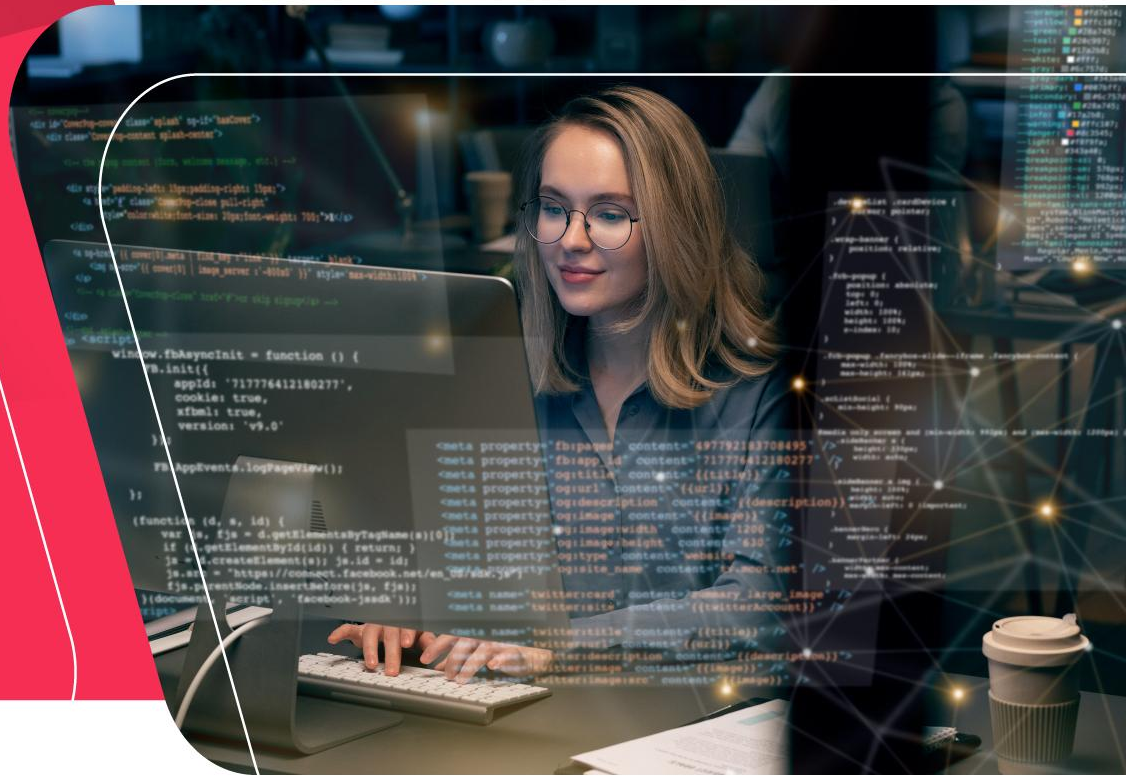
TIA-Academy

Java Bootcamp

Day 04



Technologies will be Use

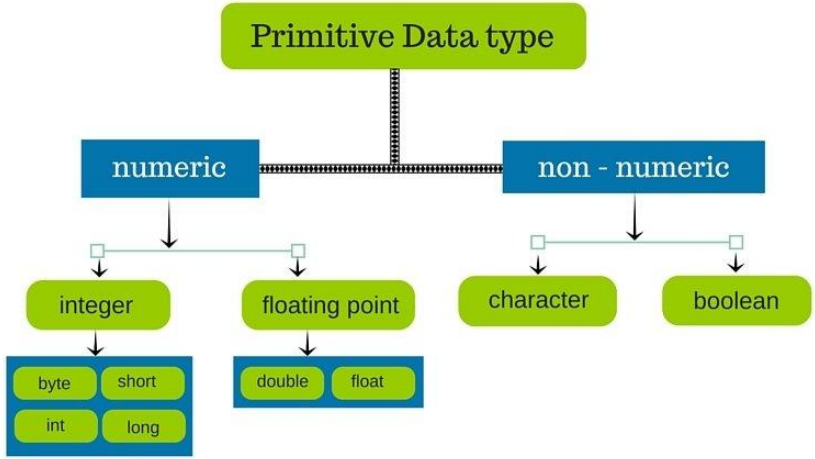
- JDK 8/**11**/15
- JRE 8/**11**/15
- Writing Code using Notepad++ (For Windows) or Vim (For Linux and Mac)
- Compiling with Command Prompt (For Windows) or Terminal (For Linux and Mac)
- ***Forbidden of using IDE Based Development Tools as of Intellij IDEA or Eclipse***

TIA-Academy

Java Data Types (Primitive)

Java Data Types

- In this tutorial, we will learn about all **8 primitive data types in Java** with the help of examples.
- As the name suggests, data types specify the **type of data that can be stored inside variables in Java.**
- Java is a statically-typed language. This means that all variables must be declared before they can be used.



Java Data Types

```
int speed;
```

- Here, **speed is a variable**, and the **data type of the variable is int**.
- The **int** data type determines that the speed variable **can only contain integers**.
- There are 8 data types predefined in Java, known as primitive data types.
- Note: In addition to primitive data types, **there are also referenced types (object type).**

1. boolean type

- The boolean data type **has two possible values, either true or false.**
- Default value: false.
- They are usually used for true/false conditions.

```
class Main {
  public static void main(String[] args) {
    boolean flag = true;
    System.out.println(flag);
    // prints true
  }
}
```

2. byte type

- The byte data type can have values from -128 to 127 (8-bit signed two's complement integer).
- **If it's certain that the value of a variable will be within -128 to 127, then it is used instead of int to save memory.**
- Default value: 0

```
class Main {
  public static void main(String[] args) {
    byte range;
    range = 124;
    System.out.println(range);
    // prints 124
  }
}
```

3. short type

- The short data type in Java can have values from -32768 to 32767 (16-bit signed two's complement integer).
- **If it's certain that the value of a variable will be within -32768 and 32767, then it is used instead of other integer data types (int, long).**
- Default value: 0

```
class Main {
  public static void main(String[] args) {
    short temperature;
    temperature = -200;
    System.out.println(temperature);
    // prints -200
  }
}
```

4. int type

- The int data type can have values from -2^{31} to $2^{31}-1$ (32-bit signed two's complement integer).
- **If you are using Java 8 or later, you can use an unsigned 32-bit integer. This will have a minimum value of 0 and a maximum value of $2^{32}-1$.**
- Default value: 0

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

    int range = -4250000;
    System.out.println(range);
    // print -4250000
  }
}
```

5. long type

- The long data type can have values from -2^{63} to $2^{63}-1$ (64-bit signed two's complement integer).
- If you are using Java 8 or later, you can use an unsigned 64-bit integer with a minimum value of 0 and a maximum value of $2^{64}-1$.
- Default value: 0
- Notice**, the use of L at the end of -42332200000. This represents that it's an integer of the long type**.

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

    long range = -42332200000L;
    System.out.println(range);
    // prints -42332200000
  }
}
```

6. double type

- The double data type is a double-precision 64-bit floating-point.
- Default value: 0.0 (0.0d)

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

    double number = -42.3;
    System.out.println(number);
    // prints -42.3
  }
}
```

7. float type

- The float data type is a single-precision 32-bit floating-point. Learn more about single-precision and double-precision floating-point if you are interested.
- Default value: 0.0 (0.0f)
- Notice that **we have used -42.3f instead of -42.3 in the above program. It's because -42.3 is a double literal.**
- **To tell the compiler to treat -42.3 as float rather than double, you need to use f or F.**

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

    float number = -42.3f;
    System.out.println(number);
    // prints -42.3
  }
}
```

8. char type

- It's a 16-bit Unicode character.
- The minimum value of the char data type is '\u0000' (0) and the maximum value of the is '\uffff'.
- Defaul
- Here, the **Unicode value of Q is \u0051**. Hence, we get Q as the output.
- t value: '\u0000'

```
class Main {
  public static void main(String[] args) {
    char letter = '\u0051';
    System.out.println(letter);  // prints Q
  }
}
```

8. char type

- Here, we have assigned 9 as a character (specified by single quotes) to the letter1 variable. However, the letter2 variable is assigned 65 as an integer number (no single quotes).
- Hence, A is printed to the output. It is because **Java treats characters as an integer** and the ASCII value of A is 65.

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

    char letter1 = '9';
    System.out.println(letter1);  // prints 9

    char letter2 = 65;
    System.out.println(letter2);  // prints A

  }
}
```

Java's primitive types

Ordered by descending upper range limit

CATEGORIZATION	NAME	UPPER RANGE	LOWER RANGE	BITS
Floating point	double	Really huge positive	Really huge negative	64
	float	Huge positive	Huge negative	32
Integral	long	9,223,372,036,854,775,807	-9,223,372,036,854,775,808	64
	int	2,147,483,647	-2,147,483,648	32
	char	65,535	0	16
	short	32,767	-32,768	16
	byte	127	-128	8
Boolean	boolean	No numeric equivalent	true or false	1

String type

- Java also provides support for character strings via java.lang.String class. **Strings in Java are not primitive types. Instead, they are objects**. For example,

```
String myString = "Java Programming";
```

- Here, myString is an **object** of the String class.

TIA-Academy

ASSIGNMENT 01

TIA-Academy

Java Operators

Java Operators

- JVM (Java Virtual Machine) is an **abstract machine that enables your computer to run a Java program.**
- When you run the Java program, Java compiler first compiles your Java code to bytecode. Then, the JVM translates **bytecode** into **native machine code (set of instructions that a computer's CPU executes directly).**

Java Operators

Operators in Java can be classified into 6 types:

1. **Arithmetic Operators**
2. **Assignment Operators**
3. **Relational Operators**
4. **Logical Operators**
5. *Unary Operators*
6. *Bitwise Operators*

1. Java Arithmetic Operators

- Arithmetic operators are used to perform arithmetic operations on variables and data. For example,

```
a + b;
```

- Here, the **+ operator is used to add two variables a and b**. Similarly, there are various other arithmetic operators in Java.

1. Java Arithmetic Operators

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

1. Java Arithmetic Operators

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

    // declare variables
    int a = 12, b = 5;

    // addition operator
    System.out.println("a + b = " + (a + b));

    // subtraction operator
    System.out.println("a - b = " + (a - b));

    // multiplication operator
    System.out.println("a * b = " + (a * b));

    // division operator
    System.out.println("a / b = " + (a / b));

    // modulo operator
    System.out.println("a % b = " + (a % b));
  }
}
```

1. Java Arithmetic Operators

- Output

```
a + b = 17
a - b = 7
a * b = 60
a / b = 2
a % b = 2
```

- In the above example, we have used +, -, and * operators to compute addition, subtraction, and multiplication operations.

1. Java Arithmetic Operators

- / Division Operator
- Note the operation, a / b in our program. The / operator is the division operator.
- If we use the division operator with two integers, then the resulting quotient will also be an integer. And, if one of the operands is a floating-point number, we will get the result will also be in floating-point.

```
In Java,

(9 / 2) is 4
(9.0 / 2) is 4.5
(9 / 2.0) is 4.5
(9.0 / 2.0) is 4.5
```

1. Java Arithmetic Operators

- % Modulo Operator
- The modulo operator % computes the remainder.
- When a = 7 is divided by b = 4, the remainder is 3.
- Note: The % operator is mainly used with integers.

2. Java Assignment Operators

- Assignment operators are used in Java to assign values to variables. For example,

```
int age;
age = 5;
```

- Here, = is the assignment operator. It assigns the value on its right to the variable on its left. That is, **5 is assigned to the variable age**.

2. Java Assignment Operators

- Let's see some more assignment operators available in Java.

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;`

2. Java Assignment Operators

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

    // create variables
    int a = 4;
    int var;

    // assign value using =
    var = a;
    System.out.println("var using =: " + var);

    // assign value using =+
    var += a; //var = var + a
    System.out.println("var using +=: " + var);

    // assign value using =*
    var *= a;
    System.out.println("var using *=: " + var);
  }
}
```

2. Java Assignment Operators

- Output

```
var using =: 4
var using +=: 8
var using *=: 32
```

3. Java Relational Operators

- Relational operators are used to check the relationship between two operands. For example,

```
// check if a is less than b
a < b;
```

- Here, < operator is the relational operator. It checks if **a is less than b** or not.
- It **returns either true or false**.

3. Java Relational Operators

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**

3. Java Relational Operators

- Note: Relational operators are used in decision making and loops.

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

    // create variables
    int a = 7, b = 11;

    // value of a and b
    System.out.println("a is " + a + " and b is " + b);

    // == operator
    System.out.println(a == b);  // false

    // != operator
    System.out.println(a != b);  // true

    // > operator
    System.out.println(a > b);  // false

    // < operator
    System.out.println(a < b);  // true

    // >= operator
    System.out.println(a >= b);  // false

    // <= operator
    System.out.println(a <= b);  // true
  }
}
```

4. Java 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

4. Java Logical Operators

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

    // && operator
    System.out.println( (5 > 3) && (8 > 5));  // true
    System.out.println( (5 > 3) && (8 < 5));  // false

    // || operator
    System.out.println((5 < 3) ||  (8 > 5));  // true
    System.out.println( (5 > 3) || (8 < 5));  // true
    System.out.println((5 < 3) || (8 < 5));  // false

    // ! operator
    System.out.println(!(5 == 3));  // true
    System.out.println(!(5 > 3));  // false
  }
}
```

4. Java Logical Operators

Working of Program

- (5 > 3) && (8 > 5) returns true because both (5 > 3) and (8 > 5) are true.
- (5 > 3) && (8 < 5) returns false because the expression (8 < 5) is false.
- (5 < 3) || (8 > 5) returns true because the expression (8 > 5) is true.
- (5 > 3) && (8 > 5) returns true because the expression (5 > 3) is true.
- (5 > 3) && (8 > 5) returns false because both (5 < 3) and (8 < 5) are false.
- !(5 == 3) returns true because 5 == 3 is false.
- !(5 > 3) returns false because 5 > 3 is true.

5. Java 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

Increment and Decrement Operators

- Java also provides increment and decrement operators: ++ and -- respectively. ++ increases the value of the operand by 1, while -- decrease it by 1.
- For example,

```
int num = 5;

// increase num by 1
++num;
```

- Here, the value of **num** gets increased to 6 from its initial value of 5.

Increment and Decrement Operators

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

    // declare variables
    int a = 12, b = 12;
    int result1, result2;

    // original value
    System.out.println("Value of a: " + a);

    // increment operator
    result1 = ++a;
    System.out.println("After increment: " + result1);

    System.out.println("Value of b: " + b);

    // decrement operator
    result2 = --b;
    System.out.println("After decrement: " + result2);
  }
}
```

Increment and Decrement Operators

- Output

```
Value of a: 12
After increment: 13
Value of b: 12
After decrement: 11
```

- In the above program, we have used the ++ and -- operator as prefixes (++a, --b). We can also use these operators as postfix (a++, b++).
- There is a slight difference when these operators are used as prefix versus when they are used as a postfix.

6. Java Bitwise Operators

- Bitwise operators in Java are used to perform operations on individual bits. For example,

```
Bitwise complement Operation of 35

35 = 00100011 (In Binary)

~ 00100011
  ________
  11011100  = 220 (In decimal)
```

- Here, ~ is a bitwise operator. It inverts the value of each bit (0 to 1 and 1 to 0).

6. Java Bitwise Operators

- The various bitwise operators present in Java are:

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

- **These operators are not generally used in Java.**

Other Operators - Java instanceof Operator

- The instanceof operator checks whether an object is an instanceof a particular class.
- For example,

```
class Main {
  public static void main(String[] args) {
    String str = "Programiz";
    boolean result;

    // checks if str is an instance of
    // the String class
    result = str instanceof String;
       System.out.println("Is  str  an  object  of  String?  "  +
result);
  }
}
```

Other Operators - Java instanceof Operator

- Output

```
Is str an object of String? true
```

- Here, str is an instance of the String class.
- Hence, the instanceof operator returns true.

Other Operators - 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.

Other Operators - Java Ternary Operator

- Output

```
Leap year
```

- In the above example, we have used the ternary operator to check if the year is a leap year or not.

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

    int februaryDays = 29;
    String result;

    // ternary operator
    result = (februaryDays == 28) ? "Not a leap year" : "Leap year";
    System.out.println(result);
  }
}
```

Java Operator Precedence

- Operator precedence determines the order in which the operators in an expression are evaluated
- Now, take a look at the statement below:

```
int myInt = 12 - 4 * 2;
```

- What will be the value of myInt? Will it be (12 - 4)*2, that is, 16? Or it will be 12 - (4 * 2), that is, 4?
- When **two operators share a common operand**, 4 in this case, the operator with the highest precedence is operated first.
- In Java, the precedence of * is higher than that of -. Hence, the multiplication is performed before subtraction, and the value of myInt will be 4.

Operator Precedence Table

Java Operator Precedence	
Operators	Precedence
postfix increment and decrement	`++` `--`
prefix increment and decrement, and unary	`++` `--` `+` `-` `~` `!`
multiplicative	`*` `/` `%`
additive	`+` `-`
shift	`<<` `>>` `>>>`
relational	`<` `>` `<=` `>=` `instanceof`
equality	`==` `!=`
bitwise AND	`&`
bitwise exclusive OR	`^`
bitwise inclusive OR	`|`
logical AND	`&&`
logical OR	`||`
ternary	`? :`
assignment	`=` `+=` `-=` `*=` `/=` `%=` `&=` `^=` `|=` `<<=` `>>=` `>>>=`

- The table beside lists the precedence of operators in Java; higher it appears in the table, the higher its precedence.

Operator Precedence Table

- Output

2

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

        int a = 10, b = 5, c = 1, result;
        result = a-++c-++b;

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

Associativity of Operators in Java

- If an expression has two operators with similar precedence, the expression is evaluated according to its associativity (either left to right, or right to left). Let's take an example.

```
a = b = c;
```

- Here, the value of c is assigned to variable b. Then the value of b is assigned of variable a. Why? It's because the associativity of = operator is from right to left.

Associativity of Operators in Java

Java Operator Precedence and Associativity		
Operators	**Precedence**	**Associativity**
postfix increment and decrement	`++` `--`	left to right
prefix increment and decrement, and unary	`++` `--` `+` `-` `~` `!`	right to left
multiplicative	`*` `/` `%`	left to right
additive	`+` `-`	left to right
shift	`<<` `>>` `>>>`	left to right
relational	`<` `>` `<=` `>=` `instanceof`	left to right
equality	`==` `!=`	left to right
bitwise AND	`&`	left to right
bitwise exclusive OR	`^`	left to right
bitwise inclusive OR	`|`	left to right
logical AND	`&&`	left to right
logical OR	`||`	left to right
ternary	`? :`	right to left
assignment	`=` `+=` `-=` `*=` `/=` `%=` `&=` `^=` `|=` `<<=` `>>=` `>>>=`	right to left

- The table beside shows the associativity of Java operators along with their associativity.

Associativity of Operators in Java

- **You don't need to memorize everything here.**
- Most of the time, the precedence and associativity of operators makes sense in itself.
- You can always come back to this article for reference when in doubt.
- Also, you can use parenthesis if you think it makes your code easier to understand.

TIA-Academy

ASSIGNMENT 01

TIA-Academy

Java Basic
Input and Output

Java Output

- In Java, you can simply use

```
System.out.println(); or
System.out.print(); or
System.out.printf();
```

- to send output to standard output (screen).

Here,

- System is a class
- out is a public static field: it accepts output data.

- Don't worry if you don't understand it. **We will discuss class, public, and static in later chapters**.

Java Output

- Let's take an example to output a line.
- Output:

```
Java programming is interesting.
```

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

        System.out.println("Java programming is interesting.");

    }
}
```

- Here, we have used the println() method to display the string.

Difference between println(), print() and printf()

- **print()** - It prints string inside the quotes.
- **println()** - It prints string inside the quotes similar like print() method. **Then the cursor moves to the beginning of the next line.**
- **printf()** - It provides string formatting (similar to printf in C/C++ programming).

print() and println()

- Output:

```
1. println
2. println
1. print 2. print
```

- In the above example, we have shown the working of the print() and println() methods.

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

        System.out.println("1. println ");
        System.out.println("2. println ");

        System.out.print("1. print ");
        System.out.print("2. print");
    }
}
```

Printing Variables and Literals

- When you run the program, the output will be:

```
5
-10.6
```

- Here, you can see that we have not used the quotation marks. **It is because to display integers, variables and so on, we don't use quotation marks**.

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

        Double number = -10.6;

        System.out.println(5);
        System.out.println(number);
    }
}
```

Print Concatenated Strings

- Output:

```
I am awesome.
Number = -10.6
```

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

        Double number = -10.6;

        System.out.println("I am " + "awesome.");
        System.out.println("Number = " + number);
    }
}
```

Print Concatenated Strings

- In the above example, notice the line,

```
System.out.println("I am " + "awesome.");
```

- Here, **we have used the + operator to concatenate (join) the two strings: "I am " and "awesome.".**
- And also, the line,

```
System.out.println("Number = " + number);
```

- Here, first the value of **variable number is evaluated**. Then, the value is **concatenated to the string: "Number = ".**

Java Input

- Java provides different ways to get input from the user. However, in this tutorial, you will learn to get input from user using the object of Scanner class.
- In order to use the object of Scanner, we need **to import java.util.Scanner package.**

```
import java.util.Scanner;
```

- Then, we need to create an object of the Scanner class. We can use the object to take input from the user.

```
// create an object of Scanner
Scanner input = new Scanner(System.in);

// take input from the user
int number = input.nextInt();
```

Get Integer Input From the User

- Output:

```
Enter an integer: 23
You entered 23
```

```
import java.util.Scanner;

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

        Scanner input = new Scanner(System.in);

        System.out.print("Enter an integer: ");
        int number = input.nextInt();
        System.out.println("You entered " +
number);

        // closing the scanner object
        input.close();
    }
}
```

Java Input

- In the above example, we have created an object named input of the Scanner class. We then call the **nextInt() method of the Scanner class to get an integer input from the user.**
- Similarly, we can **use nextLong(), nextFloat(), nextDouble(), and next() methods to get long, float, double, and string input respectively from the user.**
- Note: We have used the close() method to close the object. **It is recommended to close the scanner object once the input is taken**.

Get float, double and String Input

```
import java.util.Scanner;

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

        Scanner input = new Scanner(System.in);

        // Getting float input
        System.out.print("Enter float: ");
        float myFloat = input.nextFloat();
        System.out.println("Float entered = " + myFloat);

        // Getting double input
        System.out.print("Enter double: ");
        double myDouble = input.nextDouble();
        System.out.println("Double entered = " + myDouble);

        // Getting String input
        System.out.print("Enter text: ");
        String myString = input.next();
        System.out.println("Text entered = " + myString);

        input.close();
    }
}
```

Get float, double and String Input

- Output:

```
Enter float: 2.343
Float entered = 2.343
Enter double: -23.4
Double entered = -23.4
Enter text: Hey!
Text entered = Hey!
```

- As mentioned, there are other several ways to get input from the user.

TIA-Academy

ASSIGNMENT 03

TIA-Academy

Java Expressions, Statements and Blocks

Java Expressions, Statements and Blocks

- In this tutorial, you will learn about Java expressions, Java statements, difference between expression and statement, and Java blocks with the help of examples.
- In previous chapters, we have used expressions, statements, and blocks without much explaining about them.
- Now that you know about variables, operators, and literals, it will be easier to understand these concepts.

Java Expressions

- A Java **expression** consists of **variables, operators, literals, and method calls**. For example,

```
int score;
score = 90;
```

- Here, **score = 90** is an **expression** that returns an int. Consider another example,

```
Double a = 2.2, b = 3.4, result;
result = a + b - 3.4;
```

Java Expressions

- Here, a + b - 3.4 is an expression.

```
if (number1 == number2)
    System.out.println("Number 1 is larger than number 2");
```

- Here, **number1 == number2** is an expression that returns a boolean value. Similarly, "Number 1 is larger than number 2" is a string expression.

Java Statements

- In Java, each statement is a complete unit of execution. For example,

```
int score = 9*5;
```

- Here, we have a statement. The complete execution of this statement involves **multiplying integers 9 and 5** and then assigning the result to the variable score.
- In the above statement, we have an expression 9 * 5. In Java, expressions are **part of statements**.

Expression statements

- We can convert an expression into a statement by terminating the expression with a **;**. These are known as **expression statements**. For example,

```
// expression
number = 10
// statement
number = 10;
```

- In the above example, we have an expression **number = 10**. Here, by adding a semicolon **(;),** we have **converted** the expression into a statement (number = 10;).

Expression statements

- Consider another example,

```
// expression
++number
// statement
++number;
```

- Similarly, ++number is an expression whereas ++number; is a statement.

Declaration Statements

- In Java, **declaration statements** are used for declaring variables. For example,

```
double tax = 9.5;
```

- The statement above declares a variable tax which is initialized to 9.5.
- Note: There are control flow statements that are used in decision making and looping in Java. You will learn about control flow statements in later chapters.

Java Blocks

- A block is a group of statements (zero or more) that is enclosed in curly braces { }. For example,

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

        String band = "Beatles";

    {
          int a = 5;
          System.out.println("Azizi "+ a) ;
    }

        if (band == "Beatles") { // start of block
            System.out.print("Hey ");
            System.out.print("Jude!");
        } // end of block
    }
}
```

Java Blocks

- Output:

```
Hey Jude!
```

- In the above example, we have a **block if** {....}.
- Here, inside the block we have **two statements**:

```
System.out.print("Hey ");
System.out.print("Jude!");
```

Java Blocks

- However, **a block may not have any statements**. Consider the following examples,
- This is a valid Java program. Here, we have a block if {...}. However, there is no any statement inside this block.

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

        if (10 > 5) { // start of block

        } // end of block
    }
}
```

Java Blocks

- Here, we have block **public static void main()** {...}. However, similar to the above example, this block does not have any statement.

```
class AssignmentOperator {
    public static void main(String[] args) {   // start of block



    } // end of block
}
```

TIA-Academy

Java Comments

Java Comments

- In this tutorial, you will learn about Java comments, why we use them, and how to use comments in right way.
- In computer programming, comments are a portion of the program that are completely ignored by Java compilers. They are **mainly used to help programmers to understand the code.** For example,

```
// declare and initialize two variables
int a = 1;
int b = 3;

// print the output
System.out.println("This is output");
```

- Here, we have used the following comments,
  - declare and initialize two variables
  - print the output

Java Comments

In Java, there are two types of comments:

- single-line comment
- multi-line comment

Single-line Comment

- A single-line comment starts and ends in the same line. To write a single-line comment, we can use the **// symbol**.
- For example,
- Output:

```
Hello, World!
```

```
// "Hello, World!" program example

class Main {
    public static void main(String[] args) {
    {
        // prints "Hello, World!"
        System.out.println("Hello, World!");
    }
}
```

Single-line Comment

- Here, we have used two single-line comments:
  - "Hello, World!" program example
  - prints "Hello World!“
- The Java compiler ignores everything from // to the end of line. Hence, it is also known as End of Line comment.

Multi-line Comment

- When we want to write comments in multiple lines, we can use the multi-line comment. To write multi-line comments, we can use the **/*....*/ symbol**. For example,
- Output:

```
Hello, World!
```

```
/* This is an example of  multi-line comment.
 * The program prints "Hello, World!" to the standard
output.
 */

class HelloWorld {
    public static void main(String[] args) {
    {
        System.out.println("Hello, World!");
    }
}
```

Multi-line Comment

- Here, we have used the multi-line comment:

```
/* This is an example of multi-line comment.
* The program prints "Hello, World!" to the standard output.
*/
```

- This type of comment is also known as Traditional Comment. In this type of comment, the Java compiler ignores everything from /* to */.

Use Comments the Right Way

- One thing you should always consider that comments shouldn't be the substitute for a way to explain poorly written code in English. You should always write well structured and self explaining code. And, then use comments.
- Some believe that **code should be self-describing and comments should be rarely used**. However, in my personal opinion, there is nothing wrong with using comments. **We can use comments to explain complex algorithms, regex or scenarios where we have to choose one technique among different technique to solve problems**.
- **Note**: In most cases, always use comments to explain '**why**' rather than '**how**' and you are good to go.

TIA-Academy

ASSIGNMENT 04
(HOME ASSIGNMENT)

TIA-Academy

Thank You