CSE1007 : Programming Java
Java Basics

Dr. Pradeep K V
Assistant Professor (Sr.)
School of Computer Science and Engineering
VIT - Chennai

Contents

What is Java

- Java is a popular programming language.
- JAVA was developed by Sun Microsystems Inc in the year 1991,
- It is owned by Oracle, and more than 3 billion devices run Java.
- It was developed by **James Gosling** and **Patrick Naughton**.
- Java is a Open Source, High level, Robust, Object-oriented and Secure programming language.
- **Platform**: Any hardware or software environment in which a program runs, is known as a platform. Since Java has a runtime environment (JRE) and API, it is called a platform.
- Java works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc.)

There are many devices where Java is currently used and are as follows:

- Desktop Applications such as acrobat reader, media player, antivirus, etc.
- Web Applications such as irctc.co.in, javatpoint.com, etc.
- Enterprise Applications such as banking applications.
- Mobile
- Embedded System
- Smart Card
- Robotics
- Games, etc.

There are 4 types of applications that can be created using Java programming:

- Standalone Application
- Web Applications
- Enterprise Application
- Mobile Application

There are 4 Platforms or Editions of Java:

- Java SE (Java Standard Edition) : It includes Java programming APIs such as java.lang, java.io, java.net, java.util, java.sql, java.math etc. It includes core topics like OOPs, String, Regex, Exception, Inner classes, Multithreading, I/O Stream, Networking, AWT, Swing, Reflection, Collection, etc.
- Java EE (Java Enterprise Edition) : It is built on the top of the Java SE platform. It includes topics like Servlet, JSP, Web Services, EJB, JPA, etc.
- Java ME (Java Micro Edition) : used to develop mobile applications.
- JavaFX : used to develop rich internet applications

1. Simple : is very easy to learn, its syntax is simple, clean and easy.
2. Object-Oriented : Everything in Java is an object. OOPs is a methodology that simplifies software development and maintenance by providing some rules(Object, Class, Inheritance, Polymorphism, Abstraction, Encapsulation).
3. Platform independent : it facilitates you to carry the Java bytecode to any platform.
4. Secured : can develop virus-free systems because of
   - No explicit pointer
   - Java Programs run inside a virtual machine sandbox
5. Robust :
   - It uses strong memory management.
   - There is a lack of pointers that avoids security problems.
   - There is automatic garbage collection in java 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.

6. Architecture neutral : there are no implementation dependent.
7. High Performance : Java is faster than other traditional interpreted programming languages because Java bytecode is "close" to native code.
8. Multithreaded : A thread is like a separate program, executing concurrently.
9. Distributed : it facilitates users to create distributed applications in Java using RMI and EJB.
10. Dynamic : Classes are loaded on demand and also supports functions from its native languages, i.e., C and C++.

C++ Verses Java

Comparison Index	C++	Java
Platform	Dependent	Independent.
Mainly used for	System programming.	Application/Window/ Web-based/Mobile programming.
Design Goal	System/Applications programming.	Network Computing.
Goto statement	Yes	No
Multiple inheritance	Yes	interfaces in java.
Operator Overloading	Yes	No
Pointers	Yes	No
Compiler and Interpreter	Compiler	Both
Call by Value and Call by reference	Yes	Only Call by Value

Comparison Index	C++	Java
Structure and Union	Yes	No
Thread Support	No	Yes
Documentation comment	No	Yes
Virtual Keyword	Yes	No
unsigned right shift >>>	No	Yes
Inheritance Tree	Yes	Single
Hardware	Nearer	Far
Object-oriented	Partially	Fully

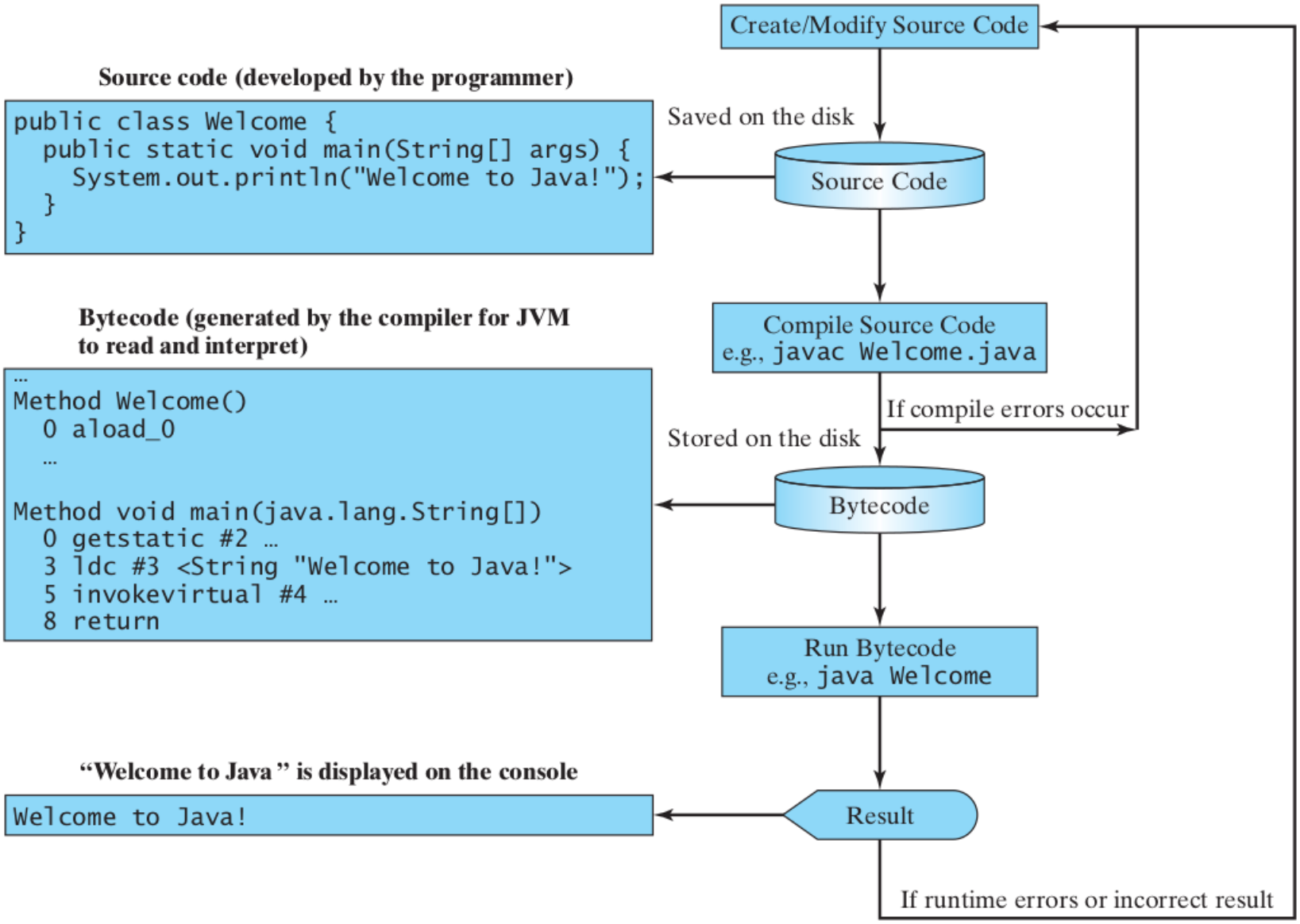
First Java Program

```
/* FileName : "Welcome.java". */

class Welcome
{
    // Your program begins with a call to main().
    public static void main(String args[])   {
        System.out.println("Welcome to Java");
    }
}
```

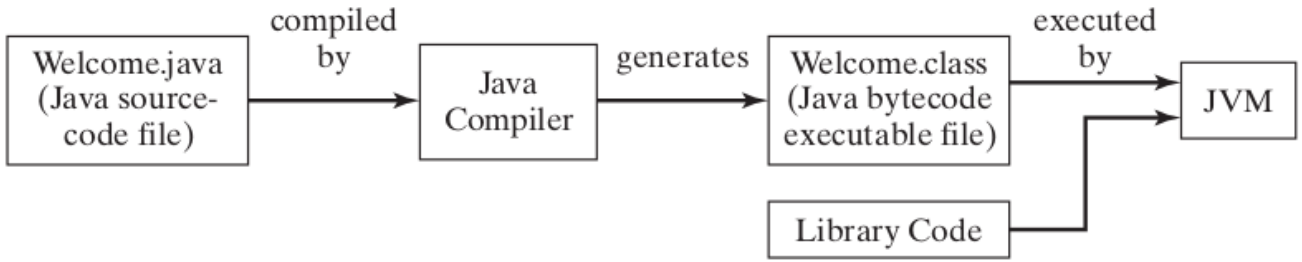
The basic understanding of the code:

1. Class definition : uses the keyword class to declare a new class. Welcome is an identifier and is the name of the class.
2. main() : Starting Point, every application must contain.
3. **public**: JVM can execute the method from anywhere.
4. **static**: Main method is to be called without object.
5. **void**: The main method doesn't return anything.
6. **String**[]: command line arguments
7. **System.out.println**() is used to print statement.
   Here, System is a class, out is the object of PrintStream class, println() is the method of PrintStream class

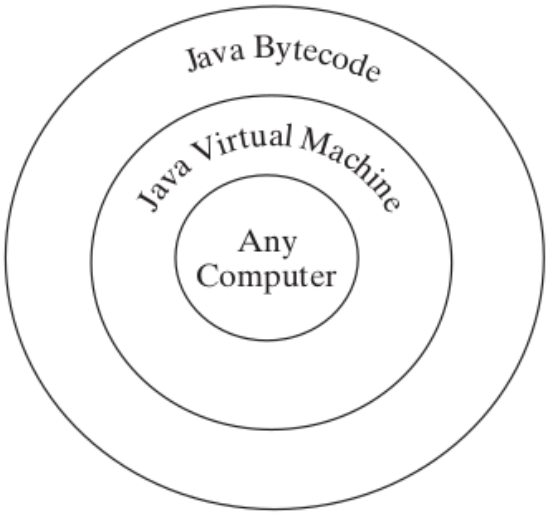
Java Compilation Process I



Java Compilation Process II



(a)



(b)

Modifications in Java Program

1. By changing the sequence of the modifiers, method prototype is not changed in Java

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

2. The subscript notation in Java array can be used after type, before the variable or after the variable.

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

3. You can provide var-args support to the main method by passing 3 ellipses (dots)

```
public static void main(String... args)
```

4. Having a semicolon at the end of class is optional in Java.

Valid/Invalid main() Signatures

Valid java main method signature

```
public static void main(String[] args)
public static void main(String []args)
public static void main(String args[])
public static void main(String... args)
static public void main(String[] args)
public static final void main(String[] args)
final public static void main(String[] args)
final strictfp public static void main(String[] args)
```

Invalid java main method signature

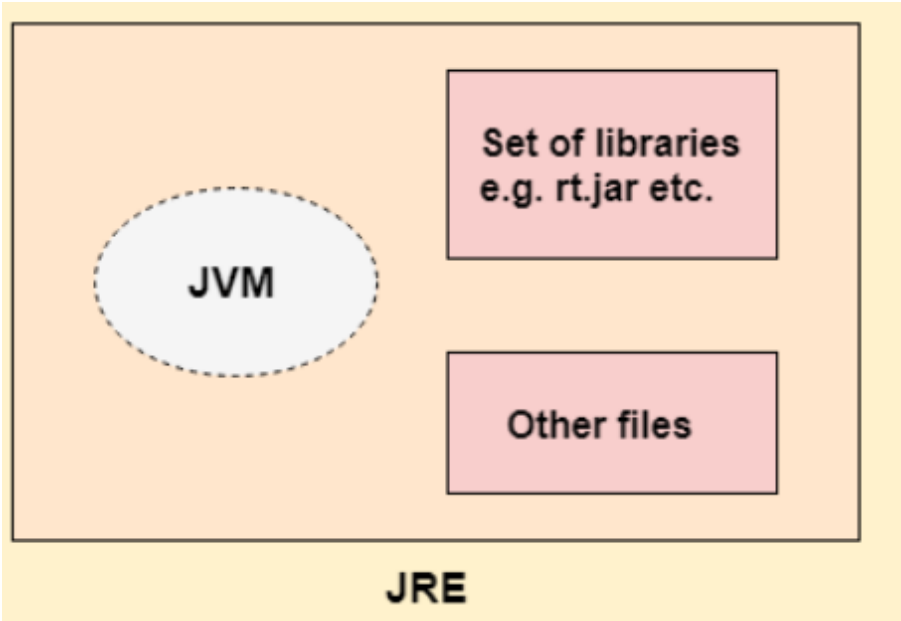
```
public void main(String[] args)
static void main(String[] args)
public void static main(String[] args)
abstract public static void main(String[] args)
```

JVM - Java Virtual Machine

- JVM (Java Virtual Machine) is an abstract machine.
- It is called a virtual machine because it doesn't physically exist.
- It is a specification that provides a runtime environment in which Java bytecode can be executed.
- It can also run those programs which are written in other languages and compiled to Java bytecode.
- JVMs are available for many hardware and software platforms.
- The JVM performs the following main tasks:
  - Loads code
  - Verifies code
  - Executes code
  - Provides runtime environment

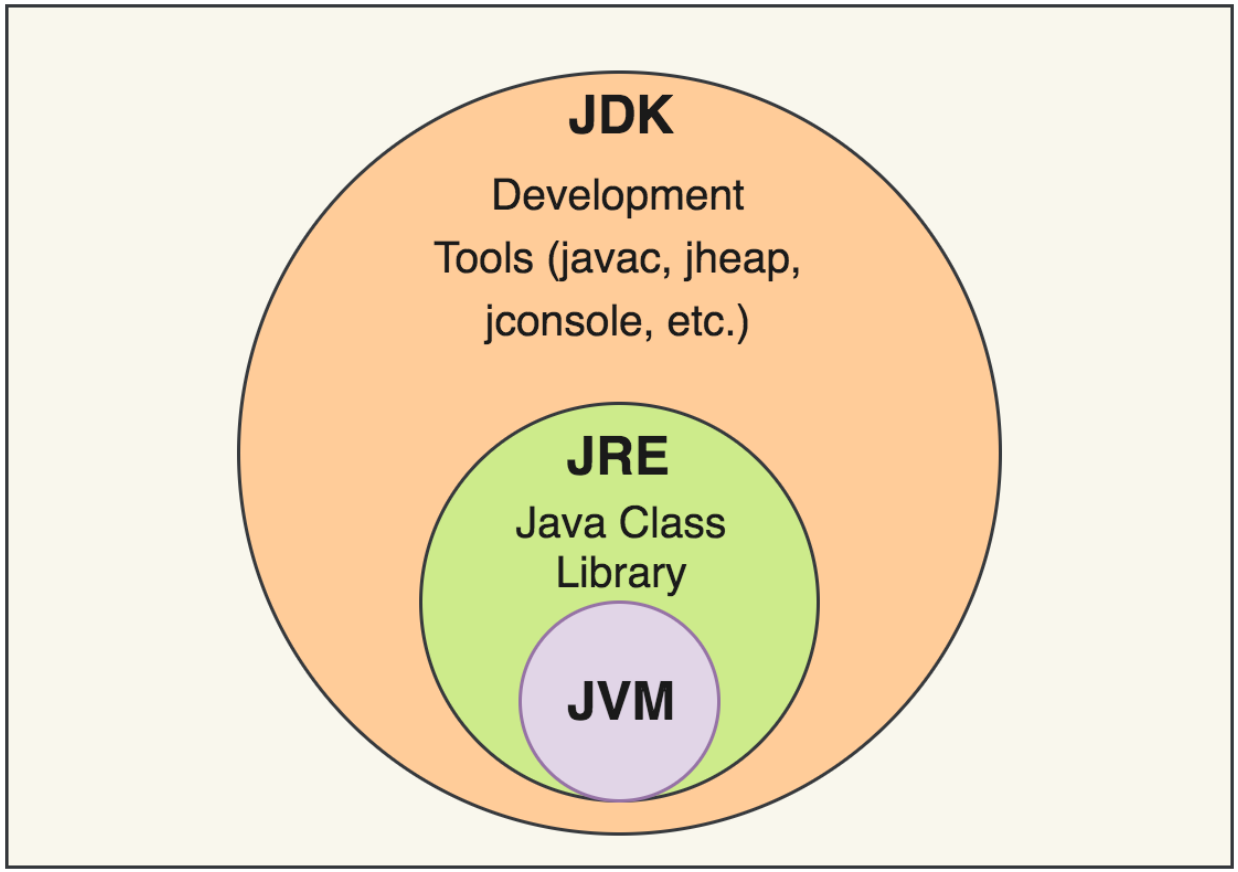
JRE - Java Runtime Environment

- JRE is an acronym for Java Runtime Environment.
- It is also written as Java RTE.
- The Java Runtime Environment is a set of software tools which are used for developing Java applications.
- It physically exists and contains a set of libraries + other files that JVM uses at runtime.
- The implementation of JVM is also actively released by other companies besides Sun Micro Systems.



JDK - Java Development Kit

- It is a software development environment which is used to develop Java applications and applets.
- It physically exists and contains JRE + development tools.



- Comments can be used to explain Java code, and to make it more readable.
- It can also be used to prevent execution when testing alternative code.

1. Single Line Comment

```
// This is a comment
System.out.println("Hello World");
System.out.println("Hello World");   // This is a comment
```

2. Multi Line Comment

```
/* The code below will print the words Hello World
     to the screen, and it is amazing */
System.out.println("Hello World");
```

Java Identifiers

- They are used for identification purposes.
- It is can be a class name, method name, variable name, or label.
- For example :

```
public class Test{
        public static void main(String[] args){
            int a = 20;
        }
}
```

- In the above java code, we have 5 identifiers namely :
  - Test : class name.
  - main : method name.
  - String : predefined class name.
  - args : variable name.
  - a : variable name.

Rules to define Identifiers

There are certain rules for defining a valid java identifier. These rules must be followed, otherwise we get compile-time error

- The only allowed characters for identifiers are all alphanumeric characters([A-Z],[a-z],[0-9]), $(dollar sign) and _ (underscore).
  For example "Deepu@" is not a valid java identifier as it contain '@' special character.
- Identifiers should not start with digits([0-9]).
  For example "123Deepu" is a not a valid java identifier.
- Java identifiers are case-sensitive.
- There is no limit on the length of the identifier but it is advisable to use an optimum length of 4 – 15 letters only.
- Reserved Words can't be used as an identifier.
  For example "int while = 20;" is an invalid statement as while is a reserved word. There are 53 reserved words in Java.

Valid - Invalid Identifiers

Examples of valid identifiers :

```
MyVariable, MYVARIABLE
myvariable, x
i, x1, i1,
_myvariable, $myvariable
sum_of_array, Deepu123
```

Examples of invalid identifiers :

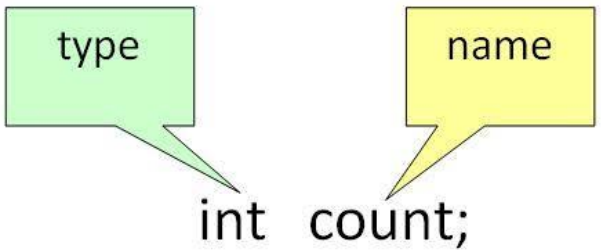
```
My Variable          // contains a space
123Deepu           // Begins with a digit
a+c                  // + sign is not an alphanumeric character
variable-2         // hyphen is not an alphanumeric character
sum_&_difference// ampersand is not an alphanumeric character
```

- Any programming language reserves some words to represent functionalities defined by that language.
- Types of Reserve Words
  - keywords(50) - define functionalities and
  - literals(3) - define a value.

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

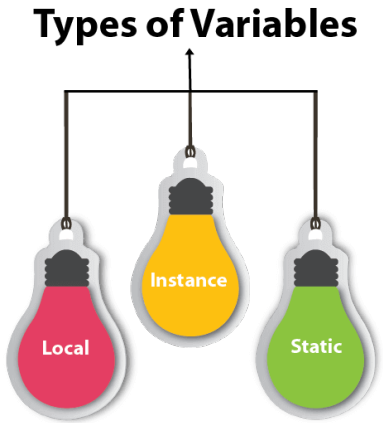
It is a name given to a memory location. It is the basic unit of storage in a program.

- The value stored in a variable can be changed during program execution.
- A variable is only a name given to a memory location, all the operations done on the variable effects that memory location.
- In Java, all the variables must be declared before use.



- Examples :

```
float simpleInterest;              //Declaring
int time = 10, speed = 20;         //Declaring and Initializing
char var='h';                      //Declaring and Initializing
```



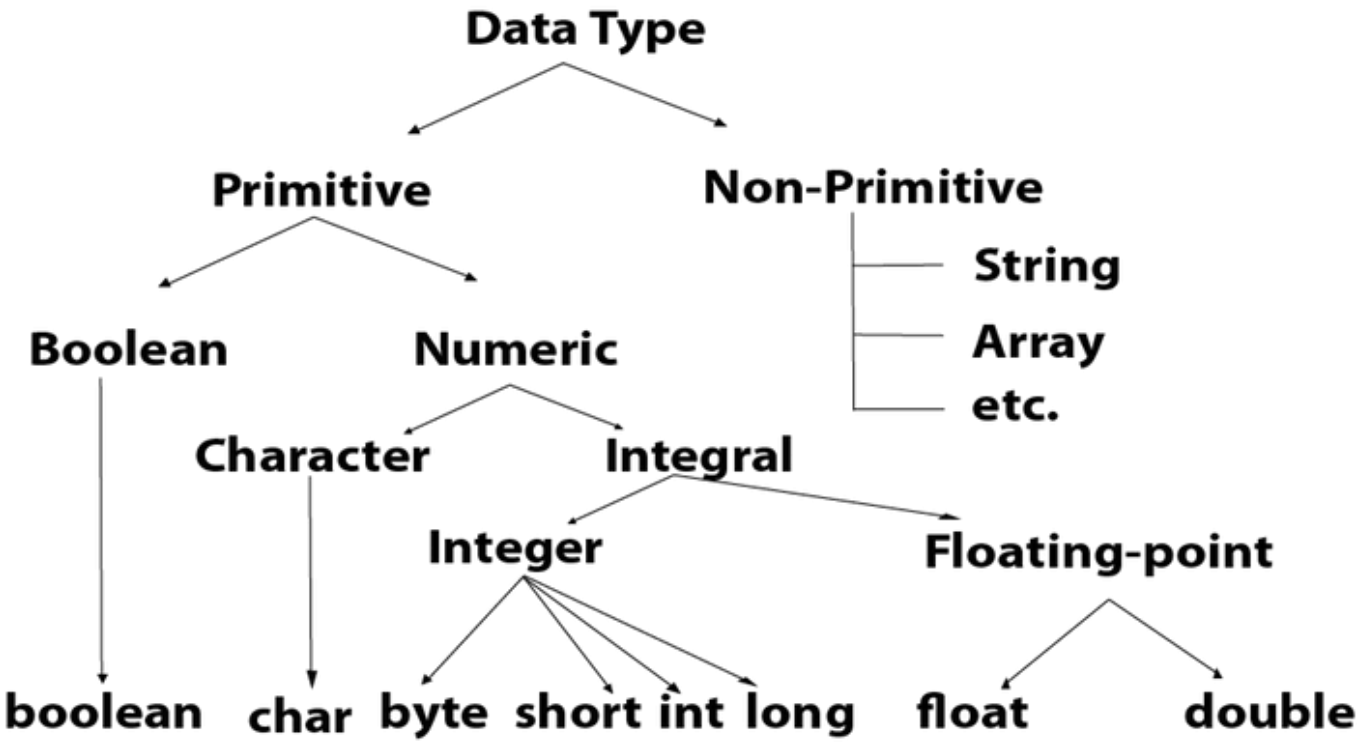
- Static : is declared as static and can create a single copy of static variable and share among all the instances of the class.
  Memory allocation for static variable happens only once when the class is loaded in the memory.
- Instance : declared inside the class but outside the body of the method.
- Local : declared inside the body of the method and the other methods in the class aren't even aware that the variable exists.

```
class A{
        int data=50;                    //instance variable
        static int m=100;               //static variable
        void method(){
                int n=90;               //local variable
        }
}
```

Java Variable Example: Add Two Numbers

```
class Simple{
        public static void main(String[] args){
                int a=10;
                int b=10;
                int c=a+b;
                System.out.println(c);
        }
}
```

Java Data types



Java provides many types of operators which can be used according to the need. They are classified based on the functionality they provide. Some of the types are:

- Arithmetic Operators : +, -, *, /,
- Unary Operators : +, -, ++, –, !
- Assignment Operator : =, +=, -=, *=, /=,
- Relational Operators : >, <, <=, >=, ==, !=
- Logical Operators : &&, ||
- Ternary Operator : ?:
- Shift Operators : >>, <<, >>>
- Bitwise Operators : &, |

and soon...

Table: Arithmetic and Assignment Operators

Operator	Name	Description	Example
+	Addition	Adds together two values	x + y
-	Subtraction	Subtracts one value from another	x - y
*	Multiplication	Multiplies two values	x * y
/	Division	Divides one value by another	x / y
%	Modulus	Returns the division remainder	x % y
++	Increment	Increases the value of a variable by 1	++x
−−	Decrement	Decreases the value of a variable by 1	−−x

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3

Operator	Example	Same As
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^3
>>=	x >>= 3	x = x >>3
<<=	x <<= 3	x = x <<3

Table: Relational Operators

Operator	Name	Example
==	Equal to	x == y
!=	Not equal	x != y
>	Greater than	x >y
<	Less than	x <y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Table: Logical Operator

Operator	Name	Description	Example
&&	Logical and	Returns true if both statements are true	x <5 && x <10
||	Logical or	Returns true if one of the statements is true	x <5 || x <4
!	Logical not	Reverse the result, returns false if the result is true	!(x <5 && x <10)

Table: Bitwise Operators

Operator	Example	Same as	Result	Decimal
&	5 & 1	0101 & 0001	0001	1
|	5 | 1	0101 | 0001	0101	5
~	~5	~0101	1010	10
^	5 ^1	0101 ^0001	0100	4
<<	9 <<1	1001 <<1	0010	2
>>	9 >>1	1001 >>1	1100	12
>>>	9 >>>1	1001 >>>1	0100	4

Type casting is when you assign a value of one primitive data type to another type and are of two types in Java.

- Widening Casting (automatically) : converting a smaller to a larger type.
- Narrowing Casting (manually) : converting a larger to a smaller size type.

Widening Casting :

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

Narrowing Casting :

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

Examples

```java
// Widening Casting
public class Main {
  public static void main(String[] args) {
    int myInt = 9;
    double myDouble = myInt; // Automatic casting: int to double

    System.out.println(myInt);      // Outputs 9
    System.out.println(myDouble);   // Outputs 9.0
  }
}
```

```java
// Narrowing Casting
public class Main {
  public static void main(String[] args) {
    double myDouble = 9.78;
    int myInt = (int) myDouble; // Manual casting: double to int

    System.out.println(myDouble);   // Outputs 9.78
    System.out.println(myInt);      // Outputs 9
  }
}
```

There are three different ways for reading input from the user in the command line environment(console).

- Using Buffered Reader Class : is a classical java method, used by wrapping the System.in (standard input stream) in an InputStreamReader which is wrapped in a BufferedReader (efficient).
- Using Scanner Class : is used to parse primitive types and strings using regular expressions. (nextInt(), nextFloat(), . . .).
- Using Console Class : it can be used for reading password-like input without echoing the characters entered by the user; the format string syntax can also be used (like System.out.printf()).

BufferReader Example


```java
// Java program to demonstrate BufferedReader
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
public class Test
{
    public static void main(String[] args) throws IOException
    {
        //Enter data using BufferReader
        BufferedReader reader =
                    new BufferedReader(new InputStreamReader(System.in));

        // Reading data using readLine
        String name = reader.readLine();

        // Printing the read line
        System.out.println(name);
    }
}
```

```java
// Java program to demonstrate working of Scanner in Java
import java.util.Scanner;

class GetInputFromUser
{
    public static void main(String args[])
    {
        // Using Scanner for Getting Input from User
        Scanner in = new Scanner(System.in);

        String s = in.nextLine();
        System.out.println("You entered string "+s);

        int a = in.nextInt();
        System.out.println("You entered integer "+a);

        float b = in.nextFloat();
        System.out.println("You entered float "+b);
    }
}
```

```java
// Java program to demonstrate working of System.console()
// Note that this program does not work on IDEs as
// System.console() may require console

public class Sample
{
    public static void main(String[] args)
    {
        // Using Console to input data from user
        String name = System.console().readLine();

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

- Decision Making in programming is similar to decision making in real life.
- In programming also we face some situations where we want a certain block of code to be executed when some condition is fulfilled.
- A programming language uses control statements to control the flow of execution of program based on certain conditions.
- These are used to cause the flow of execution to advance and branch based on changes to the state of a program.

Sr.No.	Statement & Description
1	if statement : consists of a boolean expression followed by one or more statements.
2	if...else statement : can be followed by an optional else statement, which executes when the boolean expression is false.
3	nested if statement : can use one if or else if statement inside another if or else if statement(s).
4	switch statement : allows a variable to be tested for equality against a list of values.

Example - if


```java
// Java program to illustrate If statement
class IfDemo
{
    public static void main(String args[])
    {
        int i = 10;

        if (i > 15)
            System.out.println("10 is less than 15");

        // This statement will be executed
        // as if considers one statement by default
        System.out.println("I am Not in if");
    }
}
```

```java
// Java program to illustrate if-else statement
class IfElseDemo
{
    public static void main(String args[])
    {
        int i = 10;

        if (i < 15)
            System.out.println("i is smaller than 15");
        else
            System.out.println("i is greater than 15");
    }
}
```

Example - nested if


```java
// Java program to illustrate nested-if statement
class NestedIfDemo {
    public static void main(String args[]) {
        int i = 10;

        if (i == 10) {
            // First if statement
            if (i < 15)
                System.out.println("i is smaller than 15");

            // Second if statement
            if (i < 12)
                System.out.println("i is smaller than 12 too");
            else
                System.out.println("i is greater than 15");
        }
    }
}
```

Example - if-else-if Ladder

```java
// Java program to illustrate if-else-if ladder
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");
    }
}
```

```java
// Java program to illustrate switch-case
class SwitchCaseDemo
{
    public static void main(String args[])
    {
        int i = 9;
        switch (i) {
        case 0:
            System.out.println("i is zero.");
            break;
        case 1:
            System.out.println("i is one.");
            break;
        case 2:
            System.out.println("i is two.");
            break;
        default: System.out.println("i is greater than 2.");
        }
    }
}
```

Java supports three jump statement: break, continue and return. These three statements transfer control to other part of the program.

1. **jump** : is used for
   - Terminate a sequence in a switch statement (discussed above).
   - To exit a loop.
   - Used as a "civilized" form of goto.
2. **break** : can force immediate termination of a loop, bypassing the conditional expression and any remaining code in the body of the loop.
3. **continue** : Sometimes it is useful to force an early iteration of a loop.
4. **return** : is used to explicitly return from a method and transfer program control back to the caller of the method.

```java
// Java program to illustrate using break to exit a loop

class BreakLoopDemo
{
    public static void main(String args[])
    {
        // Initially loop is set to run from 0-9
        for (int i = 0; i < 10; i++)
        {
            // terminate loop when i is 5.
            if (i == 5)
                break;

            System.out.println("i: " + i);
        }
        System.out.println("Loop complete.");
    }
}
```

Example - goto


```java
// Java program to illustrate using break with goto
class BreakLabelDemo {
    public static void main(String args[])     {
        boolean t = true;

        first: {                    // label first
           second: {                    // label second
                third: {            // label third
                    // Before break
                    System.out.println("Before the break statement");

                    // break will take the control out of second label
                    if (t) break second;
                    System.out.println("This won't execute.");
                }
                System.out.println("This won't execute.");
            }
            System.out.println("This is after second block.");
        }
    }
}
```

Example - continue

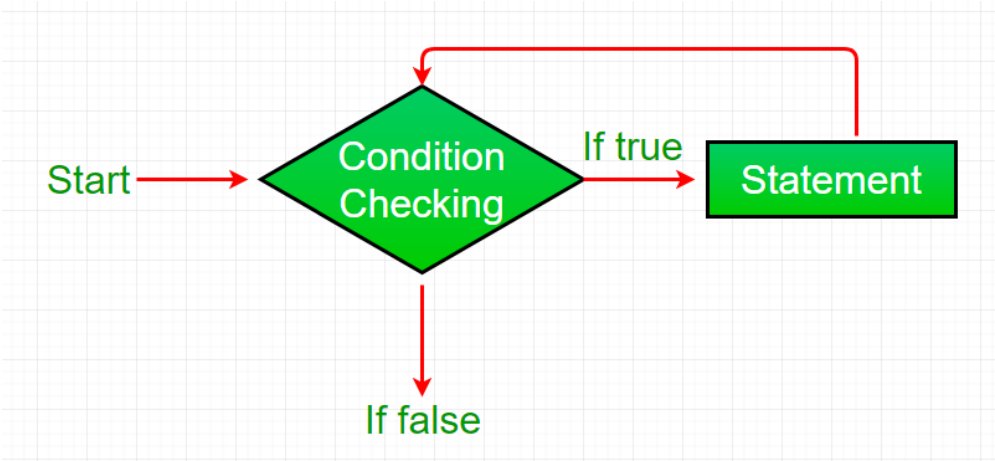
```java
// Java program to illustrate using continue in an if statement
class ContinueDemo
{
    public static void main(String args[])
    {
        for (int i = 0; i < 10; i++)
        {
            // If the number is even
            // skip and continue
            if (i%2 == 0)
                continue;

            // If number is odd, print it
            System.out.print(i + " ");
        }
    }
}
```

Example - return

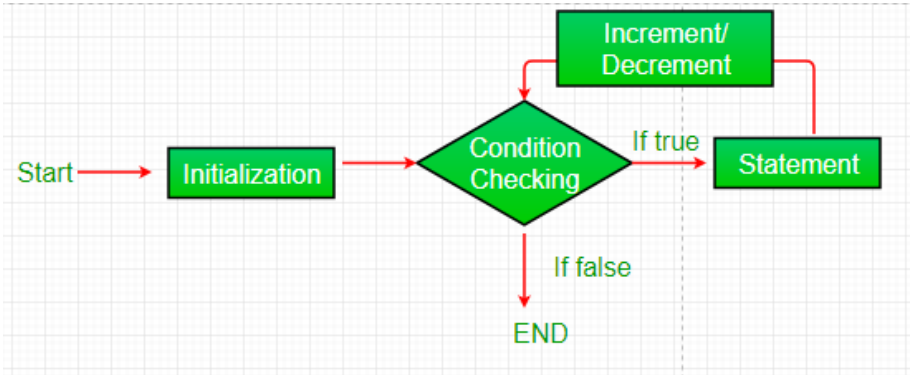

```java
// Java program to illustrate using return
class Return
{
    public static void main(String args[])
    {
        boolean t = true;
        System.out.println("Before the return.");

        if (t)
            return;

        // Compiler will bypass every statement
        // after return
        System.out.println("This won't execute.");
    }
}
```

- 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.

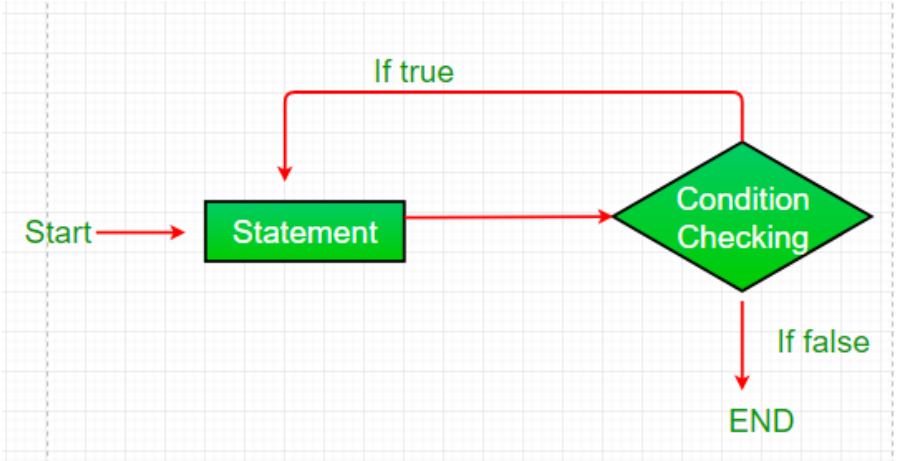
Sr.No.	Loop & Description
1	while loop : Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.
2	for loop : Execute a sequence of statements multiple times and abbreviates the code that manages the loop variable.
3	do...while loop : Like a while statement, except that it tests the condition at the end of the loop body.



while loop

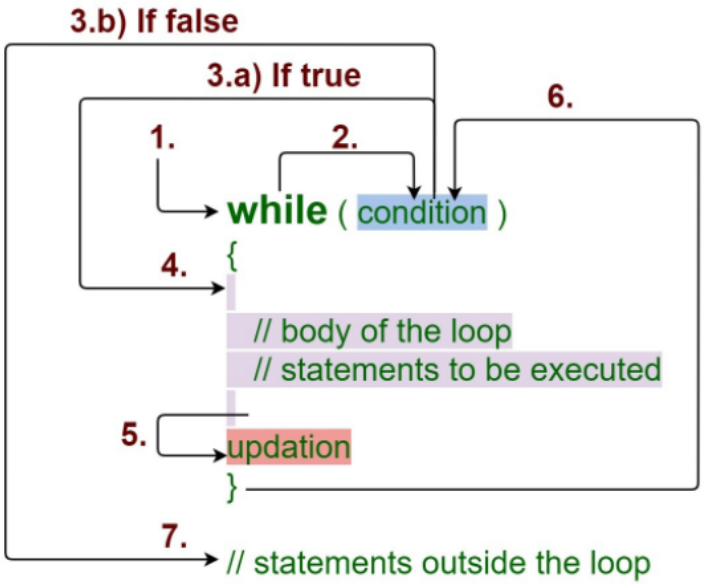


for loop

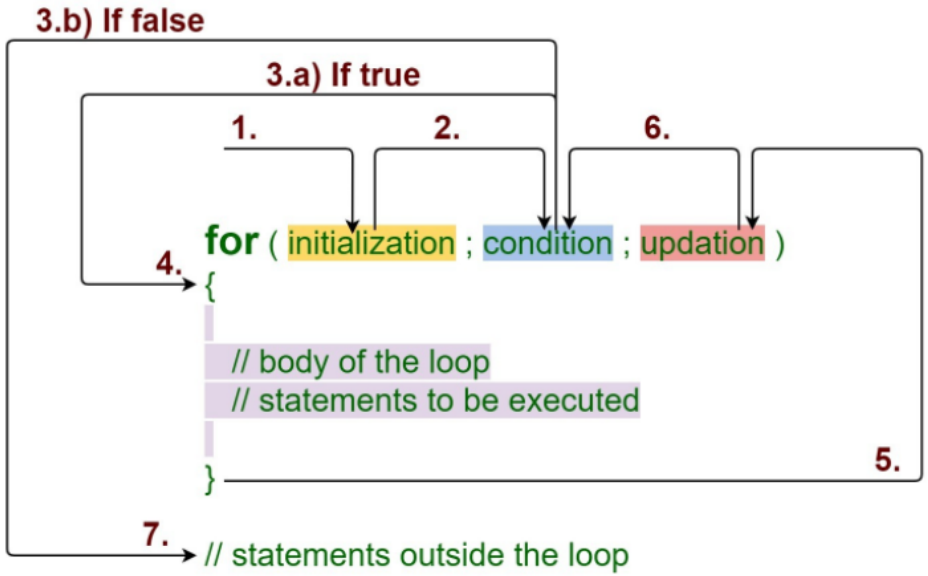


do..while loop

While Loop



For Loop



Example - while


```java
// Java program to illustrate while loop
class whileLoopDemo
{
    public static void main(String args[])
    {
        int x = 1;

        // Exit when x becomes greater than 4
        while (x <= 4)
        {
            System.out.println("Value of x:" + x);

            // Increment the value of x for
            // next iteration
            x++;
        }
    }
}
```

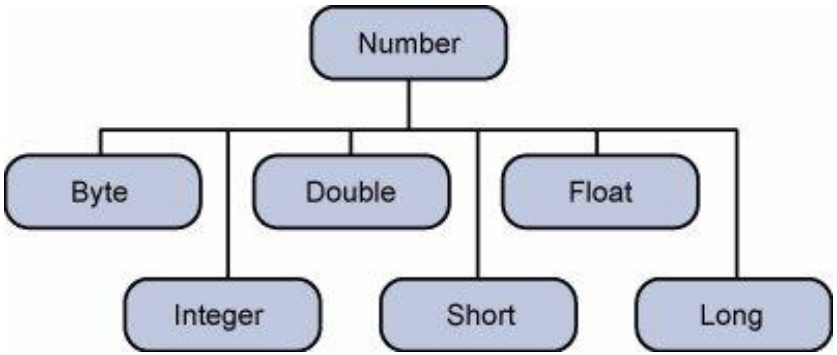
Example - for


```java
// Java program to illustrate for loop.
class forLoopDemo
{
    public static void main(String args[])
    {
        // for loop begins when x=2
        // and runs till x <=4
        for (int x = 2; x <= 4; x++)
            System.out.println("Value of x:" + x);
    }
}
```

Example - do...while


```java
// Java program to illustrate do-while loop
class dowhileloopDemo
{
    public static void main(String args[])
    {
        int x = 21;
        do
        {
            // The line will be printed even
            // if the condition is false
            System.out.println("Value of x:" + x);
            x++;
        }
        while (x < 20);
    }
}
```

- Normally, when we work with Numbers, we use primitive data types such as byte, int, long, double, etc.
- However, in development, we come across situations where we need to use objects instead of primitive data types. In order to achieve this, Java provides wrapper classes.
- All the wrapper classes (Integer, Long, Byte, Double, Float, Short) are subclasses of the abstract class Number.



Sr.No.	Method & Description
1	xxxValue() : Converts the value of this Number object to the xxx data type and returns it.
2	compareTo() :Compares this Number object to the argument.
3	equals() : Determines whether this number object is equal to the argument.
4	valueOf() : Returns an Integer object holding the value of the specified primitive.
5	toString() : Returns a String object representing the value of a specified int or Integer.
6	parseInt() : is used to get the primitive data type of a certain String.
7	abs() : Returns the absolute value of the argument.
8	ceil() : Returns the smallest integer that is greater than or equal to the argument. Returned as a double.
9	floor() : Returns the largest integer that is less than or equal to the argument. Returned as a double.
10	rint() : Returns the integer that is closest in value to the argument. Returned as a double.
11	round() : Returns the closest long or int.
12	min() : Returns the smaller of the two arguments.

Sr.No.	Method & Description
13	max() : Returns the larger of the two arguments.
14	exp() : Returns the base of the natural logarithms i.e. e to the power of the argument.
15	log() : Returns the natural logarithm of the argument.
16	pow() : Returns the value of the first argument raised to the power of the second argument.
17	sqrt() : Returns the square root of the argument.
18	sin() : Returns the sine of the specified double value.
19	cos() : Returns the cosine of the specified double value.
20	tan() : Returns the tangent of the specified double value.
21	asin() : Returns the arcsine of the specified double value.
22	acos() : Returns the arccosine of the specified double value.
23	atan() : Returns the arctangent of the specified double value.
24	atan2() : Converts rectangular coordinates (x, y) to polar coordinate (r, theta) and returns theta.
25	toDegrees() : Converts the argument to degrees.
26	toRadians() : Converts the argument to radians.
27	random() : Returns a random number.

```
public class Test {

   public static void main(String args[]) {
      Integer x = 5, y=10, z=5;
      Short a = 5;

      // Returns byte primitive data type
      System.out.println( x.byteValue() );

      // Returns double primitive data type
      System.out.println(x.doubleValue());

      // Returns long primitive data type
      System.out.println( x.longValue() );

      System.out.println(x.compareTo(3));
      System.out.println(x.compareTo(5));
      System.out.println(x.compareTo(8));
```

```
System.out.println(x.equals(y));
System.out.println(x.equals(z));
System.out.println(x.equals(a));

Integer x =Integer.valueOf(9);
Double c = Double.valueOf(5);
Float a = Float.valueOf("80");
Integer b = Integer.valueOf("444",16);

System.out.println(x);
System.out.println(c);
System.out.println(a);
System.out.println(b);

double x = 11.635;
double y = 2.76;

System.out.printf("The value of e is %.4f%n", Math.E);
System.out.printf("log(%.3f) is %.3f%n", x, Math.log(x));
```

```
        System.out.printf("The value of e is %.4f%n", Math.E);
        System.out.printf("exp(%.3f) is %.3f%n", x, Math.exp(x));

        System.out.println(Math.min(12.123, 12.456));
        System.out.println(Math.min(23.12, 23.0));

        System.out.println(Math.max(12.123, 12.456));
        System.out.println(Math.max(23.12, 23.0));

        double d = -100.675;
        float f = -90;

        System.out.println(Math.ceil(d));          // prints -100.0
        System.out.println(Math.ceil(f));          // prints -90.0

        System.out.println(Math.floor(d));          // prints -101.0
        System.out.println(Math.floor(f));          // prints -90.0

    }
}
```

- Normally, when we work with characters, we use primitive data types char.

Example:

```
char ch = 'a';
        // Unicode for uppercase Greek omega character
char uniChar = '\u039A';
char[] charArray ={ 'a', 'b', 'c', 'd', 'e' }; // an array of chars
```

- However in development, a situation demands to use objects instead of primitive data types.
- Java provides wrapper class **Character** for primitive data type **char**.
- The **Character** class offers a number of useful class (i.e., static) methods for manipulating characters. You can create a Character object with the Character constructor.

Example-1 :

```
Character ch = new Character('a');
```

Example-2:

```
Character ch = 'a';
char c = test('x');
```

A character preceded by a backslash (\) is an escape sequence and has a special meaning to the compiler.

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

Sr.No.	Method & Description
1	isLetter() : Determines whether the specified char value is a letter.
2	isDigit() : Determines whether the specified char value is a digit.
3	isWhitespace() : Determines whether the specified char value is white space.
4	isUpperCase() : Determines whether the specified char value is uppercase.
5	isLowerCase() : Determines whether the specified char value is lowercase.
6	toUpperCase() : Returns the uppercase form of the specified char value.
7	toLowerCase() : Returns the lowercase form of the specified char value.
8	toString() : Returns a String object representing the specified character value that is, a one-character string.

- Strings are widely used in Java programming,
- Strings are a sequence of characters.
- In Java, Strings are treated as objects.
- The String class is immutable, so that once it is created a String object cannot be changed.
- Creating a String :-

```
String greeting = "Hello world!";
```

Example :

```
public class StringDemo {

   public static void main(String args[]) {
      char[] helloArray = { 'h', 'e', 'l', 'l', 'o', '.' };
      String helloString = new String(helloArray);
      System.out.println( helloString );
   }
}
```

String Methods

Sr.No.	Method & Description
1	char charAt(int index) Returns the character at the specified index.
2	int compareTo(Object o) Compares this String to another Object.
3	int compareTo(String anotherString) Compares two strings lexicographically.
4	int compareToIgnoreCase(String str) Compares two strings lexicographically, ignoring case differences.
5	String concat(String str) Concatenates the specified string to the end of this string.
6	boolean contentEquals(StringBuffer sb) Returns true if and only if this String represents the same sequence of characters as the specified StringBuffer.
7	static String copyValueOf(char[] data) Returns a String that represents the character sequence in the array specified.
8	static String copyValueOf(char[] data, int offset, int count) Returns a String that represents the character sequence in the array specified.
9	boolean endsWith(String suffix) Tests if this string ends with the specified suffix.
10	boolean equals(Object anObject) Compares this string to the specified object.

Sr. No	Method and Description
11	boolean isEmpty() checks if string is empty.
12	String concat(String str) concatenates the specified string.
13	String replace(char old, char new) replaces all occurrences of the specified char value.
14	String replace(CharSequence old, CharSequence new) replaces all occurrences of the specified CharSequence.
15	static String equalsIgnoreCase(String another) compares another string. It doesn't check case.
16	String[] split(String regex) returns a split string matching regex.
17	String[] split(String regex, int limit) returns a split string matching regex and limit.
18	String intern() returns an interned string.
19	int indexOf(int ch) returns the specified char value index.
20	int indexOf(int ch, int fromIndex) returns the specified char value index starting with given index.

Sr. No	Method and Description
21	int indexOf(String substring) returns the specified substring index.
22	int indexOf(String substring, int fromIndex) returns the specified substring index starting with given index.
23	String toLowerCase() / toUpperCase() returns a string in lowercase/uppercase
24	String toLowerCase(Locale l) / toUpperCase(Locale l) returns a string in lowercase/uppercase using specified locale.
25	String trim() removes beginning and ending spaces of this string.
26	static String valueOf(int value) converts given type into string. It is an overloaded method.

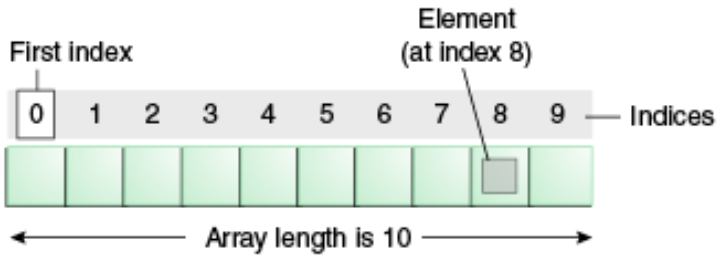
```java
// Program to Demonstrate String Mehtods

public class StringDemo {

        public static void main(String args[]){
              String str = "Little Star";
              String anotherString = "little star";
              Object objStr = str;

              System.out.println( str.compareTo(anotherString) );
              System.out.println( str.compareToIgnoreCase(anotherString) );
              System.out.println( str.compareTo(objStr.toString()));

              String s1 = "MyJavatutorial";
              String s2 = "MyJavatutorial";
              String s3 = new String ("My Tutorials Point");
              System.out.println(s1.equals(s2));
              System.out.println(s2.equals(s3));

              System.out.println(s1 == s2);
              System.out.println(s2 == s3);
```

Example : String Methods....! II


```
String strOrg = "Hello world ,Hello Reader";
int lastIndex = strOrg.lastIndexOf("Hello");

if(lastIndex == - 1){
   System.out.println("Hello not found");
} else {
   System.out.println("Last occurrence of Hello is at index "+ las
}

String t1 = "MyTutorialpoint";
int index = t1.lastIndexOf("p");
System.out.println(index);

String string = "abcdef";
String reverse = new StringBuffer(string).reverse().toString();
System.out.println("\nString before reverse: "+string);
System.out.println("String after reverse: "+reverse);

String strOrig = "Hello readers";
int intIndex = strOrig.indexOf("readers");
```

```
        if(intIndex == - 1) {
            System.out.println("Hello not found");
        } else {
            System.out.println("Found Hello at index " + intIndex);
        }

        String text = "The cat is on the table";
        System.out.print(text.contains("the"));

        String str1 = "string abc touppercase ";
        String strUpper = str1.toUpperCase();
        System.out.println("Original String: " + str1);
        System.out.println("String changed to upper case: " + strUpper);
    }

}
```

- An array is a collection of similar type(Homegeneous) of elements which has contiguous memory location.
- Java array is an object which contains elements of a similar data type.
- The elements of an array are stored in a contiguous memory location.
- It is a data structure where can store similar elements.
- It is static.
- It is index-based, the first element of the array is stored at the 0^{th}index, 2^{nd} element is stored on 1^{st} index and so on.
- In Java, array is an object of a dynamically generated class.
- Java array inherits the Object class, and implements the Serializable as well as Cloneable interfaces.



Advantages :

- Code Optimization: It makes the code optimized, we can retrieve or sort the data efficiently.
- Random access: We can get any data located at an index position.

Disadvantages :

- Size Limit: We can store only the fixed size of elements in the array. It doesn't grow its size at runtime.
  To solve this problem, collection framework is used in Java which grows automatically.

Types of Array in java

- Single Dimensional Array
- Multidimensional Array

Single Dimensional Array in Java

Syntax :

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

Instantiation : `arrayRefVar=new datatype[size];`

Declaration, Instantiation and Initialization of Java Array

```
int a[]={33,3,4,5};

        (or)

a[0] = 33;        a[1] = 2;
a[2] = 4;        a[3] = 5;
```

Multidimensional Array in Java

Data is stored in row and column based index
(also known as matrix form).

Syntax to Declare Multidimensional Array in Java

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

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

Example to initialize Multidimensional Array in Java

```
arr[0][0]=1;        arr[0][1]=2;
arr[0][2]=3;        arr[1][0]=4;
arr[1][1]=5;        arr[1][2]=6;
arr[2][0]=7;        arr[2][1]=8;
arr[2][2]=9;
```

```java
//Program to copy all elements of one array into another array

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

        int [] arr1 = new int [] {1, 2, 3, 4, 5};
        int arr2[] = new int[arr1.length];

        for (int i = 0; i < arr1.length; i++) {   arr2[i] = arr1[i];   }

        //Displaying elements of array arr1
        System.out.println("Elements of original array: ");
        for (int i = 0; i < arr1.length; i++) {
           System.out.print(arr1[i] + " ");
        }

        System.out.println();

        //Displaying elements of array arr2
        System.out.println("Elements of new array: ");
        for (int i = 0; i < arr2.length; i++) {
           System.out.print(arr2[i] + " ");
        }
    }
}
```

```java
// To print the elements of an array present on even odd position

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

        //Initialize array
        int [] arr = new int [] {1, 2, 3, 4, 5};

        System.out.println("Elements at even position: ");

        for (int i = 1; i < arr.length; i = i+2) {
            System.out.println(arr[i]);
        }

        System.out.println("Elements at odd position: ");

        for (int i = 0; i < arr.length; i = i+2) {
            System.out.println(arr[i]);
        }
    }
}
```

```java
// Program to sort the elements of an array in ascending order

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

      int [] arr = new int [] {5, 2, 8, 7, 1};      int temp = 0;

      System.out.println("Elements of original array: ");
      for (int i = 0; i < arr.length; i++) {  System.out.print(arr[i] + " "); }

      //Sort the array in ascending order
      for (int i = 0; i < arr.length; i++)
          for (int j = i+1; j < arr.length; j++) {
             if(arr[i] > arr[j]) {
                 temp = arr[i];   arr[i] = arr[j];   arr[j] = temp;
          }
      }

      System.out.println("Elements of array sorted in ascending order: ");
      for (int i = 0; i < arr.length; i++) {  System.out.print(arr[i] + " "); }
}
```

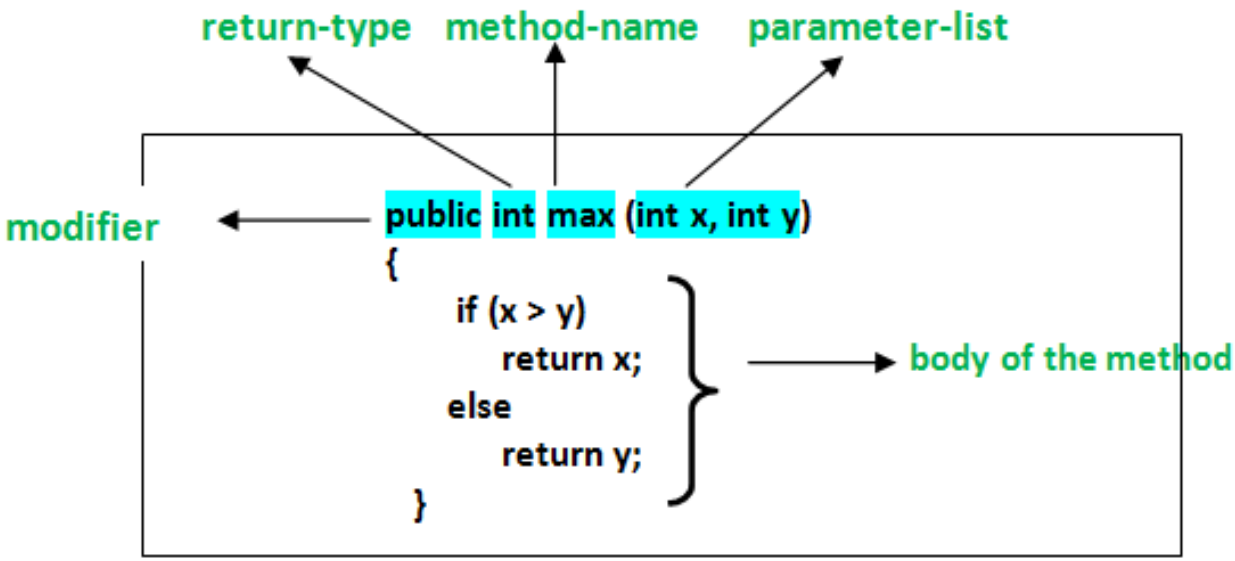
```java
// Program to print number of Characters in a string

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

        String string = "The best of both worlds";
        int count = 0;

        //Counts each character except space
        for(int i = 0; i < string.length(); i++) {
            if(string.charAt(i) != ' ')
                count++;
        }

        //Displays the total number of characters present in the given string
        System.out.println("Total number of characters in a string: " + count);
    }
}
```

```java
// Program to print number of Vowels, Consonant in a string

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

        int vCount = 0, cCount = 0;
        String str = "This is a really simple sentence";
        str = str.toLowerCase();

        for(int i = 0; i < str.length(); i++) {

            if(str.charAt(i) == 'a' || str.charAt(i) == 'e' ||
               str.charAt(i) == 'i' || str.charAt(i) == 'o' ||
               str.charAt(i) == 'u') {
                  vCount++;    //Increments the vowel counter
            }
            //Checks whether a character is a consonant
            else if(str.charAt(i) >= 'a' && str.charAt(i)<='z') {
                cCount++;    //Increments the vowel counter
            }
        }
        System.out.println("Number of vowels: " + vCount);
        System.out.println("Number of consonants: " + cCount);
    }
}
```

List of Programs to practice

- Java Program to Fibonacci Series in Java
- Java Program to Prime Number Program in Java
- Java Program to Palindrome Program in Java
- Java Program to Factorial Program in Java
- Java Program to find Reverse of the string
- Java program to find the duplicate characters in a string
- Java program to find the duplicate words in a string
- Java Program to find the frequency of characters
- Java Program to find the largest and smallest word in a string
- Java Program to find the most repeated word in a text file
- Java Program to find the number of the words in the given text file
- Java Program to separate the Individual Characters from a String
- Java Program to swap two string variables without using third or temp variable.
- Java Program to print smallest and biggest possible palindrome word in a given string

- A method is a block of code which only runs when it is called.
- Can pass data, known as parameters, into a method.
- Methods are used to perform certain actions, and they are also known as functions.
- Why use methods? To reuse code: define the code once, and use it many times.



Example of Java Method

Create a Method :

- A method must be declared within a class.
- It is defined with the name of the method, followed by parentheses ().
- Java provides some pre-defined methods, such as System.out.println(), but you can also create your own methods to perform certain actions:

Call a Method :

- To call a method in Java, write the method's name followed by two parentheses () and a semicolon;
- In the following example, myMethod() is used to print a text (the action), when it is called:

```
public class Main {
  static void myMethod() {
        System.out.println("I just got executed!");
  }

  public static void main(String[] args) {
        myMethod();         // I just got executed!
        myMethod();         // I just got executed!
  }
}
```

- Information can be passed to methods as parameter. Parameters act as variables inside the method.
- Parameters are specified after the method name, inside the parentheses. And can add multiple parameters, just separate them with a comma.

```
// Single and Multiple Parameters

public class Main {
  static void myMethod(String fname) {
        System.out.println(fname);
  }
  static void myMethod1(String fname, int age) {
        System.out.println(fname + " is " + age);
  }

  public static void main(String[] args) {
        myMethod("Pradeep");    myMethod("Nachiyappan");
        myMethod("Tulasi");

        myMethod1("Seeta", 15); myMethod1("Geeta", 18);
        myMethod1("Meeta", 31);
  }
}
```

```java
// Multiple Parameters
public class Main {
  static int myMethod(int x, int y) { return x + y;   }
    public static void main(String[] args) {
                System.out.println(myMethod(5, 3));
    }
}
```

```java
// Single Parameters
public class Main {
  static void checkAge(int age) {
    if (age < 18) {
                System.out.println("Access denied - You are not old enough!");
        }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
  }
}
```

Variables declared directly inside a method are available anywhere in the method following the line of code in which they were declared.

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

        // Code here CANNOT use x

        int x = 100;

        // Code here can use x
        System.out.println(x);
  }
}
```

- **Block Scope** : It refers to all of the code between curly braces {}.
- Variables declared inside blocks of code are only accessible within the curly braces.
- Variables declared directly inside a method are available anywhere in the method following the line of code in which they were declared.
- *A block of code may exist on its own or it can belong to an if, while or for statement. In the case of for statements, variables declared in the statement itself are also available inside the block's scope.*

```
public class Main {
  public static void main(String[] args) {
        // Code here CANNOT use x

        { // This is a block
          // Code here CANNOT use x

          int x = 100;

          // Code here CAN use x
          System.out.println(x);
        } // The block ends here

  // Code here CANNOT use x
  }
}
```

- It is the technique of making a function call itself. It provides a way to break complicated problems down into simple problems which are easier to solve.
- It may be a bit difficult to understand. The best way to figure out how it works is to experiment with it.

```
// Recursion Example

        public class Main {

          public static void main(String[] args) {
                int result = sum(10);
                System.out.println(result);
          }

          public static int sum(int k) {
                if (k > 0) {
                  return k + sum(k - 1);
                } else {
                  return 0;
                }
          }
        }
```


Thank you!