2.1 Writing Comments

The Java comments are the statements in a program that are not executed by the compiler and interpreter.

Types of Java Comments

There are three types of comments in Java.

1. Single Line Comment
    - This type of comment begins with a // and ends at the end of the line.
    - Example: //This method is used for adding two numbers.
2. Multi Line Comment
    - This types of comment must begins with /* and end with */.
    - A multiline comment may be several lines long.
    - Example: /*
      This method takes two parameters
      and returns its sum.
      */
3. Documentation Comment
    - This types of comment is used to produce an HTML file that documents our program.
    - We can use javadoc utility program to extract the information and put it into an HTML file.
    - The documentation comment begins with a /** and ends with a */.
    - Example: /**
      *This program can add two numbers.
      */

2.2 Basic Data Types

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.

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

here are 8 types of primitive data types:

- boolean data type
- byte data type
- char data type
- short data type
- int data type
- long data type
- float data type
- double data type

Data Type	Default Value	Default size
boolean	false	1 bit
char	'\u0000'	2 byte
byte	0	1 byte
short	0	2 byte
int	0	4 byte
long	0L	8 byte
float	0.0f	4 byte
double	0.0d	8 byte

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 isan 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 - 2,147,483,648and 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 - 9,223,372,036,854,775,808and 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'
```

Why char uses 2 byte in java and what is \u0000 ?

It is because java uses Unicode system (universal international standard character encoding that is capable of representing most of the world's written languages.) not ASCII code system. The \u0000 is the lowest range of Unicode system.

Escape Sequences

A character preceded by a backslash (\) is an *escape sequence* and has special meaning to the compiler. The following table shows the Java escape sequences:

Escape Sequences

Escape Sequence	Description
\t	Insert a tab in the text at this point.
\b	Insert a backspace in the text at this point.
\n	Insert a newline in the text at this point.
\r	Insert a carriage return in the text at this point.
\f	Insert a form feed in the text at this point.
\'	Insert a single quote character in the text at this point.
\"	Insert a double quote character in the text at this point.
\\	Insert a backslash character in the text at this point.

When an escape sequence is encountered in a print statement, the compiler interprets it accordingly. For example, if you want to put quotes within quotes you must use the escape sequence, \", on the interior quotes. To print the sentence

She said "Hello!" to me.

you would write

System.out.println("She said \"Hello!\" to me.");

2.3 Variables and Constants

Keywords:

- There are 50 keywords defined in the java language.
- These Keywords canot be used as identifiers such as names of variable, class or method.
- Following are the keywords defined in java:

abstarct	continue	for	new	switch
assert	default	goto	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp	volatile
const	float	native	super	while

Variables:

A variable is a container which holds the value while the Java program is executed. A variable is assigned with a data type.

Variable is a name of memory location. There are three types of variables in java:

1) Local Variable

A variable declared inside the body of the method is called local variable.

2) Instance Variable

A variable declared inside the class but outside the body of the method, is called an instance variable. It is not declared as static.

It is called an instance variable because its value is instance-specific and is not shared among instances.

3) Static variable

A variable that is declared as static is called a static variable. It cannot be local. You can create a single copy of the static variable and share it among all the instances of the class. Memory allocation for static variables happens only once when the class is loaded in the memory.

```
public class A
{
   static int m=100;//static variable
   void method()
   {
      int n=90;//local variable
   }
   public static void main(String args[])
   {
      int data=50;//instance variable
   }
}//end of class
```

Constant

- We denote constants in java using final keyword.
- The keyword final indicates that variable can only be assigned once and then its value cannot be changed.
- It is general practice in java to name constant in all uppercase.

  Example -

  Final int PI = 3.1415;

2.4 Operators

Operators are symbols that perform operations on variables and values.

Operators in Java can be classified into 5 types:

1. Arithmetic Operators

Arithmetic operators are used to perform arithmetic operations on variables and data.

Operator	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulo Operation (Remainder after division)

2. Assignment Operators

Assignment operators are used in Java to assign values to variables.

Operator	Example	Equivalent to
=	a = b;	a = b;
+=	a += b;	a = a + b;
-=	a -= b;	a = a - b;
*=	a *= b;	a = a * b;
/=	a /= b;	a = a / b;
%=	a %= b;	a = a % b;

3. Relational/Comparison Operators

Relational operators are used to check the relationship between two operands.

Operator	Description	Example
==	Is Equal To	3 == 5 returns **false**
!=	Not Equal To	3 != 5 returns **true**
>	Greater Than	3 > 5 returns **false**
<	Less Than	3 < 5 returns **true**
>=	Greater Than or Equal To	3 >= 5 returns **false**
<=	Less Than or Equal To	3 <= 5 returns **true**

4. Logical Operators

Logical operators are used to check whether an expression is true or false. They are used in decision making.

Operator	Example	Meaning
&& (Logical AND)	expression1 **&&** expression2	true only if both expression1 and expression2 are true
|| (Logical OR)	expression1 **||** expression2	true if either expression1 or expression2 is true
! (Logical NOT)	**!**expression	true if expression is false and vice versa

5. Unary Operators

Unary operators are used with only one operand. For example, ++ is a unary operator that increases the value of a variable by **1**. That is, ++5 will return **6**.

Different types of unary operators are:

Operator	Meaning
+	**Unary plus**: not necessary to use since numbers are positive without using it
-	**Unary minus**: inverts the sign of an expression
++	**Increment operator**: increments value by 1
--	**Decrement operator**: decrements value by 1
!	**Logical complement operator**: inverts the value of a boolean

6. Bitwise Operators

Bitwise operators in Java are used to perform operations on individual bits.

Operator	Description
~	Bitwise Complement
<<	Left Shift
>>	Right Shift
>>>	Unsigned Right Shift
&	Bitwise AND
^	Bitwise exclusive OR

Java Ternary Operator

The ternary operator (conditional operator) is shorthand for the `if-then-else` statement.

For example,

variable = Expression ? expression1 : expression2

Here's how it works.

- If the `Expression` is `true`, `expression1` is assigned to the `variable`.
- If the `Expression` is `false`, `expression2` is assigned to the `variable`.

2.5 Type Casting

In Java, **type casting** is a method or process that converts a data type into another data type in both ways manually and automatically. The automatic conversion is done by the compiler and manual conversion performed by the programmer.

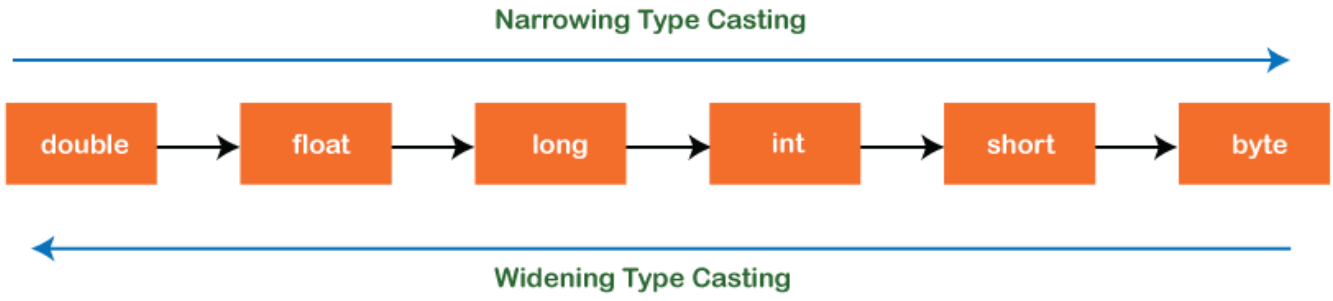
Type casting

Convert a value from one data type to another data type is known as **type casting**.

Types of Type Casting

There are two types of type casting:

- Widening Type Casting
- Narrowing Type Casting



Type Casting in Java

Widening Type Casting

Converting a lower data type into a higher one is called **widening** type casting. It is also known as **implicit conversion** or **casting down**. It is done automatically. It is safe because there is no chance to lose data. It takes place when:

- Both data types must be compatible with each other.
- The target type must be larger than the source type.

byte -> **short** -> **char** -> **int** -> **long** -> **float** -> **double**

For example, the conversion between numeric data type to char or Boolean is not done automatically. Also, the char and Boolean data types are not compatible with each other. Let's see an example.

WideningTypeCastingExample.java

```
public class WideningTypeCastingExample
{
public static void main(String[] args)
{
int x = 7;
//automatically converts the integer type into long type
long y = x;
//automatically converts the long type into float type
float z = y;
System.out.println("Before conversion, int value "+x);
System.out.println("After conversion, long value "+y);
System.out.println("After conversion, float value "+z);
}
}
```

Output

```
Before conversion, the value is: 7
After conversion, the long value is: 7
After conversion, the float value is: 7.0
```

In the above example, we have taken a variable x and converted it into a long type. After that, the long type is converted into the float type.

Narrowing Type Casting

Converting a higher data type into a lower one is called **narrowing** type casting. It is also known as **explicit conversion** or **casting up**. It is done manually by the programmer. If we do not perform casting then the compiler reports a compile-time error.

double -> **float** -> **long** -> **int** -> **char** -> **short** -> **byte**

Let's see an example of narrowing type casting.

NarrowingTypeCastingExample.java

```
public class NarrowingTypeCastingExample
{
public static void main(String args[])
{
double d = 166.66;
//converting double data type into long data type
long l = (long)d;
//converting long data type into int data type
int i = (int)l;
System.out.println("Before conversion: "+d);
//fractional part lost
System.out.println("After conversion into long type: "+l);
//fractional part lost
System.out.println("After conversion into int type: "+i);
}
}
```

Output

```
Before conversion: 166.66
After conversion into long type: 166
After conversion into int type: 166
```

2.6 Control Flow

Java provides three types of control flow statements.

Decision Making statements

- if statements
  - Simple if statement
  - if-else statement
  - if-else-if ladder
  - Nested if-statement
- switch statement

Loop statements

- do while loop
- while loop
- for loop
- for-each loop

```
String[] names = {"Java","C","C++","Python","JavaScript"};
System.out.println("Printing the content of the array names:\n");
for(String name:names) {
        System.out.println(name);
}
```

Jump statements

- break statement

```
for(int i=1;i<=10;i++){
    if(i==5){
        //breaking the loop
        break;
    }
    System.out.println(i);
```

- continue statement

```
for(int i=1;i<=10;i++){
    if(i==5){
       //using continue statement
       continue;//it will skip the rest statement
    }
    System.out.println(i);
}
```

If, else, elseif, switch, for, while, do..while, break,continue. Explain each with syntax, flowchart, example.

2.7 Arrays

Java array is an object which contains elements of a similar data type. Additionally, The elements of an array are stored in a contiguous memory location. It is a data structure where we store similar elements. We can store only a fixed set of elements in a Java array.

Array in Java is index-based, the first element of the array is stored at the 0th index, 2nd element is stored on 1st index and so on.

Single Dimensional Array in Java

Syntax to Declare an Array in Java

dataType[] arr; (or)

dataType []arr; (or)

dataType arr[];

Instantiation of an Array in Java

arrayRefVar=**new** datatype[size];

Example **//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

Another Example

```
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]);
```

For-each Loop for Java Array

We can also print the Java array using **for-each loop**.

Syntax:

```
for(data_type variable:array){
//body of the loop
```

```
    }
```

Example:

```
class Testarray1{
    public static void main(String args[]){
        int arr[]={33,3,4,5};
        //printing array using for-each loop
        for(int i:arr)
        System.out.println(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

```
dataType[][] arrayRefVar; (or)
dataType [][]arrayRefVar; (or)
dataType arrayRefVar[][]; (or)
dataType []arrayRefVar[];
```

Example to instantiate Multidimensional Array in Java

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

Example to initialize Multidimensional Array in Java

```
arr[0][0]=1;
arr[0][1]=2;
arr[0][2]=3;
arr[1][0]=4;
```

```
arr[1][1]=5;
arr[1][2]=6;
arr[2][0]=7;
arr[2][1]=8;
arr[2][2]=9;
```

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
```

ArrayIndexOutOfBoundsException

The Java Virtual Machine (JVM) throws an ArrayIndexOutOfBoundsException if length of the array in negative, equal to the array size or greater than the array size while traversing the array.

```
//Java Program to demonstrate the case of
//ArrayIndexOutOfBoundsException in a Java Array.
public class TestArrayException{
```

```
        public static void main(String args[]){
                int arr[]={50,60,70,80};
                for(int i=0;i<=arr.length;i++){
                        System.out.println(arr[i]);
                }
}}
```

Array Copying

We can assign one array variable to another, but then both variables refer to same array

Example

```
int[]arr = {1,2,3,4};
Int[] arr2 = arr1; // same as arr 1
```

If we actually want to copy all values of one array into a new array, we use the copyOf method in the Arrays class (**java.util.Arrays**)

Example:

```
Int[] copyOfArr1 = Arrays.copyOf (arr1, size);
```

/*size defines the actual size for new array **copyOfArr** 1 this could be of same, less or more than that of size of arr 1 If size is more than that of arr 1 then additional elements will be 0 (for integers), false(for booleans null (for reference types) If size is less than that of arr 1 then only the initial values are copied. */