Unit-1 Introduction to Java

- History of Java
-Features of Java, Applications of Java, Java Virtual Machine (JVM) and Byte Code, Buzz Words.
- Basics Concept of OOP: Abstraction and Encapsulation, Inheritance and Polymorphism
- Comparison Between C++ and Java.
- Data types, Operators.
- Control Statement, Array, and command line argument.

What is Java?

- Java is a high-level, Robust, general-purpose, object-oriented, and secure programming language.
- JAVA developed by James Gosling at Sun Microsystems, Inc. in 1991.
- It is formally known as OAK.
- In 1995, Sun Microsystem changed the name to Java.
- In 2009, Sun Microsystem takeover by Oracle Corporation.
- Java is a class-based object-oriented programming language that implements the principle of write once code anywhere.

History of java

- Java was originally designed for interactive television, but it was too advanced technology for the digital cable television industry at the time. The history of Java starts with the Green Team. Java team members (also known as Green Team), initiated this project to develop a language for digital devices such as set-top boxes, televisions, etc. However, it was best suited for internet programming. Later, Java technology was incorporated by Netscape.

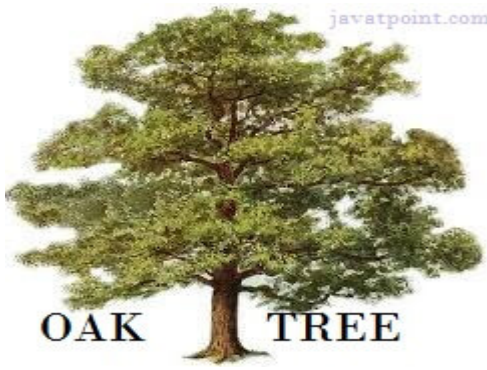
- The principles for creating Java programming were "Simple, Robust, Portable, Platform-independent, Secured, High Performance, Multithreaded, Architecture Neutral, Object-Oriented, Interpreted, and Dynamic". Java was developed by James Gosling, who is known

as the father of Java, in 1995. James Gosling and his team members started the project in the early '90s.

Currently, Java is used in internet programming, mobile devices, games, e-business solutions, etc. Following are given significant points that describe the history of Java.

1) **James Gosling, Mike Sheridan**, and **Patrick Naughton** initiated the Java language project in June 1991. The small team of sun engineers called **Green Team**.

2) Initially it was designed for small, embedded systems in electronic appliances like set-top boxes.
3) Firstly, it was called **"Greentalk"** by James Gosling, and the file extension was .gt.
4) After that, it was called **Oak** and was developed as a part of the Green project.



5)Why Oak? Oak is a symbol of strength and chosen as a national tree of many countries like the U.S.A., France, Germany, Romania, etc.

6) In 1995, Oak was renamed as **"Java"** because it was already a trademark by Oak Technologies.

7) Why had they chose the name Java for Java language? The team gathered to choose a new name. The suggested words were "dynamic", "revolutionary", "Silk", "jolt", "DNA", etc. They wanted something that reflected the essence of the technology: revolutionary, dynamic, lively, cool, unique, and easy to spell, and fun to say.
According to James Gosling, "Java was one of the top choices along with **Silk**". Since Java was so unique, most of the team members preferred Java than other names.

8) Java is an island in Indonesia where the first coffee was produced (called Java coffee). It is a kind of espresso bean. Java name was chosen by James Gosling while having a cup of coffee nearby his office.

9) Notice that Java is just a name, not an acronym.

10) Initially developed by James Gosling at Sun Microsystems (which is now a subsidiary of Oracle Corporation) and released in 1995.

11) In 1995, Time magazine called **Java one of the Ten Best Products of 1995**.

12) JDK 1.0 was released on January 23, 1996. After the first release of Java, there have been many additional features added to the language. Now Java is being used in Windows applications, Web applications, enterprise applications, mobile applications, cards, etc. Each new version adds new features in Java.

- Java is now one of the most popular programming languages in the world. It is used to develop a wide variety of applications, including web applications, enterprise applications, and mobile applications. Java is also a popular choice for teaching programming because it is relatively easy to learn and use.

Editions of Java

- Each edition of Java has different capabilities. There are three editions of Java:
    1. **Java to Standard Editions (J2SE):** It is used to create programs for a desktop computer.
    2. **Java to Enterprise Edition (J2EE):** It is used to create large programs that run on the server and manages heavy traffic and complex transactions.
    3. **Java to Micro Edition (J2ME):** It is used to develop applications for small devices such as set-top boxes, phone, and appliances.

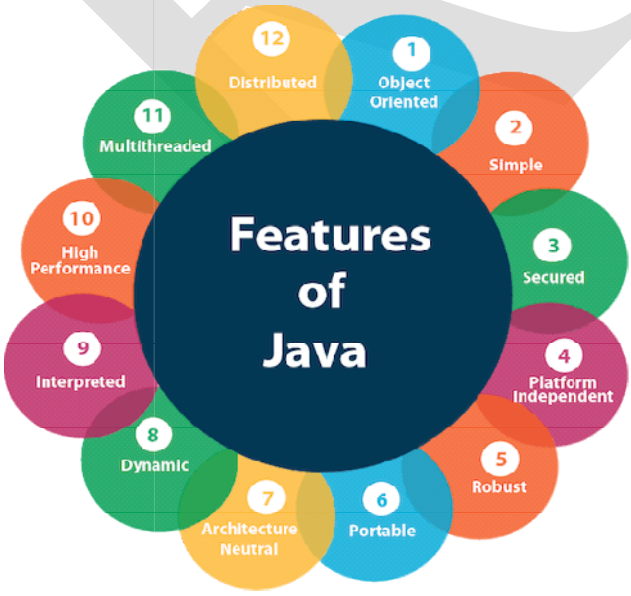
Types of Java Applications

There are four types of Java applications that can be created using Java programming:

1. **Standalone Applications:** Java standalone applications uses GUI components such as AWT, Swing, and JavaFX. These components contain buttons, list, menu, scroll panel, etc. It is also known as desktop alienations.
2. **Enterprise Applications:** An application which is distributed in nature is called enterprise applications.
3. **Web Applications:** An applications that run on the server is called web applications. We use JSP, Servlet, Spring, and Hibernate technologies for creating web applications.
4. **Mobile Applications:** Java ME is a cross-platform to develop mobile applications which run across smartphones. Java is a platform for App Development in Android.

Features of Java /Buzz Words of Java

A list of the most important features of the Java language is given below.



1. Object Oriented

- In java, everything is an object which has some data and behaviour. Java can be easily extended as it is based on Object Model. Following are some basic concept of OOP's.
  i. Object
  ii. Class
  iii. Inheritance
  iv. Polymorphism
  v. Abstraction
  vi. Encapsulation

2. Simple

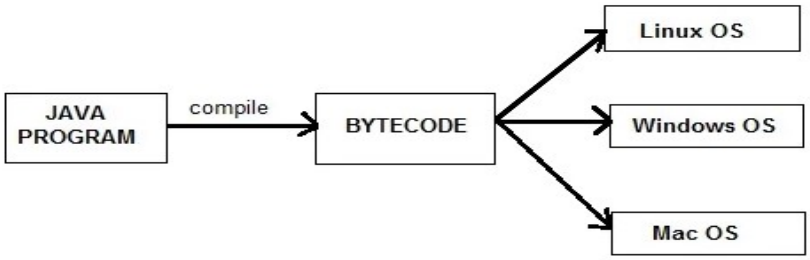
- Java is very easy to learn, and its syntax is simple, clean and easy to understand. According to Sun Microsystem, Java language is a simple programming language

3. Secure

- Security is one of the biggest concern in programming language as these programming langauges are used to develop some of the critical and sensitive applications that needs to be secured such as banking applications. Java is more secure than C/C++ as does not allow developers to create pointers, thus it becomes impossible to access a variable from outside if it's not been initialized.
- We don't have pointers and we cannot access out of bound arrays (you get ArrayIndexOutOfBoundsException if you try to do so) in java. That's why several security flaws like stack corruption or buffer overflow is impossible to exploit in Java.

4. Platform Independent

- Unlike other programming languages such as C, C++ etc which are compiled into platform specific machines. Java is guaranteed to be write-once, run-anywhere language.
- On compilation Java program is compiled into bytecode. This bytecode is platform independent and can be run on any machine, plus this bytecode format also provide security. Any machine with Java Runtime Environment can run Java Programs.



5.Robust

Robust means strong. Java is robust because:

- It uses strong memory management.
- There is a lack of pointers that avoids security problems.
- Java provides automatic garbage collection which runs on the Java Virtual Machine to get rid of objects which are not being used by a Java application anymore.
- There are exception handling and the type checking mechanism in Java. All these points make Java robust.

6.Portable

- Java is portable because it facilitates you to carry the Java bytecode to any platform. It doesn't require any implementation.

7. Architectural Neutral

- Compiler generates bytecodes, which have nothing to do with a particular computer architecture, hence a Java program is easy to intrepret on any machine.

8. Dynamic

- Java is a dynamic language. It supports the dynamic loading of classes. It means classes are loaded on demand. It also supports functions from its native languages, i.e., C and C++.

9. Interpreted

- Java is considered an interpreted language.
- As Java is neither fully interpreted nor compiled, it is a combination of both.
- First, the source code gets compiled into a .class file i.e. bytecode to get understood by JRE.
- After that, bytecode is interpreted by the JVM making it an interpreted language.

10) High Performance

- Java is an interpreted language, so it will never be as fast as a compiled language like C or C++. But, Java enables high performance with the use of just-in-time compiler.

11)Multithreading

- Java supports multithreading. Multithreading is a Java feature that allows concurrent execution of two or more parts of a program for maximum utilisation of CPU.

12) Distributed

- Java is also a distributed language. Programs can be designed to run on computer networks.
- Java has a special class library for communicating using TCP/IP protocols. Creating network connections is very much easy in Java as compared to C/C++.

JVM

- JVM Stands for Java Virtual Machine.
- JVM is the engine that drives the Java code.
- Mostly in other Programming Languages, compiler produce code for a particular system but Java compiler produce Bytecode for a Java Virtual Machine.
- When we compile a Java program, then bytecode is generated. Bytecode is the source code that can be used to run on any platform.
- Bytecode is an intermediary language between Java source and the host system.
- It is the medium which compiles Java code to bytecode which gets interpreted on a different machine and hence it makes it Platform/Operating system independent.

.

JVM's work can be explained in the following manner

- Reading Bytecode.
- Verifying bytecode.
- Linking the code with the library.

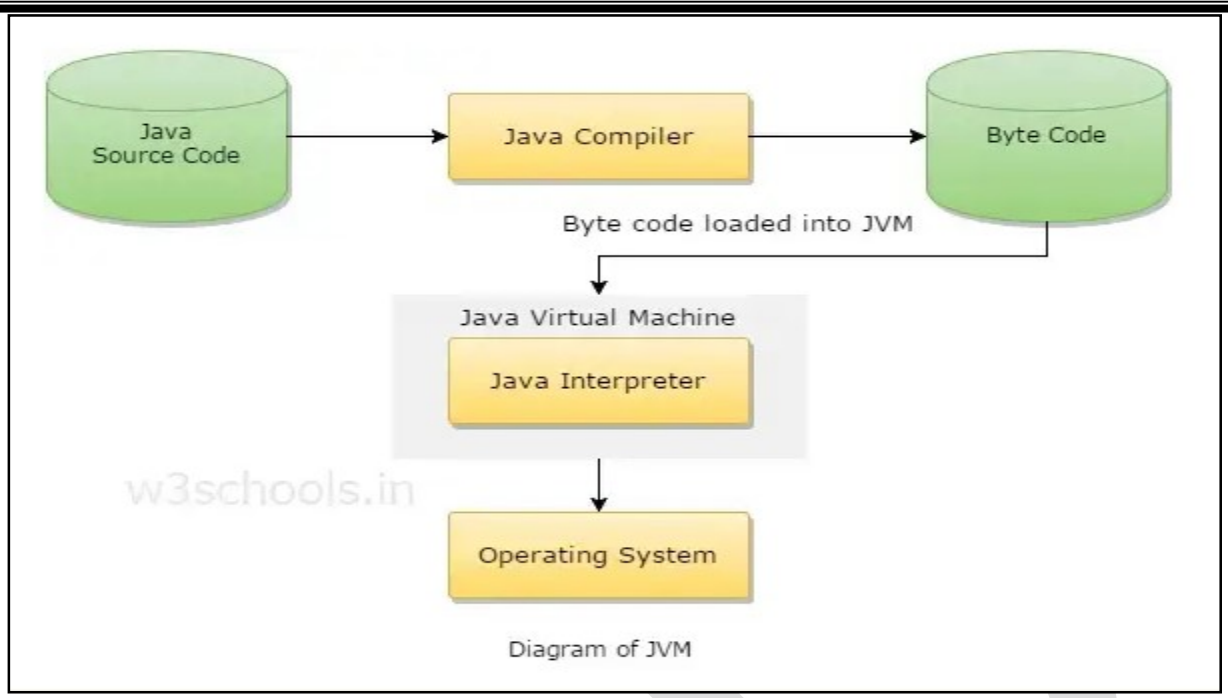


Diagram of JVM

- JVM generates a .class(Bytecode) file, and that file can be run in any OS, but JVM should have in OS because JVM is platform dependent.
- Java is called platform independent because of Java Virtual Machine. As different computers with the different operating system have their JVM, when we submit a .class file to any operating system, JVM interprets the bytecode into machine level language.
- JVM is the main component of Java architecture, and it is the part of the JRE (Java Runtime Environment).
- A program of JVM is written in C Programming Language, and JVM is Operating System dependent.
- JVM is responsible for allocating the necessary memory needed by the Java program.
- JVM is responsible for deallocating memory space.

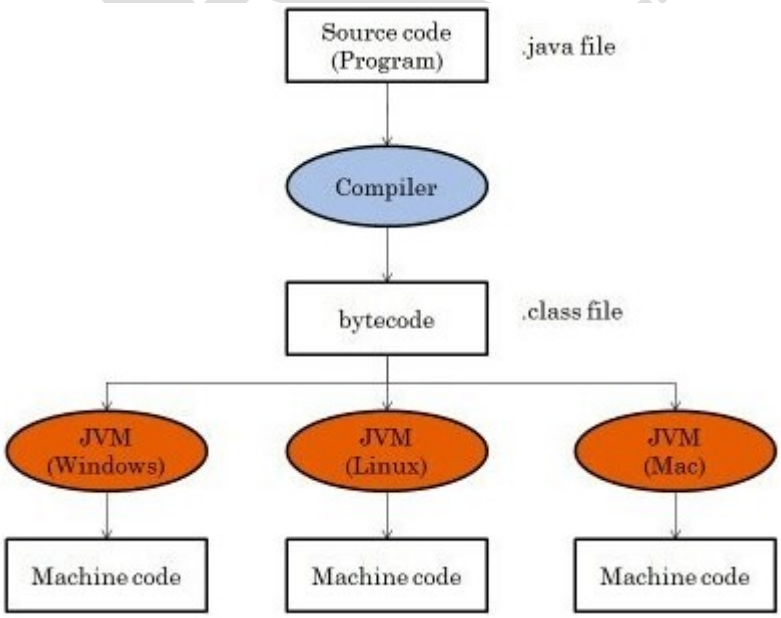
Byte Code

What is Java Bytecode?

- Java bytecode is the instruction set for the Java Virtual Machine. It acts similar to an assembler which is an alias representation of a C++ code. As soon as a java program is compiled, java bytecode is generated. In more terms, java bytecode is the machine code in the form of a .class file. With the help of java bytecode we achieve platform independence in java.

How does it works?

- When we write a program in Java, firstly, the compiler compiles that program and a bytecode is generated for that piece of code. When we wish to run this .class file on any other platform, we can do so.
- After the first compilation, the bytecode generated is now run by the Java Virtual Machine and not the processor in consideration. This essentially means that we only need to have basic java installation on any platforms that we want to run our code on.
- Resources required to run the bytecode are made available by the Java Virtual Machine, which calls the processor to allocate the required resources. JVM's are stack-based so they stack implementation to read the codes.



Advantage of Java Bytecode

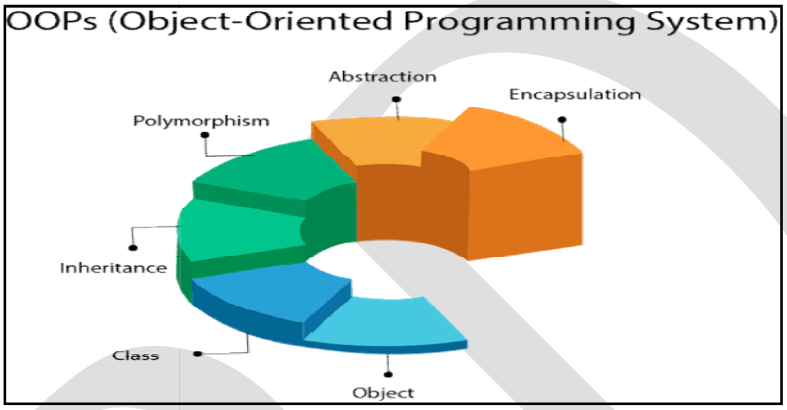
- Platform independence is one of the soul reasons for which James Gosling started the formation of java and it is this implementation of bytecode which helps us to achieve this.
- Hence bytecode is a very important component of any java program.The set of instructions for the JVM may differ from system to system but all can interpret the bytecode. A point to keep in mind is that bytecodes are non-runnable codes and rely on the availability of an interpreter to execute and thus the JVM comes into play.
- Bytecode is essentially the machine level language which runs on the Java Virtual Machine. Whenever a class is loaded, it gets a stream of bytecode per method of the class. Whenever that method is called during the execution of a program, the bytecode for that method gets invoked.
- Javac not only compiles the program but also generates the bytecode for the program. Thus, we have realized that the bytecode implementation makes Java a **platform-independent** language. This helps to add portability to Java which is lacking in languages like C or C++. Portability ensures that Java can be implemented on a wide array of platforms like desktops, mobile devices, severs and many more. Supporting this, Sun Microsystems captioned JAVA as *"write once, read anywhere" or "WORA"* in resonance to the bytecode interpretation.

Disadvantages

- The bytecode cannot run without an interpreter or JVM. If any device doesn't have JVM, bytecode won't run on that device.
- It is difficult to analyze the bytecode as it is in the form of binary and not understandable by humans.

Basic Concept of OOP: Abstraction and Encapsulation, Inheritance and Polymorphism

➢ **Object** means a real-world entity such as a pen, chair, table, computer, watch, etc. **Object-Oriented Programming** is a methodology or paradigm to design a program using classes and objects. It simplifies software development and maintenance by providing some concepts:



Object

- Any entity that has state and behavior is known as an object. For example, a chair, pen, table, keyboard, bike, etc. It can be physical or logical.
- An Object can be defined as an instance of a class. An object contains an address and takes up some space in memory. Objects can communicate without knowing the details of each other's data or code. The only necessary thing is the type of message accepted and the type of response returned by the objects.
- **Example:** A dog is an object because it has states like color, name, breed, etc. as well as behaviors like wagging the tail, barking, eating, etc.

Class

- Collection *of* objects is called class. It is a logical entity.
- A class can also be defined as a blueprint from which you can create an individual object. Class doesn't consume any space.

Inheritance

- When one object acquires all the properties and behaviors of a parent object, it is known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.
- This is another fundamental principle of object oriented programming. Inheritance is a mechanism by which object of one class is allowed to inherit the features(fields and methods) of another class. This feature basically allows us to reuse the existing code.
- The class that inherits the feature of another class is known as subclass or child class and the class whose feature is being inherited is known as super class or parent class. By inheriting the parent class the child class get's the access of fields and methods of parent class.
- To take a real world example we can consider parent child relationship. The properties of parents like hands, legs, eyes etc and the behaviors like walk, talk, eat etc are inherited in child, so child can also use/access these properties and behavior whenever needed.

  Java provides extends keyword to inherit the feature of an existing class. The basic syntax of inheriting a class is :

```
class A extends B {
  // Here Class A will inherit the features of Class B
}
```

Polymorphism

- This is another fundamental principle of object oriented programming. The word polymorphism is made from two words, Poly which means many and morphism which means forms or types, so the word polymorphism means many forms. Polymorphims in java is a mechanism in which the object or it's behavior can have many different forms.
- If *one task is performed in different ways*, it is known as polymorphism. For example: to convince the customer differently, to draw something, for example, shape, triangle, rectangle, etc.
- In Java, we use method overloading and method overriding to achieve polymorphism.
- Another example can be to speak something; for example, a cat speaks meow, dog barks woof, etc.

Abstraction

- Hiding internal details and showing functionality is known as abstraction. For example phone call, we don't know the internal processing.
- In Java, we use abstract class and interface to achieve abstraction.
- A real world example of abstraction could be your TV remote. The remote has different functions like on/off, change channel, increase/decrease volume etc. We use these functionalities just pressing the button. The internal mechanism of these functionalities are abstracted(hidden) from us as those are not essential for us to know, this is what abstraction is. Abstraction says that, focus on what an object does rather than how it does.
- In java abstraction is achieved using abstract classes and interfaces as these types contains set of abstract behaviors(method), the internal working of these behaviors are defined by some other classes.

Encapsulation

- Binding (or wrapping) code and data together into a single unit are known as encapsulation. For example, a capsule, it is wrapped with different medicines.

- A java class is the example of encapsulation. Java bean is the fully encapsulated class because all the data members are private here.

- In java every class, interface or object that we create is a form of encapsulation. Each class contains it's information(variables and methods) associated with it, other classes needs to create object of this class in order to access these information.

Comparison Between C++ and Java

There are many differences and similarities between the C++ programming language and Java. A list of top differences between C++ and Java are given below:

Index	C++	Java
Platform-independent	C++ is platform-dependent.	Java is platform-independent.
Mainly used for	C++ is mainly used for system programming.	Java is mainly used for application programming. It is widely used in Windows-based, web-based, enterprise, and mobile applications.
Design Goal	C++ was designed for systems and applications programming. It was an extension of the C programming language.	Java was designed and created as an interpreter for printing systems but later extended as a support network computing. It was designed to be easy to use and accessible to a broader audience.

Goto	C++ supports the goto statement.	Java doesn't support the goto statement.
Multiple inheritance	C++ supports multiple inheritance.	Java doesn't support multiple inheritance through class. It can be achieved by using interfaces in java.
Operator Overloading	C++ supports operator overloading.	Java doesn't support operator overloading.
Pointers	C++ supports pointers. You can write a pointer program in C++.	Java supports pointer internally. However, you can't write the pointer program in java. It means java has restricted pointer support in java.
Compiler and Interpreter	C++ uses compiler only. C++ is compiled and run using the compiler which converts source code into machine code so, C++ is platform dependent.	Java uses both compiler and interpreter. Java source code is converted into bytecode at compilation time. The interpreter executes this bytecode at runtime and produces output. Java is interpreted that is why it is platform-independent.
Call by Value and Call by reference	C++ supports both call by value and call by reference.	Java supports call by value only. There is no call by reference in java.
Structure and Union	C++ supports structures and unions.	Java doesn't support structures and unions.
Thread Support	C++ doesn't have built-in support for threads. It relies on third-party libraries for thread support.	Java has built-in thread support.
Documentation comment	C++ doesn't support documentation comments.	Java supports documentation comment (/** ... */) to create documentation for java source code.

Virtual Keyword	C++ supports virtual keyword so that we can decide whether or not to override a function.	Java has no virtual keyword. We can override all non-static methods by default. In other words, non-static methods are virtual by default.
unsigned right shift >>>	C++ doesn't support >>> operator.	Java supports unsigned right shift >>> operator that fills zero at the top for the negative numbers. For positive numbers, it works same like >> operator.
Hardware	C++ is nearer to hardware.	Java is not so interactive with hardware.

Data types

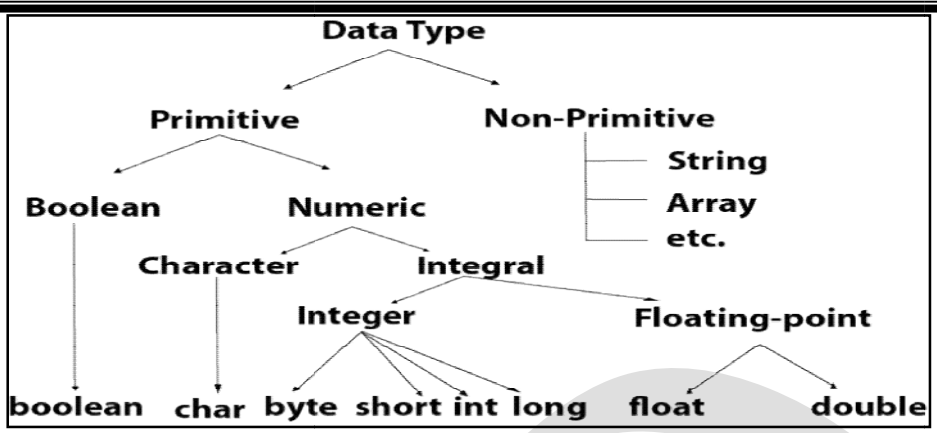
- Data types specify the different sizes and values that can be stored in the variable. There are two types of data types in Java:
  1. **Primitive data types:** The primitive data types include boolean, char, byte, short, int, long, float and double.
  2. **Non-primitive data types:** The non-primitive data types include Classes, Interfaces, and Arrays.

1. Java Primitive Data Types

- In Java language, primitive data types are the building blocks of data manipulation. These are the most basic data types available in Java language.
- Java is a statically-typed programming language. It means, all variables must be declared before its use. That is why we need to declare variable's type and name.

There 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



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.

Example 1: Java boolean data type

```
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

Example 2: Java byte data type

```
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

Example 3: Java short data type

```
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 **-231** to **231-1** (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 232-1. To learn more, visit How to use the unsigned integer in java 8?
- Default value: 0

Example 4: Java int data type

```
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 **-263** to **263-1** (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 **264-1**.
- Default value: 0

Example 5: Java long data 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.
- It should never be used for precise values such as currency.
- Default value: 0.0 (0.0d)

Example 6: Java double data type

```
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.
- It should never be used for precise values such as currency.
- Default value: 0.0 (0.0f)

Example 7: Java float data type

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

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

- Notice that we have used -42.3f instead of -42.3in 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.

- If you want to know about single-precision and double-precision, visit Java single-precision and double-precision floating-point.

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'.
- Default value: '\u0000'

Example 8: Java char data type

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

    char letter = '\u0051';
    System.out.println(letter);  // prints Q
  }
}
```

Here, the Unicode value of Q is **\u0051**. Hence, we get Q as the output.

Here is another example:

```
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

  }
}
```

- 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. To learn more about ASCII, visit What is ASCII Code?.

9.String type

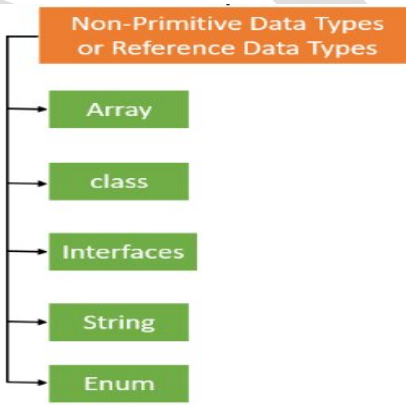
- Java also provides support for character strings via java.lang.String class. Strings in Java are not primitive types. Instead, they are objects. For example,

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

Here, `myString` is an object of the `String` class.

2.Non-Primitive Data Types

- Non-primitive data types or reference data types refer to instances or objects. They cannot store the value of a variable directly in memory. They store a memory address of the variable. Unlike primitive data types we define by Java, non-primitive data types are user-defined. Programmers create them and can be assigned with null. All non-primitive data types are of equal size.

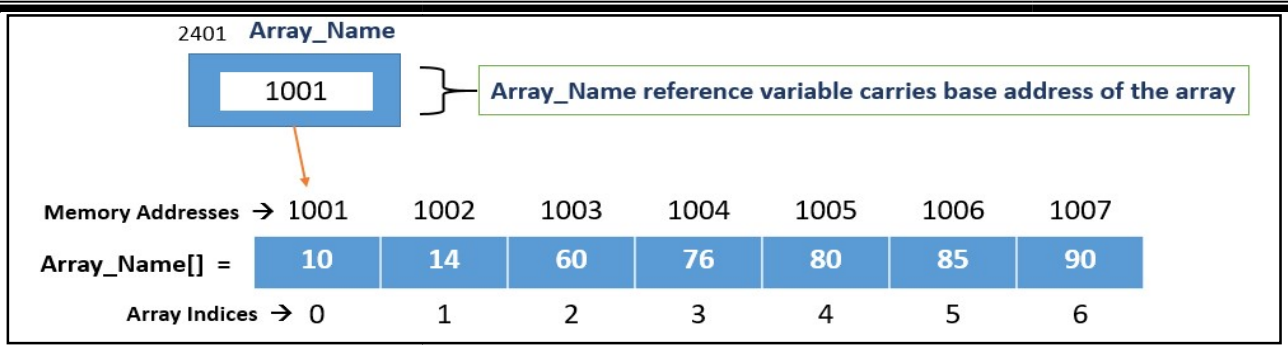


1. Array

- An array holds elements of the same type. It is an object in Java, and the array name (used for declaration) is a reference value that carries the base address of the continuous location of elements of an array.

Example:

```
int Array_Name = new int[7];
```



2.String

- The String data type stores a sequence or array of characters. A string is a non-primitive data type, but it is predefined in Java. String literals are enclosed in double quotes.

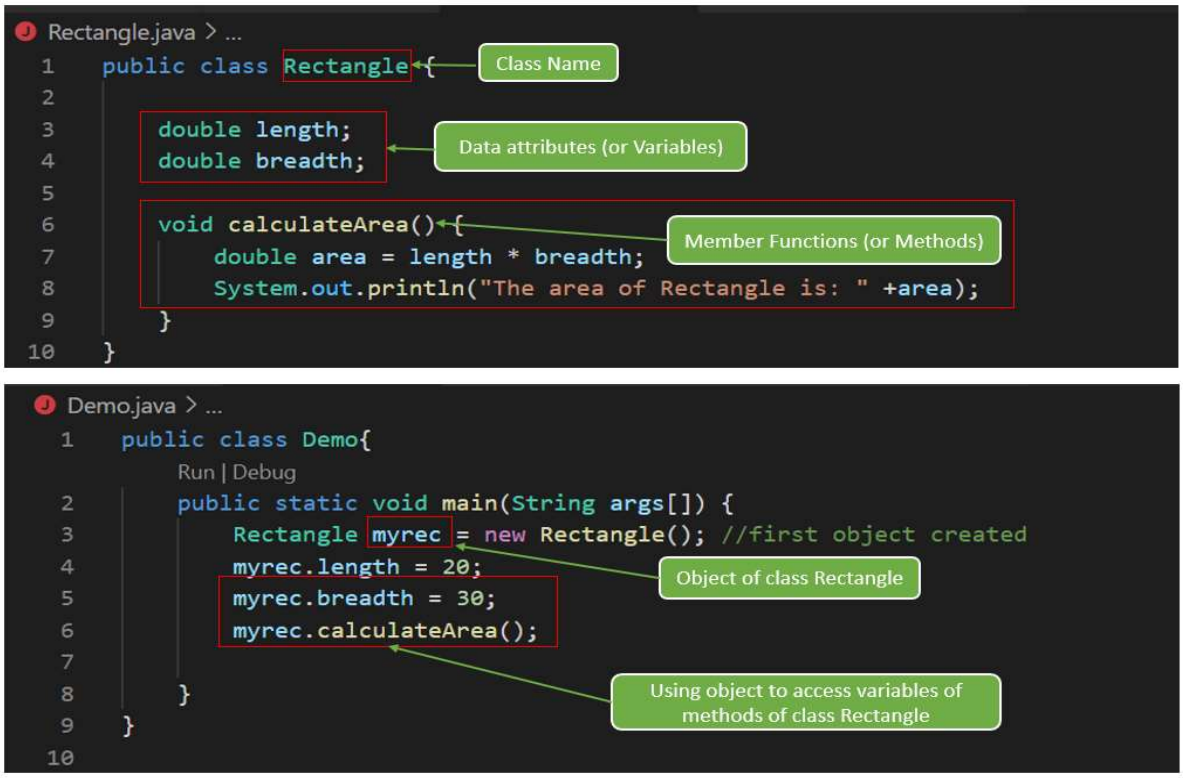
```
class Main {
  public static void main(String[] args) {

    // create strings
    String S1 = "Java String Data type";

    // print strings
    System.out.println(S1);
  }
}
```

3.Class

- A class is a user-defined data type from which objects are created. It describes the set of properties or methods common to all objects of the same type. It contains fields and methods that represent the behaviour of an object. A class gets invoked by the creation of the respective object.

- There are two types of classes: a blueprint and a template. **For instance**, the architectural diagram of a building is a **class**, and the building itself is an **object** created using the architectural diagram.

Example:



4.Interface

- An interface is declared like a class. The key difference is that the interface contains abstract methods by default; they have nobody.

Example:

```
interface printable {
void print();
}
class A1 implements printable {
public void print()
{
System.out.println("Hello");
}
public static void main(String args[]) {
A1 obj = new A1();
obj.print();
 }
}
```

5.Enum

- An enum, similar to a class, has attributes and methods. However, unlike classes, enum constants are public, static, and final (unchangeable – cannot be overridden). Developers cannot use an enum to create objects, and it cannot extend other classes. But, the enum can implement interfaces.

```
//declaration of an enum
enum Level {
  LOW,
  MEDIUM,
  HIGH
}
```

Java Operators

- Operators are symbols that perform operations on variables and values. For example, + is an operator used for addition, while * is also an operator used for multiplication.

  Operators in Java can be classified into 5 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.

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

Example 1: 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));
  }
}
```

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.

/ 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
```

% Modulo Operator

The modulo operator `%` computes the remainder. When `a = 7` is divided by `b = 4`, the remainder is **3**.

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.

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;

Example 2: 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;
    System.out.println("var using +=: " + var);

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

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`.

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**

Example 3: Relational Operators

```
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
  }
}
```

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

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

Example 4: 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
  }
}
```

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:

Increment and Decrement Operators

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

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**.

Example 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);
  }
}
```

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**).
The various bitwise operators present in Java are:

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.

Let's see an example of a ternary operator.

```
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);
  }
}
```

Output

```
Leap year
```

Java Control Statements

- Java compiler executes the code from top to bottom. The statements in the code are executed according to the order in which they appear.

- However, Java provides statements that can be used to control the flow of Java code. Such statements are called control flow statements. It is one of the fundamental features of Java, which provides a smooth flow of program.

Java provides three types of control flow statements.

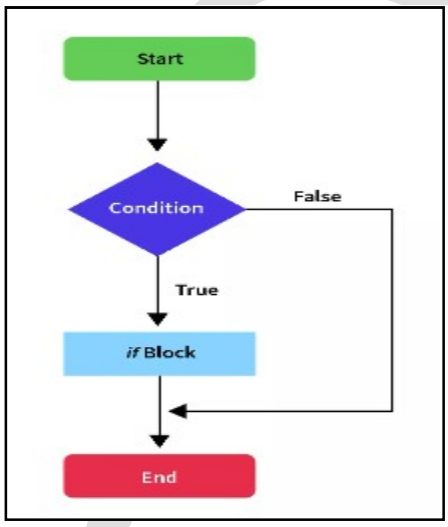
1. Decision Making statements
    - if statements
    - switch statement
2. Loop statements
    - do while loop
    - while loop
    - for loop
    - for-each loop
3. Jump statements
    - break statement
    - continue statement

Decision-Making statements:

As the name suggests, decision-making statements decide which statement to execute and when. Decision-making statements evaluate the Boolean expression and control the program flow depending upon the result of the condition provided.

1. if Statement

- These are the simplest and yet most widely used control statements in Java. The if statement is used to decide whether a particular block of code will be executed or not based on a certain condition.
- If the condition is true, then the code is executed otherwise not.
- Let's see the execution flow of the if statement in a flow diagram:



- In the above flow diagram, we can see that whenever the condition is true, we execute the if block otherwise we skip it and **continue the execution with the code following the** if **block.**

The syntax and execution flow of the if statement is as follows:

Syntax:

```
if(condition)
 {
   // block of code to be executed if the condition is true
}
```

- The if statement evaluates the given condition and if the condition is true, then the code inside the if block (marked by curly brackets) is executed.

- If we do not give curly brackets after the if statement then only the immediate statement following the if statement is considered to be inside the if block.

For example:

```
if(condition)
  statement1;
  statement2;
```

- Here statement1 will be executed only if the condition is true. The statement2 is executed irrespective of the condition being true or false.

Example:

```
class ifelseTest
{
    public static void main(String args[])
    {
        int x = 9;
        if (x > 10)
            System.out.println("i is greater than 10");
        else
            System.out.println("i is less than 10");
        System.out.println("After if else statement");
    }
}
```

Output:

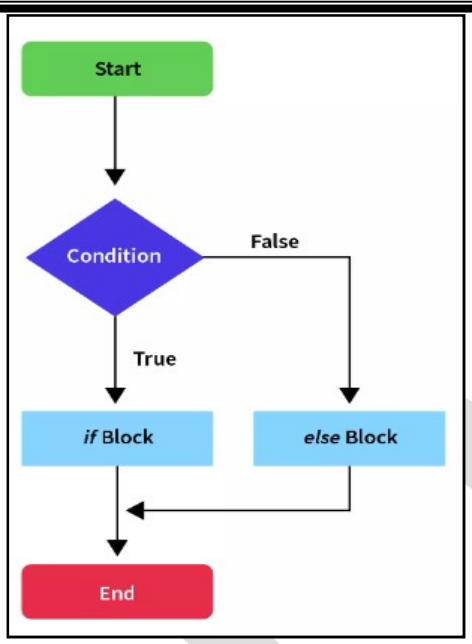
```
i is less than 10
After if else statement
```

2. if-else Statement

- The if statement is used to execute a block of code based on a condition. But if the condition is false and we want to do some other task when the condition is false, how should we do it?

- That's where else statement is used. In this, if the condition is true then the code inside the if block is executed otherwise the else block is executed.

- Let's see the execution flow of the if-else statement in a flow diagram:



- The above flow diagram is similar to the if statement, with a difference that whenever the condition is false, we execute the else block and then continue the normal flow of execution.
- The syntax and execution flow of the if-else statement is as follows:

Syntax:

```
if (condition) {
    // If block executed when the condition is true
}
else {
    // Else block executed when the condition is false
}
```

Example:

```
class ifelseTest
{
   public static void main(String args[])
   {
      int x = 9;
      if (x > 10)
         System.out.println("i is greater than 10");
      else
         System.out.println("i is less than 10");
   }
}
```

Output:
i is less than 10

Example:

```
public class Student
{
public static void main(String[] args)
{
 int x = 10;
 int y = 12;
 if(x+y < 10)
    {
 System.out.println("x + y is less than      10");
 }   else {
 System.out.println("x + y is greater than 20");
 }
}
}
```

Output:

x + y is greater than 20

3. Nested if-else Statement

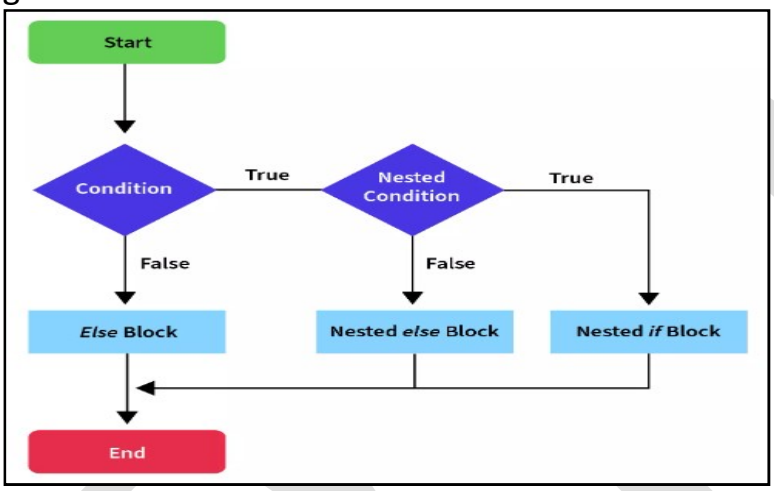
- java allows us to **nest control statements within control statements**.
- Nested control statements mean an if-else statement inside other if or else blocks. It is similar to an if-else statement but they are defined inside another if-else statement.
- Let us see the syntax of a specific type of nested control statements where an if-else statement is nested inside another if block.

Syntax:

```
if (condition) {
   // If block code to be executed if condition is true
   if (nested condition) {
      // If block code to be executed if nested condition is true
   }
   else {
      // Else block code to be executed if nested condition is false
   }
}
else {
   // Else block code to be executed if condition is false
}
```

Explanation:

- Here, we have specified another if-else block inside the first if block. In the syntax, you can see the series of the blocks executed according to the evaluation of condition and nested condition.
- Using this, we can nest the control flow statements in Java to evaluate multiple related conditions.
- Let's see the execution flow of the above-mentioned nested-if-else statement in a flow diagram:



Example

```
class nestedifTest {
   public static void main(String args[]) {
      int x = 25;
      if (x > 10)
      {
         if (x%2==0)
         System.out.println("i is greater than 10 and even number");
         else
         System.out.println("i is greater than 10 and odd number");
      }
      else
      {
         System.out.println("i is less than 10");
      }
      System.out.println("After nested if statement");
   }
}
```

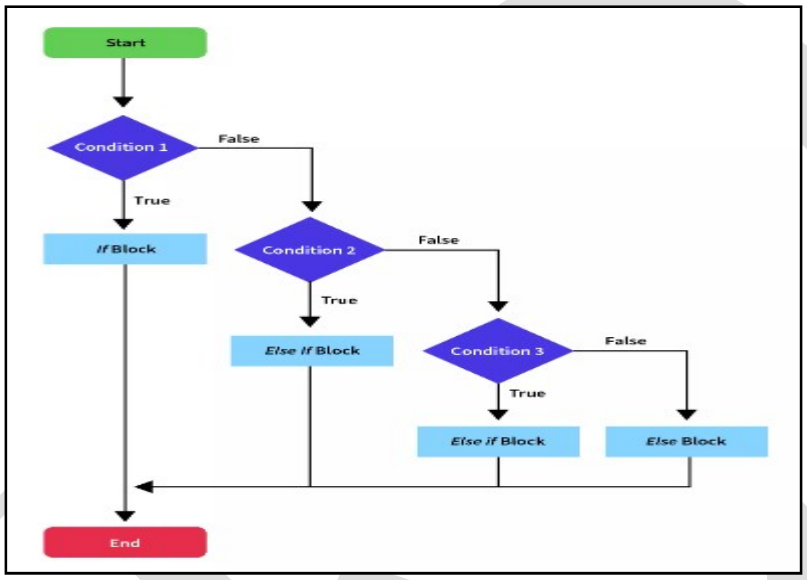
Output:

```
i is greater than 10 and odd number
After nested if statement
```

4. if-else-if Ladder

- In this, the if statement is followed by multiple else-if blocks. We can create a decision tree by using these control statements in Java in which the block where the condition is true is executed and the rest of the ladder is ignored and not executed.
- If none of the conditions is true, the last else block is executed, if present.

Let's see the execution flow of the if-else ladder in a flow diagram:



Syntax:

```
if(condition1) {
   // Executed only when the condition1 is true
}
else if(condition2) {
   // Executed only when the condition2 is true
}
.
.

else {
   // Executed when all the conditions mentioned above are true
}
```

Example

```
//DecisionTest03.java
public class DecisionTest03 {

    public static void main(String[] args) {
      int a = 30, b = 30;

      if(a > b) {
        System.out.println("a is bigger than b.");
      }
      else if(a < b) {
        System.out.println("a is smaller than b.");
      }
      else {
        System.out.println("a and b are equal.");
      }
    }
}
```

Output:

a and b are equal.

Example

```
// Java program to illustrate if-else-if ladder
import java.util.*;

class ifelseifDemo {
        public static void main(String args[])
        {
                int i = 20;

                if (i == 10)
                        System.out.println("i is 10");
                else if (i == 15)
                        System.out.println("i is 15");
                else if (i == 20)
                        System.out.println("i is 20");
                else
                        System.out.println("i is not present");
        }
}
```

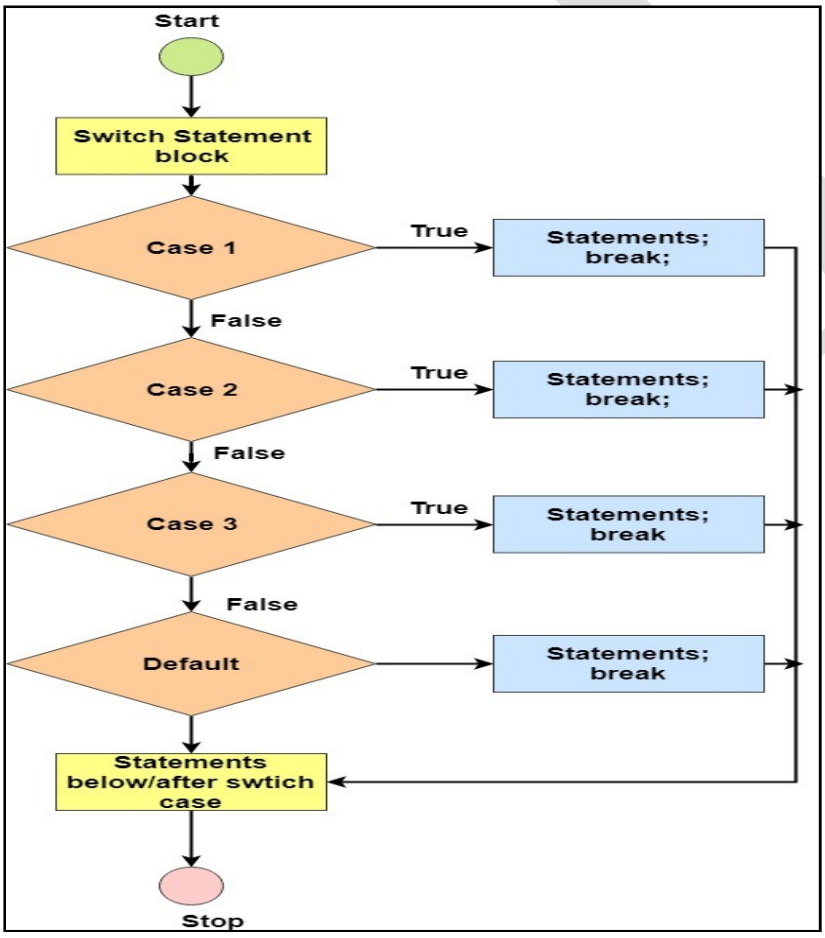
Output

```
i is 20
```

5. switch Statement

- Switch statements are almost similar to the if-else-if ladder control statements in Java. It is a multi-branch statement. It is a bit easier than the if-else-if ladder and also more user-friendly and readable.
- We should use the **switch-case statement** if we want to select one of many blocks of code to be executed.
- These blocks are called cases. We may also provide a default block of code that can be executed when none of the cases are matched similar to the else block.

The syntax and execution flow of the switch statement is as follows:



Syntax

```
switch(choice)
{
   case 1:
     execute code block 1
     break;
   case 2:
     execute code block 2
     break;
   case 3:
     execute code block 3
     break;
   default:
     execute code if choice is different
     break;
}
```

There are certain points that one needs to be remembered while using switch statements:

- The expression can be of type String, short, byte, int, char, or an enumeration.
- We cannot have any duplicate case values.
- The default statement is optional.

Example

```
//DecisionTest04.java
public class DecisionTest04 {
            public static void main(String[] args) {
    int choice = 3;
    switch(choice) {
      case 1:
        System.out.println("hello");
        break;
      case 2:
        System.out.println("hi");
        break;
      case 3:
        System.out.println("welcome");
        break;
      default:
        System.out.println("bye");
        break;
    }
  }
}
```

Output:

welcome

Example

```
public class exa
{
  public static void main(String[] args)
  {
String browser = "chrome";
switch (browser)
{
   case "safari":
      System.out.println("The browser is Safari");
   case "edge":
      System.out.println("The browser is Edge");
   case "chrome":
      System.out.println("The browser is Chrome");
   default:
      System.out.println("The browser is not supported");
}
}

}
```

Output:

```
The browser is Chrome
The browser is not supported
```

Looping Statement

- Looping in programming languages is a feature which facilitates the execution of a set of instructions/functions repeatedly while some condition evaluates to true. Java provides three ways for executing the loops. While all the ways provide similar basic functionality, they differ in their syntax and condition checking time.
- Imagine you are asked to print the value of 2^1, 2^2, 2^3, 2^4 …….2^{10}. One way could be taking a variable x=2 and printing and multiplying it with 2 each time.

```
public class Main {
 public static void main(String[] args) {
   int x = 2;
   /* printing value and multiplying it by 2 every time */
   System.out.print(x + " ");
   x *= 2;
   System.out.print(x + " ");
   x *= 2;
   System.out.print(x + " ");
   x *= 2;
```

```
    System.out.print(x + " ");
    x *= 2;
    System.out.print(x + " ");
    x *= 2;
    System.out.print(x + " ");
    x *= 2;
    System.out.print(x + " ");
    x *= 2;
    System.out.print(x + " ");
    x *= 2;
    System.out.print(x + " ");
    x *= 2;
    System.out.print(x + " ");

  }
}
```

Output:

```
2 4 8 16 32 64 128 256 512 1024
```

- Another way is using loops. Using Looping statements the above code of **10** lines can be reduced to **3** lines.

```
public class Main {
  public static void main(String[] args) {
    int x = 2;
    for (int i = 0; i < 10; i++) {
      System.out.print(x + " ");
      x *= 2;
    }
  }
}
```

Output:

```
2 4 8 16 32 64 128 256 512 1024
```

- **Need for Looping Constructs in Java**

A computer is most suitable for performing repetitive tasks and can tirelessly do tasks tens of thousands of times. We need Loops because

- Loops facilitates 'Write less, Do more' - We don't have to write the same code again and again.
- They **reduce** the **size** of the Code.
- Loops make an **easy flow of the control**.
- They also **reduce the Time Complexity and Space Complexity** - Loops are faster and more feasible as compared to Recursion.

In Java, there are three types of loops.

1. for loop
2. while loop
3. do...while loop

1. For Loop in Java

- **For Loop in Java** is a statement which allows code to be repeatedly executed. For loop contains 3 parts Initialization, Condition and Increment or Decrements.

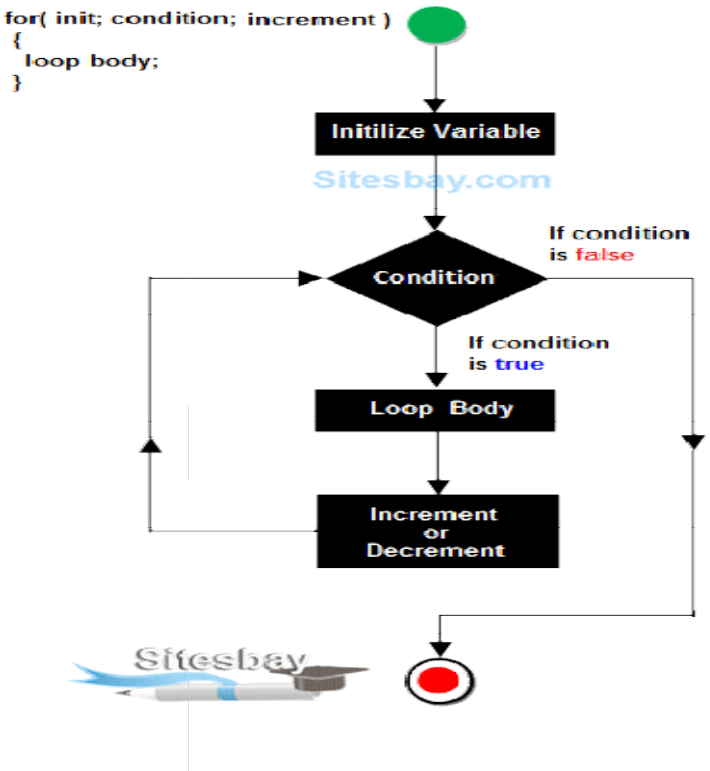
Syntax

```
for ( initialization; condition; increment/decrement )
{
 statement(s);
}
```

- **Initialization:** This step is execute first and this is execute only once when we are entering into the loop first time. This step is allow to declare and initialize any loop control variables.
- **Condition:** This is next step after initialization step, if it is true, the body of the loop is executed, if it is false then the body of the loop does not execute and flow of control goes outside of the for loop.

- **Increment or Decrements:** After completion of Initialization and Condition steps loop body code is executed and then Increment or Decrements steps is execute. This statement allows to update any loop control variables.

Flow Diagram



Example

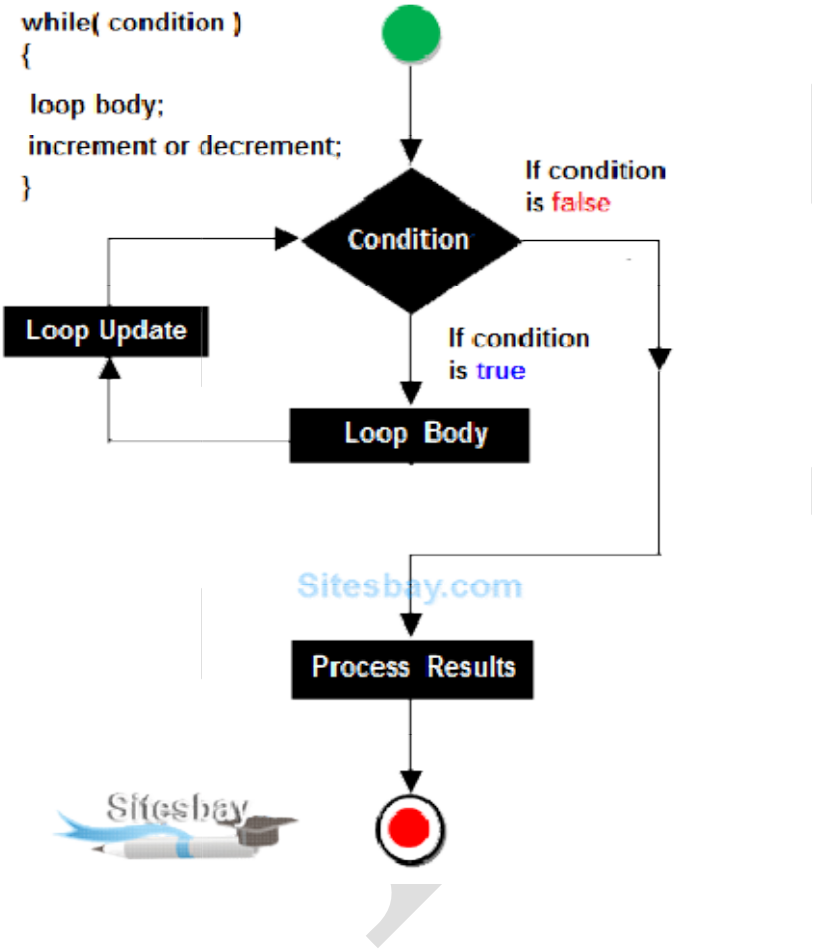
```
//Program to print even numbers between 1-10.

class ForLoopDemo
{
 public static void main(String[] args)
   {
    for (int i=1; i<=10; i++)
     {
       if (i%2==0)
       System.out.println(i);
      }
       System.out.println("Loop Ending");

   }
}
```

Output

```
2
4
6
8
10
Loop Ending
```

2)While Loop in Java

- In **while loop** in Java first check the condition if condition is true then control goes inside the loop body otherwise goes outside of the body. while loop will be repeats in clock wise direction.



Syntax

```
while(condition)
{
 Statement(s)
 Increment / decrements (++ or --);
}
```

Example :

```
// Program to display numbers from 1 to 5
class Main {
  public static void main(String[] args) {

    // declare variables
    int i = 1, n = 5;

    // while loop from 1 to 5
    while(i <= n) {
      System.out.println(i);
      i++;
    } }
}
```

Output:

```
1
2
3
4
5
```

Example:

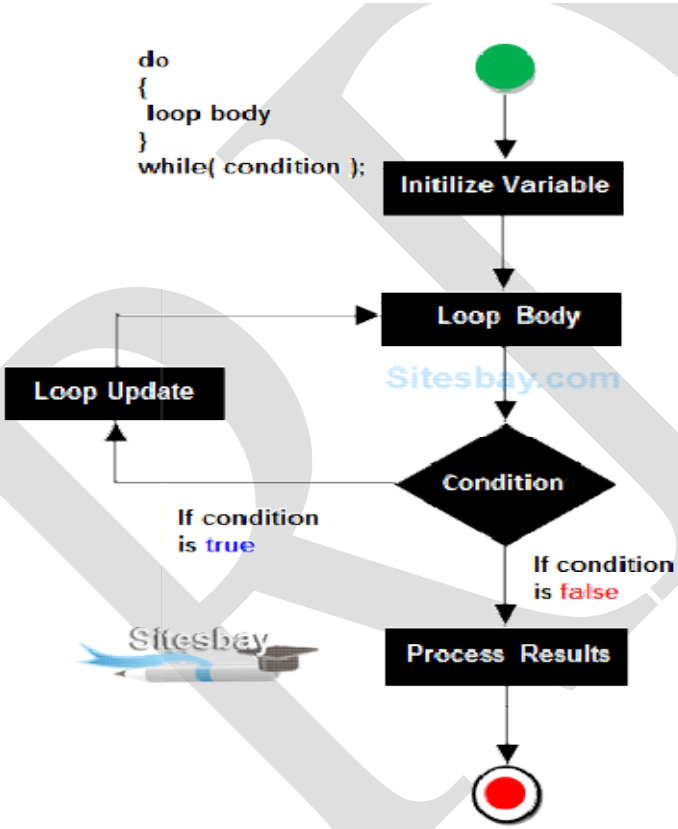
```
// Java program to find the sum of positive numbers
import java.util.Scanner;

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

    int sum = 0;

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

    System.out.println("Enter a number");
    int number = input.nextInt();

    // while loop continues
    // until entered number is positive
    while (number >= 0) {
      // add only positive numbers
      sum += number;

      System.out.println("Enter a number");
      number = input.nextInt();
    }
    System.out.println("Sum = " + sum);
  }}
```

Output

```
Enter a number
25
Enter a number
9
Enter a number
5
Enter a number
-3
Sum = 39
```

3)do-while Loop in Java

- A **do-while** loop in Java is similar to a while loop, except that a do-while loop is execute at least one time.
- A do while loop is a control flow statement that executes a block of code at least once, and then repeatedly executes the block, or not, depending on a given condition at the end of the block (in while).



When use do..while loop

- when we need to repeat the statement block **at least one time** then use do-while loop. In do-while loop post-checking process will be occur, that is after execution of the statement block condition part will be executed.

Syntax

```
do
{
Statement(s)
increment/decrement (++ or --)
} while();
```

Example

```
class  dowhileDemo
{
        public static void main(String args[])
        {
                 int i=20;
                do
                {
                        System.out.println("Hello world !");
                        i++;
                }while(i<10);
        }
}
```

Output

Hello world !

Example

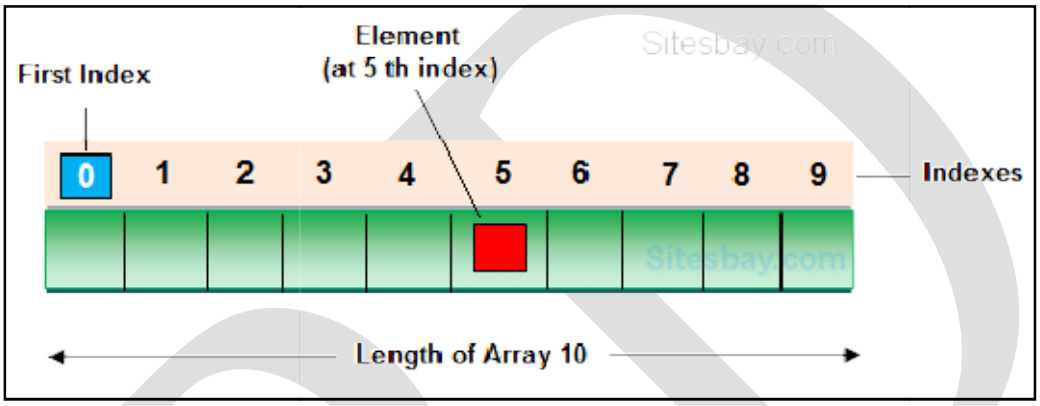
```
class  dowhileDemo
{
        public static void main(String args[])
        {
                int i=0;
                do
                {
                        System.out.println(+i);
                        i++;
                }while(i<5);
        }
}
```

Output

1
2
3
4
5

Array in java

- **Array** is a collection of similar type of data. It is fixed in size means that you can't increase the size of array at run time. It is a collection of homogeneous data elements. It stores the value on the basis of the index value.



Advantage of Array

One variable can store multiple value: The main advantage of the array is we can represent multiple value under the same name.

Code Optimization: No, need to declare a lot of variable of same type data, We can retrieve and sort data easily.

Random access: We can retrieve any data from array with the help of the index value.

Disadvantage of Array

- The main limitation of the array is **Size Limit** when once we declare array there is no chance to increase and decrease the size of an array according to our requirement, Hence memory point of view array concept is not recommended to use. To overcome this limitation in Java introduce the collection concept.

Types of Array in Java

- There are two types of array in Java.
  1. **Single Dimensional Array**
  2. **Two Dimensional Array**

Array Declaration

- Single dimension array declaration.

Syntax 1D Array

```
1. int[] a;
2. int a[];
3. int []a;
```

Note: At the time of array declaration we cannot specify the size of the array. For Example int[5] a; this is wrong.

- 2D Array declaration.

Syntax 2D Array

```
1. int[][] a;
2. int a[][];
3. int [][]a;
4. int[] a[];
5. int[] []a;
6. int []a[];
```

Array creation

- Every array in a Java is an object, Hence we can create array by using **new** keyword.

Syntax

```
int[] arr = new int[10];    // The size of array is 10.
or
int[] arr = {10,20,30,40,50};
```

Accessing array elements

- Access the elements of array by using index value of an elements.

Syntax Array in Java

```
arrayname[index number];
```

Access Array Elements

```
int[] arr={10,20,30,40};
System.out.println("Element at 3rd place"+arr[2]);
```

Example of Array

```
public class ArrayEx
{
public static void main(String []args)
{
int arr[] = {10,20,30};
for (int i=0; i < arr.length; i++)
{
 System.out.println(arr[i]);
 }}  }
```

Output

```
10
20
30
```

Note:

1) At the time of array creation we must be specify the size of array otherwise get an compile time error. For Example

int[] a=new int[]; Invalid.

int[] a=new int[5]; Valid

2) If we specify the array size as negative int value, then we will get run-time error,

NegativeArraySizeException.

3) To specify the array size the allowed data types are byte, short, int, char If we use other data type then we will get an Compile time error.

4) The maximum allowed size of array in **Java is 2147483647** (**It is the maximum value of int data type)**

Difference Between Length and Length() in Java

length: It is a final variable and only applicable for array. It represent size of array.

Example

```
int[] a=new int[10];
System.out.println(a.length); // 10
System.out.println(a.length()); // Compile time error
```

length(): It is the final method applicable only for String objects. It represents the number of characters present in the String.

Example

```
String s="Java";
System.out.println(s.length());  // 4
System.out.println(s.length); // Compile time error
```

Java Command Line Arguments

- The java command-line argument is an argument i.e. passed at the time of running the java program.
- The arguments passed from the console can be received in the java program and it can be used as an input.

Example

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

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

```
compile by > javac CommandLineExample.java
run by > java CommandLineExample xyz
Output: Your first argument is: xyz
```

Example of command-line argument that prints all the values

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

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

compile by > javac A.java
run by > java A abc xyz 1 3

Output:

abc
xyz
1
3