ADVANCED INFORMATION TECHNOLOGY INSTITUTE

GHANA-INDIA KOFI ANNAN CENTRE OF EXCELLENCE IN ICT

INTRODUCTION TO CORE JAVA

Table of Contents

4. **Conditional Statements**
    1. If Statement
    2. Else Statement
    3. Else if Statement
    4. Switch Statement

    Loops
    1. While Loop
    2. Do while Loop
    3. For Loop
    4. Foreach Loop
    5. Break/Continue Loop

5. **Arrays**

6. Java Methods
7. Java Interface
8. Java – Strings
9. Java - Exception Handling
10. Java – JSP
11. Java – JDBC(Java Database Connectivity)
12. Java – Spring Framework

WHAT IS JAVA PROGRAMMING?

- **Java Programming** is a high-level, class-based, object-oriented **programming** language that is designed to have as few implementation dependencies. Java is used to develop mobile apps, web apps, desktop apps, games and much more. Java is a high-level programming language originally developed by Sun Microsystems and released in 1995. Java runs on a variety of platforms, such as Windows, Mac OS, and the various versions of UNIX.

- The latest release of the Java Standard Edition is Java SE 18.0.1.1. With the advancement of Java and its widespread popularity, multiple configurations were built to suit various types of platforms. For example: J2EE for Enterprise Applications, J2ME for Mobile Applications. The new J2 versions were renamed as Java SE, Java EE, and Java ME respectively. Java is guaranteed to be **Write Once, Run Anywhere.**

WHY JAVA PROGRAMMING?

1. **Object Oriented**: In Java, everything is an Object. Java can be easily extended since it is based on the Object model.
2. **Platform Independent:** Unlike many other programming languages including C and C++, when Java is compiled, it is not compiled into platform specific machine, rather into platform independent byte code. This byte code is distributed over the web and interpreted by the Virtual Machine (JVM) on whichever platform it is being run on.
3. **Simple:** Java is designed to be easy to learn. If you understand the basic concept of OOP Java, it would be easy to master.
4. **Secure:** With Java's secure feature it enables to develop virus-free, tamper-free systems. Authentication techniques are based on public-key encryption.
5. **Architecture-neutral:** Java compiler generates an architecture-neutral object file format, which makes the compiled code executable on many processors, with the presence of Java runtime system.
6. **Portable**: Being architecture-neutral and having no implementation dependent aspects of the specification makes Java portable. Compiler in Java is written in ANSI C with a clean portability boundary, which is a POSIX subset.

7. **Robust:** Mishandled runtime errors and memory management mistakes are the two main problems which cause program failures. Java can handle these issues with high efficiency. Mishandled runtime errors could be resolved through Exception Handling protocol, while memory management mistakes could be resolved by garbage collection, which is an automatic de-allocation of objects that are already unnecessary.

8. **Multithreaded:** With Java's multithreaded feature it is possible to write programs that can perform many tasks simultaneously. This design feature allows the developers to construct interactive applications that can run smoothly.

9. **Interpreted:**
Java byte code is translated on the fly to native machine instructions and is not stored anywhere. The development process is more rapid and analytical since the linking is an incremental and light-weight process.

10. **High Performance:** With the use of Just-In-Time compilers, Java enables high performance.

11. **Distributed:** Java is designed for the distributed environment of the internet.

12. **Dynamic:** Java is considered to be more dynamic than C or C++ since it is designed to adapt to an evolving environment. Java programs can carry extensive amount of run-time information that can be used to verify and resolve accesses to objects on run-time.

JAVA ENVIRONMENT SETUP

Setting up the PATH for windows:

- Download the JDK Setup at https://www.oracle.com/java/technologies/javase/jdk15-archive-downloads.html#license-lightbox
- Double click the jdk-15.0.2_windows-x64_bin.exe file to install.

Popular Java Text Editors

Notepad: On Windows machine, you can use any simple text editor like Notepad (Recommended for this tutorial).

VS Code: An IDE developed by Microsoft which can be downloaded free from https://code.visualstudio.com/download

Netbeans: A Java IDE that is open-source and free which can be downloaded free from https://www.netbeans.org/index.html.

Intellij: A Java IDE developed by jetBrains which can be downloaded free from https://www.jetbrains.com/idea/download/

Eclipse: A Java IDE developed by the eclipse open-source community and can be downloaded from https://www.eclipse.org/

JAVA SYNTAX

About Java programs, it is very important to keep in mind the following points:

- **Case Sensitivity:** Java is case sensitive, which means identifier **Hello** and **hello** would have different meaning in Java.
- **Class Names:** For all class names the first letter should be in Upper Case. If several words are used to form a name of the class, each inner word's first letter should be in Upper Case. **Example:** *class* ***MyFirstJavaClass***
- **Method Names:** All method names should start with a Lower Case letter. If several words are used to form the name of the method, then each inner word's first letter should be in Upper Case. **Example:** *public void* **myMethodName()**
- **Program File Name:** Name of the program file should exactly match the class name. When saving the file, you should save it using the class name (Remember Java is case sensitive) and append '.java' to the end of the name (if the file name and the class name do not match, your program will not compile). But please make a note that in case you do not have a public class present in the file then file name can be different than class name. It is also not mandatory to have a public class in the file. **Example:** Assume '**MyFirstJavaProgram**' is the class name. Then the file should be saved as *'MyFirstJavaProgram.java'*.
- **public static void main(String args[ ]):** Java program processing starts from the main() method which is a mandatory part of every Java program.

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

JAVA SYNTAX

Note: Every line of code that runs in java must be inside a class. Java is case-sensitive: "MyFirstJavaClass" and "myfirstjavaclass" has different meaning.

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

The main Method: Any code inside the **main()** method will be executed. Don't worry about the keywords before and after main.

```
public static void main(String[ ]  args)
```

System.out.println(): Inside the **main()** method, we can use the **println()** method to print a line of text to the screen.

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

Note: The curly braces { } marks the beginning and the end of a block of code. **System** is a built-in Java class that contains useful methods, such as **out**. The **println()** method, short for "print line", is used to print a value to the screen (or a file).
You should also note that each code statement must end with a semicolon (;).

JAVA IDENTIFIERS

All Java components require names. Names used for classes, variables, and methods are called **identifiers.** In Java, there are several points to remember about identifiers. They are as follows;

- All identifiers should begin with a letter (A to Z or a to z), currency character ($) or an underscore (_).
- After the first character, identifiers can have any combination of characters.
- A key word cannot be used as an identifier.
- Most importantly, identifiers are case sensitive.
- Examples of legal identifiers: age, $salary, _value, __1_value.
- Examples of illegal identifiers: 123abc, -salary.

JAVA MODIFIERS

Like other languages, it is possible to modify classes, methods, etc., by using modifiers. There are two categories of modifiers.

- **Access Modifiers:** default, public , protected, private
- **Non-access Modifiers:** final, abstract.

DATA TYPES IN JAVA

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:** A primitive data type specifies the size and type of variable values, and it has no additional methods.

Data Types	Size	Description
byte	1 byte	Stores whole numbers from -128 to 127
short	2 bytes	Stores whole numbers from -32,768 to 32,767
Int	4 bytes	Stores whole numbers from -2,147,483,648 to 2,147,483,647
Long	8 bytes	Stores whole numbers from -2,147,483,648 to 2,147,483,647
float	4 bytes	Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits
double	8 bytes	Stores fractional numbers. Sufficient for storing 15 decimal digits
boolean	1 bit	Stores true or false values
char	2 bytes	Stores a single character/letter or ASCII values

DATA TYPES IN JAVA

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. **Non Primitive Data Types:** Non-primitive data types are called **reference types** because they refer to objects. Non-primitive data types includes; String, Classes, Interfaces and Arrays.

The main difference between **primitive** and **non-primitive** data types are:

Primitive Data Types	Non Primitive Data Types
Primitive types are predefined (already defined) in Java	Non-primitive types are created by the programmer and is not defined by Java (except for String)
Primitive types can not be used to call methods to perform certain operations	Non-primitive types can be used to call methods to perform certain operations
A primitive type has always a value	Non-primitive types can be null
A primitive type starts with a lowercase letter	Non-primitive types starts with an uppercase letter
The size of a primitive type depends on the data type	Non-primitive types have all the same size

Primitive Data Types

- **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 = True;

- **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 = 1000, short r = 5000;

Primitive Data Types

- **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 num = 1000000, int b = 50000;

- **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 num = 100000L, long b = 20000L;

- **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 f = 234.5f;

Primitive Data Types

- **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 d = 12.3;

- **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 f = 234.5f;

- **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: It is because java uses Unicode system not ASCII code system. The \u0000 is the lowest range of Unicode system.

Non-Primitive Data Types

- String Data Type: Is the collection or sequence of characters surrounded by double quotes: Example;

```
public class MyFirstJavaClass  {
        public static void main(String[ ]  args) {
                String greetings = new String("Hello World");
                System.out.println(greetings);
        }
}
```

Note: A String in Java is actually an object, which contain methods that can perform certain operations on strings. For example;

```
String txt = "Hello World";


System.out.println("The length of the txt string is : " + txt.length());

System.out.println("The length of the txt string is : " + txt.toUpperCase());

System.out.println("The length of the txt string is : " + txt.toLowerCase());

System.out.println("The length of the txt string is : " + txt.indexOf("World"));
```

Non-Primitive Data Types

- **Array:** Is a collection of similar type of elements which has contiguous memory location. Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value. 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. Java array inherits the Object class, and implements the Serializable as well as interfaces. We can store primitive values or objects in an array in Java. To declare an array, define the variable type with **square brackets.** Example;

String[ ] car = {"Toyota", "Benz", "Hyundai"}; **or** **String car[ ] = {"Toyota", "Benz", "Hyundai"};**

int[ ] num = {10, 20,30,40,50}; **or** **int num[ ] = {10, 20,30,40,50};**

Note: A String in Java is actually an object, which contain methods that can perform certain operations on strings. For example;

```
public class TestArray  {
        public static void main(String[ ]  args) {
                double[ ] myList =  {1.9,  2.9,  3.4,  3.5};

                for (double element:  myList )  {       // Print all  the array elements
                System.out.println(element);
                }
        }
}
```

Non-Primitive Data Types

- Interface: Is a blueprint of a class. An Interface is a completely **"abstract class"** that is used to group related methods with empty bodies. It has static constants and abstract methods. A class implements an interface, thereby inheriting the abstract methods of the interface. Another way to achieve abstraction in Java, is with interfaces. Writing an interface is similar to writing a class. But a class describes the attributes and behaviors of an object. And an interface contains behaviors that a class implements.

Note:

- You cannot instantiate an interface.
- An interface does not contain any constructors.
- All of the methods in an interface are abstract.
- An interface cannot contain instance fields. The only fields that can appear in an interface must be declared both static and final.
- An interface is not extended by a class; it is implemented by a class
- An interface can extend multiple interfaces.

WHY DO WE USE INTERFACE IN JAVA:

- It is used to achieve abstraction.
- By interface, we can support the functionality of multiple inheritance.

OPERATORS IN JAVA

Operators are used to perform operations on variables and values. Java provides a rich set of operators to manipulate variables. We can divide all the Java operators into the following groups:

TYPES OF OPERATORS

- Arithmetic operators
- Assignment operators
- Comparison/ Relational operators
- Logical operators
- Bitwise operators

ARITHMETIC OPERATORS:- Arithmetic operators are used to perform common mathematical operations. The following table lists the arithmetic operators.

Assume integer variable A holds 10 and variable B holds 20, then;

Operators	Description	Example
+ (Addition)	Adds values on either side of the operator	A + B will give 30
- (Subtraction)	Subtracts right-hand operand from left-hand operand	A + B will give 30
* (Multiplication)	Multiplies values on either side of the operator	A * B will give 200
/ (Division)	Divides left-hand operand by right-hand operand.	B / A will give 2
% (Modulus)	Divides left-hand operand by right-hand operand and returns remainder.	B / A will give 2
++ (Increment)	Increases the value of operand by 1.	B++ gives 21
-- (Decrement)	Decreases the value of operand by 1.	B-- gives 19

ASSIGNMENT OPERATORS:- Assignment operators are used to assign values to variables.

Operators	Description	Example
=	Simple assignment operator. Assigns values from right side operands to left side operand	C = A + B will assign value of A + B into C
+=	Add AND assignment operator. It adds right operand to the left operand and assign the result to left operand	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator. It subtracts right operand from the left operand and assign the result to left operand	C -= A is equivalent to C = C – A
*=	Multiply AND assignment operator. It multiplies right operand with the left operand and assign the result to left operand	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator. It divides left operand with the right operand and assign the result to left operand	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator. It takes modulus using two operands and assign the result to left operand	C %= A is equivalent to C = C % A
<<=	Left shift AND assignment operator	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator	C &= 2 is same as C = C & 2
^=	bitwise exclusive OR and assignment operator	C ^= 2 is same as C = C ^ 2
|=	bitwise inclusive OR and assignment operator	C |= 2 is same as C = C | 2

COMPARISON / RELATIONAL OPERATORS:- Comparison operators are used to compare two values.

Assume variable A holds 10 and variable B holds 20, then;

Operators	Name	Description	Example
==	Equal to	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true
!=	Not equal to	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true	(A != B) is true.
>	Greater than	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true	(A > B) is not true
<	Less than	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true	(A < B) is true.
>=	Greater than or equal to	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true	(A >= B) is not true
<=	Less than or equal to	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true

LOGICAL OPERATORS:- Logical operators are used to determine the logic between variables or values.

Assume Boolean variables A holds true and variable B holds false, then;

Operators	Name	Description	Example
&&	And	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.	(A && B) is false
||	Or	Called Logical OR Operator. If any of the two operands are non-zero, then the condition becomes true	(A || B) is true
!	Not	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false	!(A && B) is true

BITWISE OPERATORS:-Bitwise operator works on bits and performs bit-by-bit operation.

Assume integer variable A holds 60 and variable B holds 13, then;

Operators	Name	Description	Example
&	And	Binary And Operator copies a bit to the result if it exists in both operands	(A & B) will give 12 which is 0000 1100
|	Or	Binary Or Operator copies a bit if it exists in either operand.	(A | B) will give 61 which is 0011 1101
^	Xor	Binary Xor Operator copies the bit if it is set in one operand but not both.	(A ^ B) will give 49 which is 0011 0001
~	Compliment	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits	(~A) will give -61 which is 1100 0011 in 2's complement form
<<	Left shift	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand	A << 2 will give 240 which is 1111 0000
>>	Right shift	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand	A >> 2 will give 15 which is 1111
>>>	Zero fill right shift	Shift right zero fill operator. The left operands value is moved right by the number of bits specified by the right operand and shifted values are filled up with zeros	A >>>2 will give 15 which is 0000 1111

VARIABLES IN JAVA

Variables are "containers" used to store data values. A variable provides us with named storage that our programs can manipulate. Each variable in Java has a specific type, which determines the size and layout of the variable's memory; the range of values that can be stored within that memory; and the set of operations that can be applied to the variable.

In Java, there are different types of variables, for example;

- **String:** stores text, such as "Hello". String values are surrounded by double quotes. Example;

  String text = “Hello World”;

- **Int:** stores integers (whole numbers), without decimals, such as 123 or -123.

  int num = 100;

- **Float:** stores floating point numbers, with decimals, such as 19.99 or -19.99.

  float num = 13.59;

- **Char:** stores single characters, such as 'a' or 'B'. Char values are surrounded by single quotes.

  char a = ‘a’;

- **Boolean:** stores values with two states: true or false.

  boolean isMe = True;

THREE KINDS OF VARIABLES IN JAVA

1. Local Variables

- Local variables are declared in methods, constructors, or blocks.
- Local variables are created when the method, constructor or block is entered and the variable will be destroyed once it exits the method, constructor, or block.
- Access modifiers cannot be used for local variables.
- Local variables are visible only within the declared method, constructor, or block.
- Local variables are implemented at stack level internally.
- There is no default value for local variables, so local variables should be declared and an initial value should be assigned before the first use. Example;

```
public class Test{
        public void myAge() {
        int num = 30;
        age = age + 7;
                System.out.println("My age is : " + age);
        }
        public static void main(String[ ]  args) {
                Test test = new Test();
                test.myAge();
        }
}
```

THREE KINDS OF VARIABLES IN JAVA

2. Instance Variables

- Instance variables are declared in a class, but outside a method, constructor or any block.
- When a space is allocated for an object in the heap, a slot for each instance variable value is created.
- Instance variables are created when an object is created with the use of the keyword 'new' and destroyed when the object is destroyed.
- Instance variables hold values that must be referenced by more than one method, constructor or block, or essential parts of an object's state that must be present throughout the class.
- Instance variables can be declared in class level before or after use.
- Access modifiers can be given for instance variables.
- The instance variables are visible for all methods, constructors and block in the class. Normally, it is recommended to make these variables private (access level). However, visibility for subclasses can be given for these variables with the use of access modifiers.
- Instance variables have default values. For numbers, the default value is 0, for Booleans it is false, and for object references it is null. Values can be assigned during the declaration or within the constructor.
- Instance variables can be accessed directly by calling the variable name inside the class. However, within static methods (when instance variables are given accessibility.

Example of Instance Variables

```
public class Employee {

        // this instance variable is visible for any child class.
        public String name;

        // salary variable is visible in Employee class only.
        private double salary;

        // The name variable is assigned in the constructor.
        public Employee (String empName) {
                name = empName;
        }

        // The salary variable is assigned a value.
        public void setSalary(double empSal) {
                salary = empSal;
        }

        // This method prints the employee details.
        public void printEmp() {

        System.out.println("name : " + name );
        System.out.println("salary :" + salary);

        }

        public static void main(String args[]) {
        Employee empOne = new Employee("Ransika");
        empOne.setSalary(1000);
        empOne.printEmp();
        }
}
```

THREE KINDS OF VARIABLES IN JAVA

3. **Class / Static Variables**

- Class variables also known as static variables are declared with the static keyword in a class, but outside a method, constructor or a block.
- There would only be one copy of each class variable per class, regardless of how many objects are created from it.
- Static variables are rarely used other than being declared as constants. Constants are variables that are declared as public/private, final, and static. Constant variables never change from their initial value.
- Static variables are stored in the static memory. It is rare to use static variables other than declared final and used as either public or private constants.
- Static variables are created when the program starts and destroyed when the program stops.
- Visibility is similar to instance variables. However, most static variables are declared public since they must be available for users of the class.
- Default values are same as instance variables. For numbers, the default value is 0; for Booleans, it is false; and for object references, it is null. Values can be assigned during the declaration or within the constructor.
- Static variables can be accessed by calling with the class name *ClassName.VariableName*.
- When declaring class variables as public static final, then variable names (constants) are all in upper case. If the static variables are not public and final, the naming syntax is the same as instance and local variables.

Example of Class / Static Variables

```
import java.io*;

public class Employee {


        // Salary variable is a private static variable.
        private static double salary;


        // DEPARTMENT is constant.
        public static final String DEPARTMENT = "Development";


        public static void main(String args[]) {

                salary = 1000;

                System.out.println(DEPARTMENT + "average salary : " + salary);

        }

}
```

CONDITIONAL STATEMENTS

The Java *if statement* is used to test the condition. It checks **Boolean** condition: ***true*** or ***false***. Example, If user is male, then show tis message. If user is a female, then show this message.

In Java, we have the following conditional statements:

1. **The if Statement** : Is use to specify a block of code to be executed, if a specified condition is true.
2. **The else Statement** : Is use to specify a block of code to be executed, if the same condition is false.
3. **The else if Statement** : Is use to specify a new condition to test, if the first condition is false.
4. **The switch Statement** : Is use to specify many alternative blocks of code to be executed.

IF STATEMENT:- The **if** statement executes the code if a specified condition is true. Example;

```
int x = 20;
int y =  18;

    If ( x > y ) {
        System.out.println(" x is greater than y ");
    }
```

Note: In the example above we use two variables, **x** and **y**, to test whether x is greater than y (using the > operator). As x is 20, and y is 18, and we know that 20 is greater than 18, we print to the screen that "x is greater than y".

ELSE STATEMNT:- Use the **else** statement to specify a block of code to be executed if the condition is **false**. Example;

```
int time = 20;

    If ( time < 18 ) {
        System.out.println(" Good morning ");
    }
    else {
        System.out.println(" Good evening ");
    }
```

Note: In the example above, time (20) is greater than 18, so the condition is false. Because of this, we move on to the else condition and print to the screen "Good evening". If the time was less than 18, the program would print "Good morning".

ELSE IF STATEMENT:- Use the **else if** statement to specify a new condition if the first condition is. Example;

```
int time = 20;
        If ( time < 10 ) {
                System.out.println(" Good morning ");
        }
        else if ( time < 20 ) {
                System.out.println(" Good day ");
        }
        else {
                System.out.println(" Good evening ");
        }
```

Note: In the example above, time (22) is greater than 10, so the **first condition** is false. The next condition, in the else if statement, is also false, so we move on to the else condition since **condition1** and **condition2** is both false - and print to the screen "Good evening".
However, if the time was 14, our program would print "Good day.".

THE SWITCH STATEMENT:- Use the switch statement to select one of many code blocks to be executed. Example;

```
public class SwitchCaseExample {
    public static void main(String args[]) {
        Integer age = 18;
        switch (age) {
            case (16):
                System.out.println( "You are under 18.“ );
                break;
            case (18):
                System.out.println( "You are eligible for vote.“ );
                break;
            default:
                System.out.println( "Please give the valid age.“ );
                break;
        }
    }
}
```

LOOPS

Loops can execute a block of code as long as a specified condition is reached. A **loop** statement allows us to execute a statement or group of statements multiple times. Loops are handy because they save time, reduce errors, and they make code more readable.

These are the loop used in Java:

1. **While Loop-:** The **while** loop loops through a block of code as long as a specified condition is true. Example:

```
public class WhileLoopExample {

    public static void main(String args[])  {

        int i = 0;

        while ( i < 5 ) {
            System.out.println(i);
            i++;
        }

    }

}
```

Example explained: In the example below, the code in the loop will run, over and over again, as long as a variable (i) is less than 5.

2. **Do-While-:** This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true. Example:

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

                int i = 0;
                do {
                        System.out.println(i);
                        i++;
                }
                while ( i < 5 );

        }

}
```

2. **For Loop-:** The Java *for loop* is used to iterate a part of the program several times. If the number of iteration is **fixed**, it is recommended to use for loop. Example;

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

                for ( int i = 0; i < 5; i++ ) {
                        System.out.println(i);
                }
        }

}
```

4. **Foreach Loop-**: is used exclusively to loop through elements in an **array**. Example;

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

                String[ ] cars = { "Toyota", "Benz", "Hyundai" };

                for ( String i : cars ) {
                        System.out.println(i);
                }
        }

}
```

5. **Break Loop-:** The **break** can be use to jump out of a loop. This example jumps out of the loop when **i** is equal to **4**. Example;

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

        for ( int i = 0;  i < 10;  i++ ) {
                if ( i == 4 ) {
                        break;
                }
                System.out.println(i);
        }
}

}
```

6. **Continue Loop-:** The **continue** statement breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop. This example skips the value of **4** :

5.
```
public class ContinueLoopExample {
        public static void main(String args[])  {

        for ( int i = 0;  i < 10;  i++ ) {
            if ( i == 4 ) {
                continue;
            }
            System.out.println(i);
        }
    }

}
```

ARRAYS

An Array is an object which contains elements of a similar data type. Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value. To declare an array, define the variable type with **square brackets.** To use an array in a program, you must declare a variable to reference the array, and you must specify the type of array the variable can reference. Here is the syntax for declaring an array variable;

```
double[ ] myList;    // preferred way

double myList[ ];    // works but not preferred way
```

Here is a complete example of an Arrays:

```
public class TestArray {
        public static void main(String[] args) {
                double[] myList = {1.9, 2.9, 3.4, 3.5};

                // Print all the array elements
                for (int i = 0; i < myList.length; i++) {

                        System.out.println(myList[i] + " ");
                }

                // Summing all elements
                double total = 0;

                for (int i = 0; i < myList.length; i++) {
                        total += myList[i];
                }
                        System.out.println("Total is " + total);

                // Finding the largest element
                double max = myList[0];

                for (int i = 1; i < myList.length; i++) {

                        if (myList[i] > max) max = myList[i];
                }

                        System.out.println("Max is " + max);


        }
}
```

METHODS

A **method** is a block of code which only runs when it is called. A **method** is a block of code or collection of statements or a set of code grouped together to perform a certain task or operation and also known as Functions. It is used to achieve the **reusability** of code. We write a method once and use it many times. We do not require to write code again and again. It also provides the **easy modification** and **readability** of code, just by adding or removing a chunk of code. The method is executed only when we call or invoke it. A method must be declared within a class. It is defined with the name of the method, followed by parentheses **().** The most important method in Java is the **main()** method. Example;

```
Public class Main {
        public static void myMethod() {
                System.out.println( "This is a method" );
        }

        public static void main(String args[])  {
                myMethod();
        }
}
```

Here is a complete example of Java Method:

```
public class Main {

    // Create a checkAge() method with an integer variable called age;
     public static void checkAge( int age )

    // Create a conditional statement if age is less than 18, print "Access denied"
    if ( age < 18 ) {

        System.out.println( " Access denied – You are not old enough " );


    // if age is greater than or aqual to 18, print "Access granted"
    } else {

        System.out.println( "Access granted – You are old enough! " );

        }

    public static void main(String args[]) {

        checkAge(20);       // Call the checkAge method and pass along an age of 20

     }

}
```

Output: "Access granted – You are old enough!"

INTERFACE IN JAVA

An **interface** is a blueprint of a class. It has static constants and abstract methods. An **Interface** is a completely "**abstract class**" that is used to group related methods with empty bodies. To access the interface methods, the interface must be "implemented" (inherited) by another class with the **implements** keyword (instead of **extends**). The body of the interface method is provided by the "implement" class. The interface in Java is a mechanism to achieve abstraction. There can be only abstract methods in the Java interface, not method body. It is used to achieve abstraction and multiple inheritance in Java.

Why use Java interface?

- It is used to achieve abstraction.
- By interface, we can support the functionality of multiple inheritance.

```
public interface  Animal{

                public void animalSound();

                public void sleep();        // Interface methods does not have body
}
```

Note: Class extends Class
Class implements interface
interface extends interface

INTERFACE IN JAVA

Example;

```
interface  Animal{
        public void animalSound();
        public void sleep();        // Interface methods does not have body
}
class Pig implements Animal{
        public void animalSound()  {
                System.out.println("The pig says : wee wee ");
        }

        public void sleep()  {
                System.out.println("Zzz");
        }
}


public class Main  {
    public static void main(String[ ]  args) {
            Pig myPig = new Pig();
            myPig.animalSound();
            myPig.sleep();
    }
}
```

JAVA STRING

A String variable contains a collection of characters surrounded by double quotes. Strings, which are widely used in Java programming, are a sequence of characters. In Java programming language, strings are treated as objects. Example;

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

                char[ ] helloArray = { 'h', 'e', 'l', 'l', 'o' };

                String helloString = new String( helloArray )
                        System.out.println(helloString);
                }
        }
}
```

String Length: A String in Java is actually an object, which contain methods that can perform certain operations on strings. For example, the length of a string can be found with the **length()**.

```
public class StringExample {
        public static void main(String args[]) {
                String palindrome = "Hello World" ;
                int len = palindrome.length();
                        System.out.println( "String lenght is : " + len );
                }
        }
}
```

JAVA EXCEPTIONS (Try ___Catch)

An **exception** (or exceptional event) is a problem that arises during the execution of a program. When an **Exception** occurs the normal flow of the program is disrupted and the program/Application terminates abnormally, which is not recommended, therefore, these exceptions are to be handled. When executing Java code, different errors can occur: coding errors made by the programmer, errors due to wrong input, or other unforeseeable things. When an error occurs, Java will normally stop and generate an error message. The technical term for this is: Java will throw an **exception** (throw an error). Example;

Java try and catch:

- **The try** statement allows you to define a block of code to be tested for errors while it is being executed.
- **The catch** statement allows you to define a block of code to be executed, if an error occurs in the try block.
- **The final** block is used to execute the necessary code of the program. It is executed whether an exception is handled or not.
- **The throws** keyword is used to declare exceptions. It specifies that there may occur an exception in the method. It doesn't throw an exception. It is always used with method signature.

Example of Try, Catch and Final Exception Handling

```
import java.io.*;
        public class ReadData_Demo {
                public static void main(String args[]) {
                        FileReader fr = null;

                try {
                        File file = new File("file.txt");

                        fr = new FileReader(file);
                        char [] a = new char[50];
                        fr.read(a);

                        for(char c : a) System.out.print(c); // reads the content to the array

                } catch (IOException e) {
                                e.printStackTrace();
                        }finally   {
                                try   {
                                        fr.close();

                                } catch (IOException ex) {

                                        ex.printStackTrace();
                                }
                        }
                }
        }
```

JSP

Java Server Pages (**JSP**) is a server-side programming technology that enables the creation of dynamic, platform-independent method for building Web-based applications. **JSP** technology is used to create web application just like Servlet technology. **JSP** have access to the entire family of Java APIs, including the JDBC API to access enterprise databases. It can be thought of as an extension to Servlet because it provides more functionality than servlet such as expression language, JSTL(JSP Standard Tag Library). A JSP page consists of HTML tags and JSP tags. The **JSP** pages are easier to maintain than Servlet because we can separate designing and development. It provides some additional features such as Expression Language, Custom Tags etc.

ADVANTAGES OF JSP

1. **Extension to Servlet:** JSP technology is the extension to Servlet technology. We can use all the features of the Servlet in JSP. In addition to, we can use implicit objects, predefined tags, expression language and Custom tags in JSP, that makes JSP development easy.
2. **Easy to maintain:** JSP can be easily managed because we can easily separate our business logic with presentation logic. In Servlet technology, we mix our business logic with the presentation logic.
3. **Fast Development:** No need to recompile and redeploy. If JSP page is modified, we don't need to recompile and redeploy the project. The Servlet code needs to be updated and recompiled if we have to change the look and feel of the application.
4. **Less code than Servlet:** In JSP, we can use many tags such as action tags, JSTL, custom tags, etc. that reduces the code. Moreover, we can use EL, implicit objects.

JSP SCRIPTING ELEMENTS

The scripting elements provides the ability to insert java code inside the jsp. There are three types of scripting elements:

1. **Scriptlet Tag:** A scriptlet tag is used to execute java source code in JSP. Syntax is as follows;

   <% out.println("welcome to jsp"); %>

2. **Expression:** A JSP expression element contains a scripting language expression that is evaluated, converted to a String, and inserted where the expression appears in the JSP file. It is mainly used to print the values of variable or method. Example;

   <%= (new java.util.Date()).getTime() %>

3. **Declaration:** A declaration declares one or more variables or methods that you can use in Java code later in the JSP file. You must declare the variable or method before you use it in the JSP file. The **JSP declaration tag** is used *to declare fields and methods.*

   <%! int num = 50; %>

JSP SCRIPTING ELEMENTS

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1" pageEncoding="ISO-8859-1"%>
<%@ page import = "java.util.Date" %>


<!DOCTYPE html>
    <html>
        <body>

            <%  out.println( “welcome to jsp” );  %>


            <%=  (new java.util.Date()).getTime()  %>


        </body>
    </html>
```

JDBC (JAVA DATABASE CONNECTIVITY)

JDBC is a Java API to connect and execute the query with the database. JDBC is a Java API to connect and execute the query with the database. By the help of JDBC API, we can save, update, delete and fetch data from the database.

WHY JDBC?

1. Connect to the database.
2. Creating SQL or MySQL statements.
3. Execute queries and update statements to the database.
4. Retrieve the result received from the database.

The JDBC API provides the following interfaces and classes:

- **DriverManager:** This class manages a list of database drivers. Matches connection requests from the java application with the proper database driver using communication sub protocol. The first driver that recognizes a certain subprotocol under JDBC will be used to establish a database Connection.
- **Driver**: This interface handles the communications with the database server. You will interact directly with Driver objects very rarely. Instead, you use DriverManager objects, which manages objects of this type. It also abstracts the details associated with working with Driver objects.
- **Connection**: This interface with all methods for contacting a database. The connection object represents communication context, i.e., all communication with database is through connection object only.

JDBC (JAVA DATABASE CONNECTIVITY)

The JDBC API provides the following interfaces and classes:

- **Statement**: You use objects created from this interface to submit the SQL statements to the database. Some derived interfaces accept parameters in addition to executing stored procedures.
- **ResultSet:** These objects hold data retrieved from a database after you execute an SQL query using Statement objects. It acts as an iterator to allow you to move through its data.
- **SQLException**: This class handles any errors that occur in a database application.

The following steps are required to create a new Database using JDBC application:

- **Import the packages:** Requires that you include the packages containing the JDBC classes needed for database programming. ***import java.sql.****
- **Open a connection:** Requires using the ***DriverManager.getConnection()*** method to create a Connection object, which represents a physical connection with the database server.
- **Execute a query:** Requires using an object of type Statement for building and submitting an SQL statement to the database.
- **Clean up the environment:** try with resources automatically closes the resources.

JDBC (JAVA DATABASE CONNECTIVITY)

How to create a Database with Java JDBC:

```
import java.sql.*;
public class JDBCExample{

        static final String DB_URL = "jdbc:mysql://localhost/";
        static final String USER = "root";
        static final String PASS = "test";

        public static void main(String args[]) {

            try(Connection conn = DriverManager.getConnection(DB_URL, USER, PASS);
                Statement stmt = conn.createStatement();

            ) {
                String sql = "Create Database Students";
                stmt.executeUpdate(sql);
                System.out.println("Database created successfully……");

            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
}
```

JDBC (JAVA DATABASE CONNECTIVITY)

How to create a Table with Java JDBC:

```
import java.sql.*;
public class JDBCExample{

        static final String DB_URL = "jdbc:mysql://localhost/Students";
        static final String USER = "root";
        static final String PASS = "test";

        public static void main(String args[]) {
            try(Connection conn = DriverManager.getConnection(DB_URL, USER, PASS);
                Statement stmt = conn.createStatement();


              ) {
                String sql = "Create Table Registration " +
            "(id Integer not null, " + "name Varchar(255), " + "age Integer, " + "PRIMARY KEY(id))";
                        stmt.executeUpdate(sql);
                        System.out.println("Table created successfully……");

                } catch (SQLException e) {
                        e.printStackTrace();
                }
        }
}
```

JDBC (JAVA DATABASE CONNECTIVITY)

How to insert record into Table with Java JDBC:

```
import java.sql.*;
public class JDBCExample{

        static final String DB_URL = "jdbc:mysql://localhost/Students";
        static final String USER = "root";
        static final String PASS = "test";

        public static void main(String args[]) {
            try(Connection conn = DriverManager.getConnection(DB_URL, USER, PASS);
                Statement stmt = conn.createStatement();


              ) {

                String sql = "insert into Registration Values (100,  'John Doe',  20) ";
                    stmt.executeUpdate(sql);
                    System.out.println("Record inserted successfully……");

                } catch (SQLException e)  {
                    e.printStackTrace();
                }
        }
}
```

JDBC (JAVA DATABASE CONNECTIVITY)

How to select a record from a Table with Java JDBC:

```
import java.sql.*;
public class JDBCExample{
        static final String DB_URL = "jdbc:mysql://localhost/Students";
        static final String USER = "root";
        static final String PASS = "test";
        static final String QUERY = "Select id, name, age from Registration";

        public static void main(String args[]) {
            try(Connection conn = DriverManager.getConnection(DB_URL, USER, PASS);
                Statement stmt = conn.createStatement();
                Resultset rs = stmt.executeQuery(QUERY);
              ) {
                while(rs.next()) {
                    System.out.println("Id : + rs.getInt("id"));
                     System.out.println("Name: + rs.getString("name"));
                     System.out.println("Age : + rs.getInt("age"));

                } catch (SQLException e) {
                    e.printStackTrace();
                }
        }
}
```

JDBC (JAVA DATABASE CONNECTIVITY)

How to update a record in a Table with Java JDBC:

```
import java.sql.*;
public class JDBCExample{
        static final String DB_URL = "jdbc:mysql://localhost/Students";
        static final String USER = "root";
        static final String PASS = "test";
        static final String QUERY = "Select id, name, age from Registration";

        public static void main(String args[]) {
            try(Connection conn = DriverManager.getConnection(DB_URL, USER, PASS);
                Statement stmt = conn.createStatement();
               ) {
                String sql = "Update Registartion Set age = 30 where id = 100";
                stmt.executeUpdate(sql);
                Resultset rs = stmt.executeQuery(QUERY);
                while(rs.next()) {
                    System.out.println("Id : + rs.getInt("id"));
                     System.out.println("Name: + rs.getString("name"));
                     System.out.println("Age : + rs.getInt("age"));
                } catch (SQLException e) {
                    e.printStackTrace();
                }
        }
}
```

JDBC (JAVA DATABASE CONNECTIVITY)

How to delete a record in a Table with Java JDBC:

```
import java.sql.*;
public class JDBCExample{
        static final String DB_URL = "jdbc:mysql://localhost/Students";
        static final String USER = "root";
        static final String PASS = "test";
        static final String QUERY = "Select id, name, age from Registration";

        public static void main(String args[]) {
            try(Connection conn = DriverManager.getConnection(DB_URL, USER, PASS);
                Statement stmt = conn.createStatement();
               ) {
                String sql = "Delete from Registartion where id = 100";
                stmt.executeUpdate(sql);
                Resultset rs = stmt.executeQuery(QUERY);
                while(rs.next()) {
                    System.out.println("Id : + rs.getInt("id"));
                     System.out.println("Name: + rs.getString("name"));
                     System.out.println("Age : + rs.getInt("age"));
                } catch (SQLException e) {
                    e.printStackTrace();
                }
        }
    }
```

JDBC (JAVA DATABASE CONNECTIVITY)

How to fetch record in ascending order from a Table with Java JDBC:

```
import java.sql.*;
public class JDBCExample{
        static final String DB_URL = "jdbc:mysql://localhost/Students";
        static final String USER = "root";
        static final String PASS = "test";
        static final String QUERY = "Select id, name, age from Registration order by name ASC";

        public static void main(String args[]) {
            try(Connection conn = DriverManager.getConnection(DB_URL, USER, PASS);
                Statement stmt = conn.createStatement();
              ) {
                Resultset rs = stmt.executeQuery(QUERY);
                while(rs.next()) {
                    System.out.println("Id : + rs.getInt("id"));
                     System.out.println("Name: + rs.getString("name"));
                     System.out.println("Age : + rs.getInt("age"));

                } catch (SQLException e) {
                    e.printStackTrace();
                }
        }
}
```

SPRING FRAMEWORK

Spring boot is a module of spring framework which is used to create stand-alone, production-grade Spring based Applications with minimum programmer's efforts. It provides an easier and faster way to set up, configure, and run both simple and web-based applications

Advantages of Spring Boot:

1. It creates **stand-alone** Spring applications that can be started using Java **-jar**.
2. It tests web applications easily with the help of different **Embedded** HTTP servers such as **Tomcat, Jetty,** etc. We don't need to deploy WAR files.
3. It provides opinionated '**starter**' POMs to simplify our Maven configuration.
4. It provides **production-ready** features such as **metrics, health checks,** and **externalized configuration**.
5. There is no requirement for **XML** configuration.
6. It offers a **CLI** tool for developing and testing the Spring Boot application.
7. It offers the number of **plug-ins**.
8. It also minimizes writing multiple **boilerplate codes** (the code that has to be included in many places with little or no alteration), XML configuration, and annotations.
9. It **increases productivity** and reduces development time.

SPRING BOOT FRAMEWORK

```
package com.tutorialspoint.demo;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@SpringBootApplication
@RestController
public class DemoApplication {

        public static void main(String[] args) {
                SpringApplication.run(DemoApplication.class, args);

}
        @RequestMapping(value = "/")
                public String hello() {

                        return "Hello World";
                }
        }
```