2. An Overview of Java

Java Comments

The Java comments are the statements that are not executed by the compiler and interpreter. The comments can be used to provide information or explanation about the variable, method, class or any statement. It can also be used to hide program code.

Types of Java Comments

There are three types of comments in Java.

1. Single Line Comment

The single line comment is used to comment only one line.

Syntax:

//This is single line comment

2. Multi Line Comment

The multi line comment is used to comment multiple lines of code.

Syntax:

/*

This is
is
multi line
comment
*/

3. Documentation Comment

The documentation comment is used to create documentation API. To create documentation API, you need to use javadoc_tool.
Syntax:

```java
/**
 * This is documentation comment
 */

Example:

```java
/** The Calculator class provides methods to get addition and subtraction of given 2 numbers. */

public class Calculator {
    /** The add() method returns addition of given numbers. */
    public static int add(int a, int b) { return a + b; }

    /** The sub() method returns subtraction of given numbers. */
    public static int sub(int a, int b) { return a - b; }
}
```

2.2. DataTypes:

Data types specify the different sizes and values that can be stored in the variable. There are two types of data types in Java:

1. Primitive data types: The primitive data types include boolean, char, byte, short, int, long, float and double.
2. Non-primitive data types: The non-primitive data types include Classes, Interfaces, and Arrays.

Java Primitive Data Types

In Java language, primitive data types are the building blocks of data manipulation. These are the most basic data types available in Java.
Boolean Data Type

The Boolean data type is used to store only two possible values: true and false. This data type is used for simple flags that track true/false conditions.

The Boolean data type specifies one bit of information, but its "size" can't be defined precisely.

Example: Boolean one = false

Byte Data Type

The byte data type is an example of primitive data type. It is an 8-bit signed two's complement integer. Its value range lies between -128 to 127 (inclusive). Its minimum value is -128 and maximum value is 127. Its default value is 0.

The byte data type is used to save memory in large arrays where the memory savings is most required. It saves space because a byte is 4 times smaller than an integer. It can also be used in place of "int" data type.

Example: byte a = 10, byte b = -20

Short Data Type
The short data type is a 16-bit signed two's complement integer. Its value-range lies between -32,768 to 32,767 (inclusive). Its minimum value is -32,768 and maximum value is 32,767. Its default value is 0.

The short data type can also be used to save memory just like byte data type. A short data type is 2 times smaller than an integer.

Example: short s = 10000, short r = -5000

**Int Data Type**

The int data type is a 32-bit signed two's complement integer. Its value-range lies between -2,147,483,648 (-2^31) to 2,147,483,647 (2^31 -1) (inclusive). Its minimum value is -2,147,483,648 and maximum value is 2,147,483,647. Its default value is 0.

The int data type is generally used as a default data type for integral values unless if there is no problem about memory.

Example: int a = 100000, int b = -200000

**Long Data Type**

The long data type is a 64-bit two’s complement integer. Its value-range lies between -9,223,372,036,854,775,808(-2^63) to 9,223,372,036,854,775,807(2^63 -1)(inclusive). Its minimum value is -9,223,372,036,854,775,808 and maximum value is 9,223,372,036,854,775,807. Its default value is 0. The long data type is used when you need a range of values more than those provided by int.

Example: long a = 100000L, long b = -200000L

**Float Data Type**

The float data type is a single-precision 32-bit IEEE 754 floating point. Its value range is unlimited. It is recommended to use a float (instead of double) if you need to save memory in large arrays of floating point numbers. The float data type should never be used for precise values, such as currency. Its default value is 0.0F.

Example: float f1 = 234.5f

**Double Data Type**

The double data type is a double-precision 64-bit IEEE 754 floating point. Its value range is unlimited. The double data type is generally used for decimal values just like float. The double
data type also should never be used for precise values, such as currency. Its default value is 0.0d.

Example: double d1 = 12.3

Char Data Type

The char data type is a single 16-bit Unicode character. Its value-range lies between \u0000 (or 0) to \uffff (or 65,535 inclusive). The char data type is used to store characters.

Example: char letterA = 'A'

Operators in Java

Operator in Java is a symbol which is used to perform operations. For example: +, -, *, / etc.

There are many types of operators in Java which are given below:

- Unary Operator,
- Arithmetic Operator,
- Shift Operator,
- Relational Operator,
- Bitwise Operator,
- Logical Operator,
- Ternary Operator and
- Assignment Operator.

Java Unary Operator

The Java unary operators require only one operand. Unary operators are used to perform various operations i.e.:

- incrementing/decrementing a value by one
- negating an expression
- inverting the value of a boolean

Java Unary Operator Example: ++ and --

class OperatorExample{
    
    public static void main(String args[]){
    
    
}
class OperatorExample {
    public static void main(String args[]){
        int x = 10;
        System.out.println(x++); // 10 (11)
        System.out.println(++x); // 12
        System.out.println(x--); // 12 (11)
        System.out.println(--x); // 10
    }
}

Output:
10
12
12
10

Java Unary Operator Example 2: ++ and --

class OperatorExample {
    public static void main(String args[]){
        int a = 10;
        int b = 10;
        System.out.println(a++ + ++a); // 10 + 12 = 22
        System.out.println(b++ + b++); // 10 + 11 = 21
    }
}

Output:
22
21
Java Unary Operator Example: ~ and !

class OperatorExample{
    public static void main(String args[]){
        int a=10;
        int b=-10;
        boolean c=true;
        boolean d=false;
        System.out.println(~a);//-11 (minus of total positive value which starts from 0)
        System.out.println(~b);//9 (positive of total minus, positive starts from 0)
        System.out.println(!c);//false (opposite of boolean value)
        System.out.println(!d);//true
    }
}

Output:
-11
9
false
true

Java Arithmetic Operators

Java arithmetic operators are used to perform addition, subtraction, multiplication, and division. They act as basic mathematical operations.

Java Arithmetic Operator Example

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

```java
int a=10;
int b=5;

System.out.println(a+b);//15
System.out.println(a-b);//5
System.out.println(a*b);//50
System.out.println(a/b);//2
System.out.println(a%b);//0
```

Output:

```
15
5
50
2
0
```

**Java Arithmetic Operator Example: Expression**

```java
class OperatorExample{
    public static void main(String args[]){
        System.out.println(10*10/5+3-1*4/2);
    }
}
```

Output:

```
21
```

**Java Left Shift Operator**
The Java left shift operator `<<` is used to shift all of the bits in a value to the left side of a specified number of times.

**Java Left Shift Operator Example**

```java
class OperatorExample{
    public static void main(String args[])
    {
        System.out.println(10<<2);//10*2^2=10*4=40
        System.out.println(10<<3);//10*2^3=10*8=80
        System.out.println(20<<2);//20*2^2=20*4=80
        System.out.println(15<<4);//15*2^4=15*16=240
    }
}
```

Output:

40
80
80
240

**Java Right Shift Operator**

The Java right shift operator `>>` is used to move left operands value to right by the number of bits specified by the right operand.

**Java Right Shift Operator Example**

```java
class OperatorExample{
    public static void main(String args[])
    {
        System.out.println(10>>2);//10/2^2=10/4=2
        System.out.println(20>>2);//20/2^2=20/4=5
        System.out.println(20>>3);//20/2^3=20/8=2
    }
}
```
Java Shift Operator Example: >> vs >>>

class OperatorExample{
    public static void main(String args[]){
        //For positive number, >> and >>> works same
        System.out.println(20>>2);
        System.out.println(20>>>2);
        //For negative number, >>> changes parity bit (MSB) to 0
        System.out.println(-20>>2);
        System.out.println(-20>>>2);
    }
}

Output:
2
5
2

Java AND Operator Example: Logical && and Bitwise &

The logical && operator doesn't check second condition if first condition is false. It checks second condition only if first one is true.
The bitwise & operator always checks both conditions whether first condition is true or false.

1. class OperatorExample{
2.  public static void main(String args[]){
3.    int a=10;
4.    int b=5;
5.    int c=20;
6.    System.out.println(a<b&&a<c);//false && true = false
7.    System.out.println(a<b&a<c);//false & true = false
8.  }
}

Output:
false
false
false

Java AND Operator Example: Logical && vs Bitwise &

class OperatorExample{
  public static void main(String args[]){
    int a=10;
    int b=5;
    int c=20;
    System.out.println(a<b&&a++<c);//false && true = false
    System.out.println(a);//10 because second condition is not checked
    System.out.println(a<b&a++<c);//false & true = false
    System.out.println(a);//11 because second condition is checked
  }
}

Output:
false
10
Java OR Operator Example: Logical || and Bitwise |

The logical || operator doesn't check second condition if first condition is true. It checks second condition only if first one is false.

The bitwise | operator always checks both conditions whether first condition is true or false.

```java
class OperatorExample{
    public static void main(String args[]){
        int a=10;
        int b=5;
        int c=20;
        System.out.println(a>b||a<c);//true || true = true
        System.out.println(a>b|a<c);//true | true = true
        //|| vs |
        System.out.println(a>b||a++<c);//true || true = true
        System.out.println(a);//10 because second condition is not checked
        System.out.println(a>b|a++<c);//true | true = true
        System.out.println(a);//11 because second condition is checked
    }
}
```

Output:

true
true
true
true
10
Java Ternary Operator

Java Ternary operator is used as one liner replacement for if-then-else statement and used a lot in Java programming. It is the only conditional operator which takes three operands.

Java Ternary Operator Example

class OperatorExample{
    public static void main(String args[]){
        int a=2;
        int b=5;
        int min=(a<b)?a:b;
        System.out.println(min);
    }
}

Output:

2

Another Example:

class OperatorExample{
    public static void main(String args[]){
        int a=10;
        int b=5;
        int min=(a<b)?a:b;
        System.out.println(min);
    }
}
Java Assignment Operator

Java assignment operator is one of the most common operator. It is used to assign the value on its right to the operand on its left.

Java Assignment Operator Example

class OperatorExample{
    public static void main(String args[]){
        int a=10;
        int b=20;
        a+=4;//a=a+4 (a=10+4)
        b-=4;//b=b-4 (b=20-4)
        System.out.println(a);
        System.out.println(b);
    }
}

Output:
5
14
16

Java Assignment Operator Example

class OperatorExample{
    public static void main(String[] args){
        int a=10;
        }
a+=3;//10+3
System.out.println(a);
a-=4;//13-4
System.out.println(a);
a*=2;//9*2
System.out.println(a);
a/=2;//18/2
System.out.println(a);
}

Output:
13
9
18
9

Java Assignment Operator Example: Adding short

class OperatorExample{
    public static void main(String args[]){
        short a=10;
        short b=10;
        //a+=b;//a=a+b internally so fine
        a=a+b;//Compile time error because 10+10=20 now int
        System.out.println(a);
    }
}
2.3 Java final variable

If you make any variable as final, you cannot change the value of final variable (It will be constant).

Example of final variable

There is a final variable speed limit, we are going to change the value of this variable, but it can't be changed because final variable once assigned a value can never be changed.

class Bike9{
    final int speedlimit=90; // final variable

    void run(){
        speedlimit=400;
    }
}
public static void main(String args[]){
    Bike9 obj=new Bike9();
    obj.run();
}
}//end of class

Output: Compile Time Error

2.4 Declaring 1D, 2D Array:

Types of Array in java

There are two types of array.

- Single Dimensional Array
- Multidimensional Array

Single Dimensional Array in Java

Syntax to Declare an Array in Java

1. `dataType[] arr;` (or)
2. `dataType []arr;` (or)
3. `dataType arr[];

Instantiation of an Array in Java

1. `arrayRefVar=new datatype[size];`

Example of Java Array

Let's see the simple example of java array, where we are going to declare, instantiate, initialize and traverse an array.

//Java Program to illustrate how to declare, instantiate, initialize
//and traverse the Java array.

class Testarray{
public static void main(String args[])
{
    int a[]=new int[5];//declaration and instantiation
    a[0]=10;//initialization
    a[1]=20;
    a[2]=70;
    a[3]=40;
    a[4]=50;
    //traversing array
    for(int i=0;i<a.length;i++)//length is the property of array
    System.out.println(a[i]);
}

Output:
• 10
• 20
• 70
• 40
• 50

Declaration, Instantiation and Initialization of Java Array

We can declare, instantiate and initialize the java array together by:

1. int a[]={33,3,4,5};//declaration, instantiation and initialization

Let's see the simple example to print this array.

   //Java Program to illustrate the use of declaration, instantiation
   //and initialization of Java array in a single line

class Testarray1{
    public static void main(String args[])
    {
        int a[] = {33, 3, 4, 5};
    
        //traversing array
        for (int i = 0; i < a.length; i++)
        
            System.out.println(a[i]);
    
    }}
int a[]={33,3,4,5}; //declaration, instantiation and initialization

//printing array
for(int i=0;i<a.length;i++)//length is the property of array
    System.out.println(a[i]);
}

Output:
33
3
4
5

Multidimensional Array in Java

In such case, data is stored in row and column based index (also known as matrix form).

Syntax to Declare Multidimensional Array in Java

1. dataType[][] arrayRefVar; (or)
2. dataType [][]arrayRefVar; (or)
3. dataType arrayRefVar[][]; (or)
4. dataType []arrayRefVar[];

Example to instantiate Multidimensional Array in Java

1. int[][] arr=new int[3][3];//3 row and 3 column

Example to initialize Multidimensional Array in Java

1. arr[0][0]=1;
2. arr[0][1]=2;
3. arr[0][2]=3;
4. arr[1][0]=4;
5. arr[1][1]=5;
6. arr[1][2]=6;
7. arr[2][0]=7;
Example of Multidimensional Java Array

Let's see the simple example to declare, instantiate, initialize and print the 2Dimensional array.

//Java Program to illustrate the use of multidimensional array

class Testarray3{
    public static void main(String args[]){
        //declaring and initializing 2D array
        int arr[][]={{1,2,3},{2,4,5},{4,4,5}};
        //printing 2D array
        for(int i=0;i<3;i++){
            for(int j=0;j<3;j++){
                System.out.print(arr[i][j]+" ");
            }
            System.out.println();
        }
    }
}

Output:

1 2 3
2 4 5
4 4 5
2.5 Accepting Input Using Command Line Argument

The java command-line argument is an argument i.e. passed at the time of running the java program.

The arguments passed from the console can be received in the java program and it can be used as an input.

So, it provides a convenient way to check the behavior of the program for the different values. You can pass N (1,2,3 and so on) numbers of arguments from the command prompt.

Simple example of command-line argument in java

In this example, we are receiving only one argument and printing it. To run this java program, you must pass at least one argument from the command prompt.

```java
class CommandLineExample{
    public static void main(String args[]){
        System.out.println("Your first argument is: "+args[0]);
    }
}
```

compile by > javac CommandLineExample.java

run by > java CommandLineExample sonoo

Output: Your first argument is: sonoo
Example of command-line argument that prints all the values

In this example, we are printing all the arguments passed from the command-line. For this purpose, we have traversed the array using for loop.

```java
class A{
    public static void main(String args[]){
        for(int i=0;i<args.length;i++)
            System.out.println(args[i]);
    }
}
```

compile by > javac A.java

run by > java A sonoo jaiswal 1 3 abc

Output: sonoo

    jaiswal
    1
    3
    abc
2.6 Accepting Input from console (USing BufferedReader Class)

BufferedReader class

BufferedReader class can be used to read data line by line by readLine() method.

Example of reading data from keyboard by InputStreamReader and BufferedReader class

In this example, we are connecting the BufferedReader stream with the InputStreamReader stream for reading the line by line data from the keyboard.

```java
import java.io.*;

class G5{
    public static void main(String args[])throws Exception{
        InputStreamReader r=new InputStreamReader(System.in);
        BufferedReader br=new BufferedReader(r);

        System.out.println("Enter your name");
        String name=br.readLine();
        System.out.println("Welcome "+name);
    }
}
```

Output: Enter your name
Amit

Welcome Amit