DAY-8

20 June 2023 16:20

FLOAT DATA TYPE

- YOU SHOULD USED A FLOATING POINT TYPE WHENVER YOU NEED A NUMBER WITH A DECIMAL, SUCH 9.99 OR 3.1415
- ITS SIZE IS 4 BYTE
- IT RANGES FROM -3.4e^{38} to 3.4e^{38}
- NOTE THAT YOU SHOULD END THE VALUE WITH AN "f";
- EXAMPLE :

```
Float f1 = 234.244f;
```

DOUBLE DATA TYPE

- THE DOUBLE DATA TYPE IS A DOUBLE PRECISION 64-BIT FLOATING POINT 8 BYTES
- THE DOUBLE DATA TYPE IS GENERALLY USED FOR DECIMAL VALUES LIKE FLOAT
- ITS RANGE FROM -1.7e^{38} to 1.7e^{38}

EXAMPLE :

```
Double d1 = 12.3;
```

CHAR DATA TYPE

- THE CHAR DATA TYPE IS A SINGLE 16-BIT UNICODE CHARACTER
- THE CHAR DATA TYPE IS USED TO STORE CHARACTERS

EXAMPLE :

```
Char a = 'A';
```

BOOLEAN DATA TYPE

- THE BOOLEAN DATA TYOE IS USED TO STORE ONLY TWO POSSIBLE VALUES : true AND false
- THE BOOLEAN DATA TYPE SPECIFIES ONE BIT OF INFORMATION , BUT ITS SIZE CAN'T BE DEFINED PRECISELY

```
Boolean one = false;
```

VARIABLE DECALRATION AND INTIALIZATION

1st way

```
Int number; //declaration

Number = 100; //initialization
```

2nd way

```
Int number = 10; //DECLARATION AND INTIALIZATION IN SINGLE STATEMENT

 float fraction;
 fraction = 4.52f;

Float fraction = 4.52f;
```

VARIABLES

- VARIABLES CAN BE REINITIALIZED ANY NUMBER OF TIMES
- VARIABLE CAN NOT BE REDECLARED AGAIN WITHIN A SAME METHOD
- A LOCAL VARIABLE CAN NOT BE ACCESSED WITHOUT INTIALIZING IT (COMPILATION ERROR)

EXAMPLE :

```
class Test1
{


	public static void main(String[] args)
	{
		System.out.println(x);


	}

}
```

CTE

Example 2"

```
class Test1
{


	public static void main(String[] args)
	{
		int x;
		System.out.println(x);
```

```
        }

}
```

OPTIONS

1. 0
2. JUNK
3. 1
4. ERROR

ANS : CTE - variable x might not have been initialized

NOTE :

LOCAL VARIABLES DECLARED ARE VISIBLE ONLY INSIDE THE SCOPE OF THE METHOD

HOW TO DECLARE MULTPLE VARIABLES

```
Int x;
Int y;        OR    int x, y, z
Int z;
                        ✓
```

```
Int x =10;
Int y = 20;     or int x=10, y = 20, z=30;
Int z = 30;
```

```
Int x, int y, int z          ✗
```

```
Int x =10, int y = 20, int z = 30;
```

```
class Test1
{


        public static void main(String[] args)
        {

                System.out.println("Hello World");
                int x = 10;
                int y;

                int z = x + y;

                System.out.println(z);  // variable y might not have been initialized


        }
}
```

FINAL KEYWORD

- IF WE DECLARE A VARIABLE AS FINAL, WE CAN NOT REINITALISE IT AGAIN
- IF WE WANT TO DECALRE A VARIABLE WHOSE VALUE SHOULD NOT BE CHANGED /CONSTANT VARIABLE WE CAN DECLARE IT AS A FINAL VARIABLE

```
class Test1
{


        public static void main(String[] args)
        {

                final int a = 20;
                System.out.println("The value of a is "+a);

                a = 10;
                System.out.println("The value of a is "+a); //cannot assign a value to final variable a



        }

}
```

Example 2 :

```
class Test1
{


        public static void main(String[] args)
        {

                final String s = "java";

                s = "sql";


        }

}
```

OPERATORS

WHAT IS AN OPERATOR ?

- OPERATORS ARE SOME SYMBOLS USED TO PERFROM SOME OPERATIONS ON THE OPERANDS
- THERE ARE USED TO BUILD EXPRESSION

TYPES OF OPERATORS IN JAVA

- ARITHMETIC OPERATOR
- RELATIONAL OPERATOR
- LOGICAL OPERATOR
- BITWISE OPERATOR
- SHIFT OPERATOR
- TERNARY OPERATOR
- UNARY OPERATOR

JAVA ARITHMETIC OPERATOR

- JAVA ARITHEMETIC OPERATORS ARE USED PERFORM ADDITION, SUBSTRACTION, MULTIPLICATION AND DIVISION
- THEY ARE MATHEMATICAL OPERATORS
- THEY ARE
    - **" + " IS USED FOR ADDITON**
    - **" - " IS USED FOR SUBSTRACTION**
    - **" * " IS USED FOR MULTIPLICATION**
    - **" / " IS USED FOR DIVISION**
    - **% IS USED FOR MODULUS**

JAVA RELATIONAL OPERATIONAL OPERATOR

- RELATIONAL OPERATOR ALWAYS GIVE BOOLEAN RESULTS
- THEY COMPARE TWO VALUES
- THEY ARE
    - " > " GREATER THAN SYMBOL
    - " < " LESSER THAN SYMBOL
    - " >=" GREATER THAN OR EQUALS
    - " < = " LESSER THAN OR EQUALS
    - = ASSIGNMENT SYMBOL
    - == EQUALITY SYMBOL
    - != NOT EQUAL SYMBOL

GREATER THAN (>)

- CHECKS IF THE VALUE OF LEFT OPERAND IS GREATER THAN THE VALUE OF RIGHT OPERAND, IF YES THEN CONDITION BECOMES TRUE

```
class Test1
```

```
{


    public static void main(String[] args)
    {

        int a =10, b = 20;
        System.out.println("a is greater than b ? " + (a>b));


    }

}
```

LESSER THAN (>)

- CHECKS IF THE VALUE OF LEFT OPERAND IS LESS THAN THE VALUE OF RIGHT OPERAND, IF YES THEN CONDITION BECOMES TRUE

```
class Test1
{


    public static void main(String[] args)
    {

        int a =10, b = 20;
        System.out.println("a is lesser than b ? " + (a<b));


    }

}
```

GREATER THAN OR EQUALS (>=)

- CHECKS IF THE VALUE OF LEFT OPERAND IS GREATER THAN OR EQUALS THE VALUE OF RIGHT OPERAND, IF YES THEN CONDITION BECOMES TRUE

```
class Test1
{


    public static void main(String[] args)
    {

        int a =10, b = 20;
        System.out.println("a is greater or equals than b ? " + (a>=b));


    }
}
```

LESSER THAN OR EQUALS (>=)

- CHECKS IF THE VALUE OF LEFT OPERAND IS LESS THAN OR EQUALS THE VALUE OF RIGHT OPERAND, IF YES THEN CONDITION BECOMES TRUE

```
class Test1
{


        public static void main(String[] args)
        {

                int a =10, b = 20;
                System.out.println("a is less than or equals than b ? " + (a<=b));


        }

}
```

== (EQUAL TO)

- CHECKS IF THE VALUES OF TWO OPERANDS ARE EQUAL OR NOT , IF YES THEN CONDITION BECOMES TRUE

```
class Test1
{


        public static void main(String[] args)
        {

                int a =10, b = 20;
                System.out.println("a is equals to b ? " + (a==b));


        }

}
```

!= (EQUAL TO)

- CHECKS IF THE VALUES OF TWO OPERANDS ARE EQUAL OR NOT , IF YESTHE VALUES ARE NOT EQUAL THEN CONDITION BECOMES TRUE

```
class Test1
{
      public static void main(String[] args)
      {

          int a =10, b = 20;
            System.out.println("a is not equals to b ? " + (a!=b));


      }

}
```