



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_celestial_body = 3456782287542886644214367722.645; //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.

```
int salary = 20500;
if (salary > 20000) {
    salary = salary + 2000;
} else{
    salary = salary + 1000;
}
System.out.println(salary);
```

```
22500
```

FOR

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

e.g.

```
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

```
WHILE-LOOP

while (condition){
    body
}
```

e.g.

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

```
1 2 3 4 5
```

DO-WHILE-Loop

```
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 fibonacci 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:

```
1. switch(expression){
2.     case value1:
3.         //statement sequence
4.         break;
5.     case value1:
6.         //statement sequence
7.         break;
8.     .
9.     .
10.    .
11.    case valueN:
12.        //statement sequence
13.        break;
14.    default:
15.        //default statement sequence
16. }
```

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

```