promotes each byte, short, or char operand to int when evaluating an expression.

```
byte a = 40;
byte b = 30;
byte c = 100;
int d = a *b /c ;
System.out.println(d);
```

```
12
```

Here a * b easily exceeds the range of a byte operand i.e. 256, here java automatically handled the expression and a * b is performed using integers.
This can also sometimes cause a problem e.g.

```
byte b = 20;
b = b*2;
System.out.println(b);
```

```
java: incompatible types: possible lossy conversion from int to byte
```

the operands were automatically promoted to int, hence the result of the expression is now of type int, which cannot be assigned to a byte without the use of a cast.
Casting:

```
byte b = 20;
b = (byte) (b*2);
System.out.println(b);
```

```
40
```

```
import java.io.File;
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {
        System.out.println("Hello and welcome!");
        Scanner input = new Scanner(System.in);
    }
}
```

Output:

```
1. Hello and welcome!
```

SO, basically

- Scanner here is the CLASS in java.util package that allows us to take the input.
- System.in takes the input from the keyboard. We can change it to the type you want e.g. File
- Input = Variable
- The public means the class can be accessed from anywhere in the program.
- void means the method doesn't return anything.

```
import java.io.File;
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.println(input.nextInt());
    }
}
```

Output:

```
1. Input 456
2. 456
```

Here, System.out.println(input.nextInt()); reads an integer value from the user input and then prints the value to the console

To input String

```
1. System.out.println(input.next());
```

```
you are learning JAVA
you
```

Scanner uses white space as a delimiter so this will print only one word from the string, To get the whole line printed use

```
1. System.out.println(input.nextLine());
```

```
My name is Akemi
My name is Akemi
```

A Simple JAVA program:

```
import java.util.Scanner;

public class temperature {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.println("Please enter the temperature in C:");

        float C = in.nextFloat();
        float F = (C * 9/5) + 32;
        System.out.println(F);
    }
}
```

```
Please enter the temperature in C:
 40
104.0
```

Conditionals & Loops

Control Statements: Define the flow of execution to advance and branch based on changes to the state of a program.

- Selection: To choose different paths of execution based upon the outcome
- Iteration: Enables program execution to repeat one or more statements
- Jump: To execute program in a non-linear fashion.

IF

```
Syntax of IF statement:
if (boolean condition){
  //body
} else {
 // do this
}
```

e.g.

```java
int salary = 20500;
if (salary > 20000) {
    salary = salary + 2000;

} else{
    salary = salary + 1000;
}
System.out.println(salary);
```

```
22500
```

FOR

```java
//FOR-LOOP
// print no from 1 to 5
 Syntax
 for(initialization, condition, increment/decrement){
 //body
 }
```

e.g.

```java
Scanner in = new Scanner(System.in);
int n = in.nextInt();

for (int num = 1; num <= n; num++) {
    System.out.print(num + " ");
```

```
50
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
```

WHILE

```java
WHILE-LOOP

        while (condition){
        body
        }
```

e.g.

```java
int x = 1;
while (x<=5){
    System.out.print(x + ' ');
    x ++;
}
```

```
1 2 3 4 5
```

DO-WHILE-Loop

```java
DO-WHILE-LOOP
        Syntax

        do{

        } while (condition);
```

Ques: Find the largest of three numbers

Method 1:

```
import java.util.Scanner;

public class Largest_no {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int a = in.nextInt();
        int b = in.nextInt();
        int c = in.nextInt();

        int max = 0;
        if (a > max) {
            max = a;
        }

        if (b> max){
            max = b;
        }

        if (c> max){
            max=c;
        }

        System.out.println(max);

    }
```

Method 2:

```
System.out.println(Math.max(98,(Math.max(23,45))));
```

Method 3:

```
import java.util.Scanner;

public class Largest_no {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int a = in.nextInt();
        int b = in.nextInt();
        int c = in.nextInt();
    System.out.println(Math.max(c,(Math.max(a,b))));
    }

}
```

```
34 78 90
90
```

Ques:2 Fibonacci Sequence

```
import java.util.Scanner;

public class Fibonacci {
    public static void main(String[] args) {
        Scanner in= new Scanner(System.in);
        int n = in.nextInt();
        int a = 0;
        int b = 1;
        int count = 2; // cause n-th fibonaccci is the sum of 2 previous digits
```

```
        while (count <= n) {
            int temp = b;

            b = a + b;
            a = temp;
            count++;
        }
        System.out.println(b);
    }
}
```

```
7
13
```

The 7-th number in the Fibonacci sequence is 13

Ques :3 Occurrence of a number

```
import java.util.Scanner;

public class Occurence {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int n = in.nextInt();
        int num = in.nextInt();
        int count = 0;

        while (n>0) {
            int rem = n % 10;
            if (rem == num ) count++;
        n = n / 10;
        }
        System.out.println(count);
    }
}
```

```
7676767
6
3
```

6 occurred 3 times in the given number.

Ques:4 Reverse the number given

```
import java.util.Scanner;

public class Reverse {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
```

```
        int num = in.nextInt();
        int reverse = 0;

        while (num >0){
            int rem = num % 10;
            num = num / 10;
            reverse = (reverse * 10) + rem;

        }
        System.out.println(reverse);
    }
}
```

```
987654
456789
```

SWITCH STATEMENTS

Switch statements provide an easy way to dispatch execution to different parts of the code based on the value of an expression.

It often provides a better alternative to a large If-else-If statements.

Basic syntax:

```
switch(expression){
   case value1:
       //statement sequence
       break;
   case value1:
       //statement sequence
       break;
    .
    .
    .
   case valueN:
       //statement sequence
       break;
   default:
      //default statement sequence
}
```

Each value specified in the case statements must be a unique

Values of expression statement are compared with the case statements,

- If match found => statement sequence of that case get execute
- If none of the cases match => Default statement is executed.

BREAK has the effect of "jumping out" and further execution begins from first line. If break is not used, it will continue to the next case.

```
public class Switch {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        String fruit = in.next();

        switch (fruit){
            case "Mango":
                System.out.println("King of fruits");
                break;
            case "Apple":
```

```
                System.out.println("sweet fruit");
                break;
            case ("Banana"):
                System.out.println("yellow fruit");
                break;
            default:
                System.out.println("Provide a valid input");
                break;
        }
    }
```

```
Banana
yellow fruit
```

New syntax:

```
Scanner in = new Scanner(System.in);
String fruit = in.next();

switch (fruit) {
    case "Mango" -> System.out.println("King of fruits");
    case "Apple" -> System.out.println("sweet fruit");
    case "Banana" -> System.out.println("yellow fruit");
    default -> System.out.println("Provide a valid input");
}
```

Ques : Print WEEKDAY for each day from Monday to Friday and WEEKEND for Saturday and Sunday.

```
switch (day) {
    case 1, 2, 3, 4, 5 -> System.out.println("WEEKDAY");
    case 6, 7 -> System.out.println("WEEKEND");
}
```

```
7
WEEKEND
```

METHODS

- **Methods** are blocks of code that perform specific tasks, and they are defined within a class.
- Used to execute logic, **reuse code**, and organize the code/functionality.
- They have a property to **encapsulate behavior.**

General syntax:

```
1. type name(parameter-list){
2.     //body of method
3. }
```

- If the method does not return a value, its return type must be void.
- Method name is specified by name.
- list is the list of arguments to be passed to the method.

```
public class greetings {
    public static void main(String[] args) {
        greetings();
    }
    static void greetings(){         //we use `void` when we dont want to return anything, but only print!.
        System.out.println("Hello methods");
    }
}
```

- method:- greetings()
- no return type, just the method is called and it prints the text

```java
import java.util.Scanner;

public class Methods {
    public static void main(String[] args) {
         sum();

    }
    static void sum(){    /here we'll use the datatype that will be returned, i.e void as nothing is returned
        Scanner in = new Scanner(System.in);
        int a = in.nextInt();
        int b = in.nextInt();
        int sum = a+b;
        System.out.println("sum = " + sum );
    }
}
```

- method:- sum()
- no return statement

with return statement

```java
import java.util.Scanner;
public class Methods {
    public static void main(String[] args) {
        int ans =sum();
        System.out.println(ans);
    }
    static int sum(){       /int will be returned
        Scanner in = new Scanner(System.in);
        int a = in.nextInt();
        int b = in.nextInt();
        int sum1 = a+b;
        return sum1;
    }
}
```

```
Methods
43
44
87
```

- Here, the SUM of **a** and **b** is returned as `**sum1**` which is stored as **sum()**, stored in variable **ans**, then printed as **ans.**
- When the return statement gets executed, the function will end there and nothing will be executed further.

```java
String methods
import java.util.Scanner;

public class Methods {
    public static void main(String[] args) {
         String message = greet();                    //String type
         System.out.println(message);

    }
    static String greet (){
        String greeting = "hey! there!" ;
    return greeting;
```

To accept inputs in metho ds

```java
public static void main(String[] args) {
    Scanner in = new Scanner(System.in);
    String write = in.nextLine();
    String input = wave(write);
    System.out.print(input);
}

static String wave(String write) {
     String text = "JAVA " + write;
     return text;
```

```
Methods
done with time
JAVA done with time
```

Pass value and Swapping

In Java, **all arguments are passed by value**.

```
public class swapping {
    public static void main(String[] args) {
        String name = "Akemi";
        hii(name);
        System.out.println(name);
    }
    static void hii(String nam) {                          //creating a new object nam
    nam = "lily";                                          // in method hii, the name will not change
    }
}
```

```
swapping
Akemi
```

This means that when you pass the variable nam to the method hii(), a **copy** of the reference to the String object is passed, not the reference itself.

Hence, changes to the nam variable inside the hii() method do not affect the name variable in the main() method.

Case: When the name actually changes

```
public class swapping {
    public static void main(String[] args) {
        String name = "Akemi";
        name = hii(name);                      // Reassign 'name' to the value returned by 'hii', this is the diff
        System.out.println(name );            // Now it will print "Meethi"
    }
    static String hii( String name) {
        name = "lily";                      // Change the local 'name' variable
        return name;                           // Return the new value
    }
}
```

```
swapping
lily
```

In arrays and objects the value is passed by reference of that variable.

Variable Length Arguments

. When there is no clarity on how many inputs the program is going to have. It is written as

```
(<Datatype of input>…v)

e.g.
static void fun(int . . .V){                          /this create an array
static void fun(String . . .V){
static void fun(char. . .V){
```

```
public class VarArgs {
    public static void main(String[] args) {
        fun(2,4,54,67,33,4,2,1);                      //input, any number of arguments
    }
    static void fun(int ...v){                        // v is declared as an array of type int
        System.out.println(Arrays.toString(v));          //store it in an array
    }
}
```

```
VarArgs
[2, 4, 54, 67, 33, 4, 2, 1]
```

ARRAYS

Collection of a datatype

All the datatype in an array must be same.

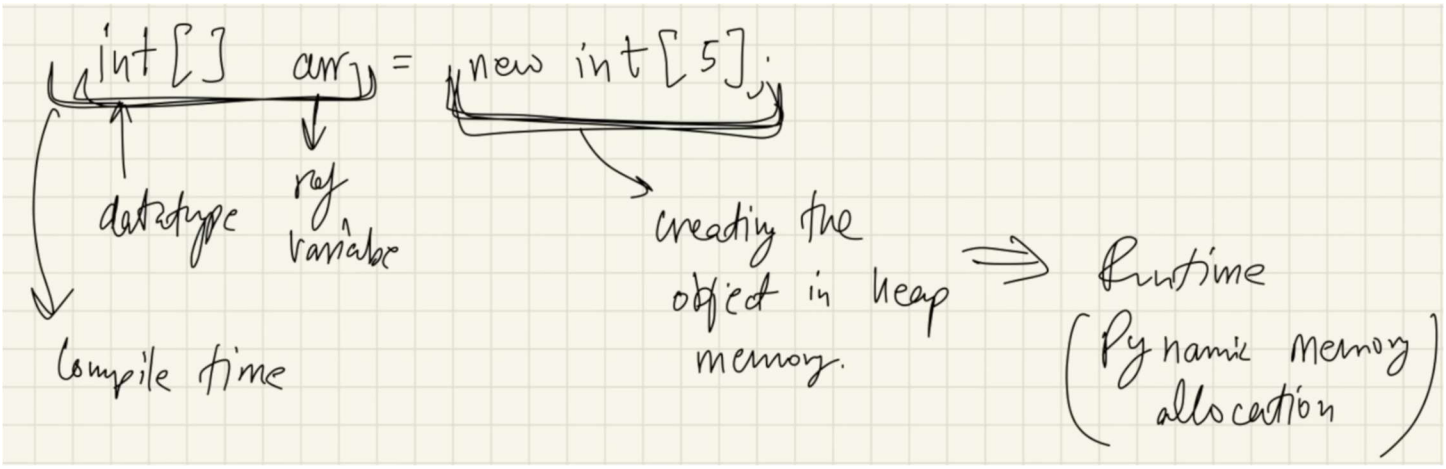
A group of like-typed variables, You need an array variable to create an array, defined as type var-name[ ];

e.g. int month_days[]; ➔ declares an array named month_days and type is array of an integer.

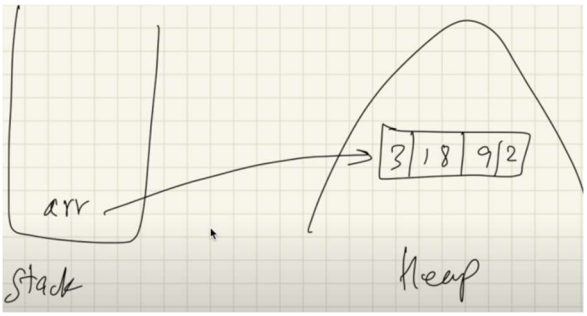
```
1.  Int[] rollno1 = {1,2,3,4,5,6};
```

```
Int[] arr;          //declaration, rollno1 is getting defined in the stack
arr = new int[5];   //initialisation, here obj is being created in the heap memory.
```

new is a special operator that allocates memory.



New int[5] ➔ new is used to create an int object in heap memory of array size 5



- array objects are in Heap
- Array is stored in Stack and is pointing towards obj in heap.
- Heap objects are not continuous. i.e. the memory allocation does not happen in a row, it is randomly done.

1	2		3	5

- Array objects are at heap area, hence they may not be conti. It depends on JVM
- Dynamic memory allocation happens at runtime

INDEX of an Array

0	1	2	3	4	5	6
23	44	56	78	12	70	46

```
print(arr[0]) ➔  23
print(arr[3]) ➔  78
```

By default, without passing an array, For int:

```
int[] arr = new int[5];
System.out.println(arr[0]);
```

```
Arrays.ok
0
```

For string:

```
int[] String = new String [5];                    //no input provided
System.out.println(arr[0]);
```

```
  Arrays.ok
 null
```

If you try to access elements outside the range of the array (negative numbers or numbers greater than the length of the array), you will cause a run-time error.

```
Scanner in = new Scanner(System.in);

int[] arr = new int[5];                                  //declaration and initialisation
 for (int i = 0 ; i < arr.length ; i++){                 // loop that iterates over the length
     arr[i] = in.nextInt();
     System.out.print(arr[i] + " ");
 }
```

```
3 4 6 8 1
3 4 6 8 1
```

Taking integer as a input and printing array: Arrays.toString

```
Scanner in = new Scanner(System.in);
        int[] arr = new int[4];                                  //defining the length of an array

         for (int i = 0 ; i < arr.length ; i++) {                //condition using arr.length
             arr[i] = in.nextInt();
         }
         System.out.println(Arrays.toString(arr));
  }
}
```

```
7 7 7 77
[7, 7, 7, 77]
```

Passing functions in arrays:

```
package Arrays;
import java.util.Arrays;

public class passingfunctions {
    public static void main(String[] args) {
        int[] nums = {3,3,3,3};
        System.out.println(Arrays.toString(nums));
        change(nums);
        System.out.println(Arrays.toString(nums));

    }
    static void change(int[] arr){
        arr[0] = 100;
    }
}
```

Question

```
1. class Employee
2. {
3. int eid;
4. String ename;
5. public void display()
6. {
7. System.out.println("eid: "+eid);
```

```
 8. System.out.println("ename: "+ename);
 9. }
10. }
11. class FClass
12. {
13. public static void main(String[] args)
14. {
15. Employee e1 = new Employee();
16. e1.display();
17. }
18. }
19.
```

For the given code, Program runs successfully, and the data members eid and ename contains 0 and null respectively.

Reason: If no constructor is defined in a class the default constructor would be executed when an object is created. In JAVA uninitialized variables have values 0 for numeric types and null for string types.

Question: Taking input as an array of 6 digits and printing average

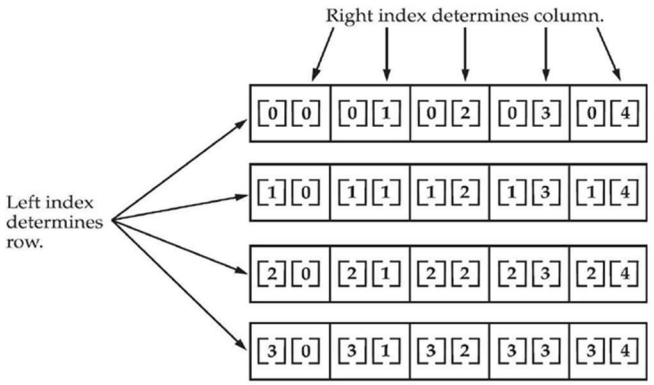
```
package Arrays;
import java.util.Scanner;
public class multi_dim_arrays {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int[] input = new int[6];                    //taking input as an array
        int sum = 0;
        for ( int i =0; i < 6 ; i++ ) {
            input[i] = in.nextInt();
            sum = sum + input[i];

        }
        System.out.println("Average is : " + sum / 6.0);
    }
}
```

2D Arrays

Two D arrays are declared as:

```
int two_d[][] = new int [4][5];
```



```java
package Arrays;

import java.sql.SQLOutput;
import java.util.Arrays;
import java.util.Scanner;

public class multi_dim {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int[][] arr = new int[3][3];

        System.out.println(arr.length); //no of rows

//input

        for (int row = 0; row < arr.length; row++) {
            //for each col in every row
            for (int col = 0; col < arr[row].length; col++) {
                arr[row][col] = in.nextInt();
            }
        }

//output

        for (int[] ints : arr) {
            //for each col in every row
            for (int col = 0; col < ints.length; col++) {
                System.out.print(ints[col] + " ");
            }
            System.out.println();

//modified form of output

        for (int[] a : arr) {
            System.out.println(Arrays.toString(a));
        }


    }
    }
```

ARRAYS R STILL LEFT!!!!!!!!

OBJECT ORIENTED PROGRAMMING

Object-oriented programming (OOP) is at the core of Java. programs started to grow larger and more complex. To manage increasing complexity, the second approach, called object-oriented programming, was conceived.

- It organizes a program around its data (that is, objects)
- It can be characterized as *data controlling access to code.*
- the class forms the basis for object-oriented programming in Java.

CLASS

The logical construct upon which the entire Java language is built because it defines the shape and nature of an object.

A class defines a new data type.