ITM (SLS) Baroda University

Second Year

Object-Oriented Programming with Java

Subject Teacher

Dr. K. V. Metre

- Teaching Scheme:
- Theory (TH): 03hrs/Week
- Practical : 4 hrs/week
- Credit Scheme: 05
- Total Marks : 150

Examination Marks

Theory Marks		Practical marks	
External	Internal	External	Internal
40	60	0	50

Contents

Basics of Java:

- Features of Java, Byte Code and Java Virtual Machine, JDK
- Data types, Operator,
- Control Statements –

  If , else, nested if, if-else ladders, Switch,

  while, do-while, for, for-each, break, continue.

Array and String:

- Single and Multidimensional Array,
- String class, StringBuffer class, Operations on string,
- Command line argument,
- Use of Wrapper Class.

Introduction

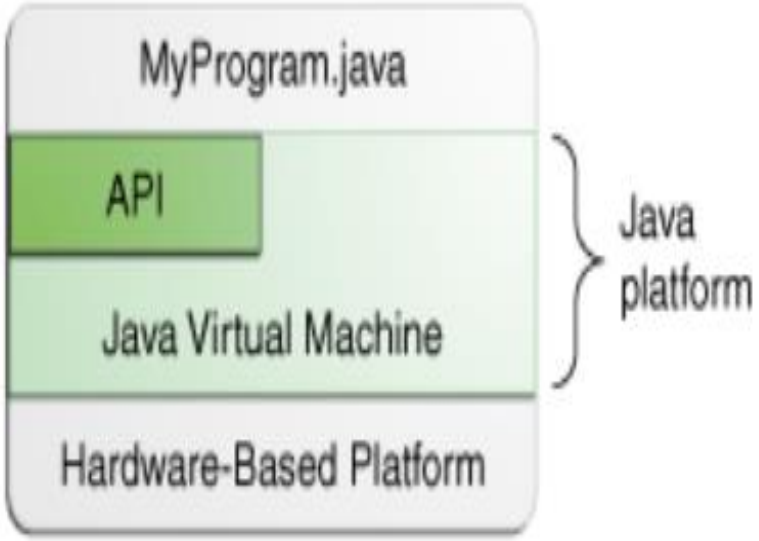
- Java was developed by James Gosling in 1991 at Sun Microsystems which is now subset of Oracle.

- The language was initially called ‘Oak’, but was renamed as ‘Java’ in 1995.

- Java is both programming language and a platform.
- A platform is a hardware or software where the programs run.

- Most platforms can be described as a combination of the operating system and underlying hardware. The Java platform differs from most other platforms in that it's a software-only platform that runs on top of other hardware-based platforms.
- Java platform has two components:
  - Java Virtual Machine(JVM)
  - Application Programming Interface

Introduction



The API and Java Virtual Machine insulate the program from the underlying hardware.

Uses of java

- It is used for developing Android Apps
- Helps you to create Enterprise Software
- Wide range of Mobile java Applications
- Scientific Computing Applications
- Use for Big Data Analytics
- Used for Server-Side Technologies like Apache, JBoss, GlassFish, etc.

POP v/s OOP

Procedural Oriented Programming	Object Oriented Programming
In POP, program is divided into small parts called **functions**.	In OOP, program is divided into parts called **objects**.
In POP,Importance is not given to **data** but to functions as well as **sequence** of actions to be done.	In OOP, Importance is given to the data rather than procedures or functions because it works as a **real world**.
POP follows **Top Down approach**.	OOP follows **Bottom Up approach**.
POP does not have any access specifier.	OOP has access specifiers named Public, Private, Protected, etc.
In POP, Data can move freely from function to function in the system.	In OOP, objects can move and communicate with each other through member functions.

POP v/s OOP

To add new data and function in POP is not so easy.	OOP provides an easy way to add new data and function.
In POP, Most function uses Global data for sharing that can be accessed freely from function to function in the system.	In OOP, data can not move easily from function to function, it can be kept public or private so we can control the access of data.
POP does not have any proper way for hiding data so it is less secure.	OOP provides Data Hiding so provides more security.
In POP, Overloading is not possible.	In OOP, overloading is possible in the form of Function Overloading and Operator Overloading.
Example of POP are : C, VB, FORTRAN, Pascal.	Example of OOP are : C++, JAVA, VB.NET, C#.NET.

Components of Java Programming

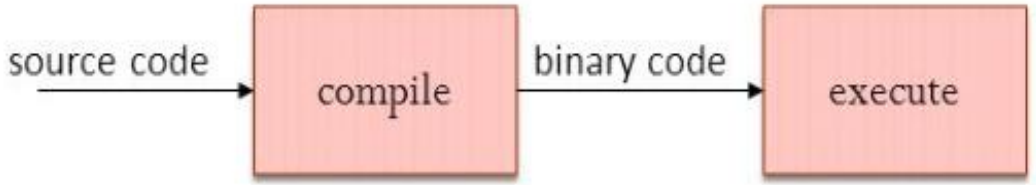
To convert the source code to machine code

Java uses three components:

1. Java Development Kit
2. Java Runtime Environment
3. Java Virtual Machine

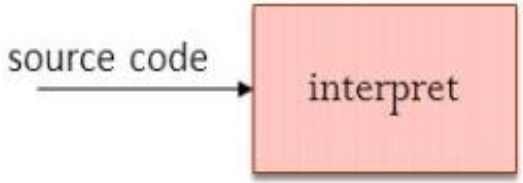
Compilers, Interpreters, and the JVM

Compiled Languages (e.g. C, C++)



Compiler is unique to each platform

Interpreted Languages (e.g. JavaScript, Perl, Ruby)

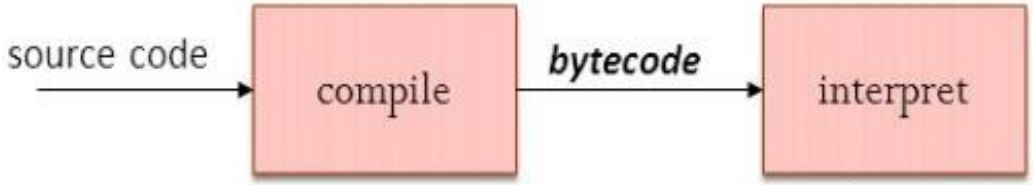


Interpreter translates code into binary and executes it

Small, easy to write

Interpreter is unique to each platform

Java

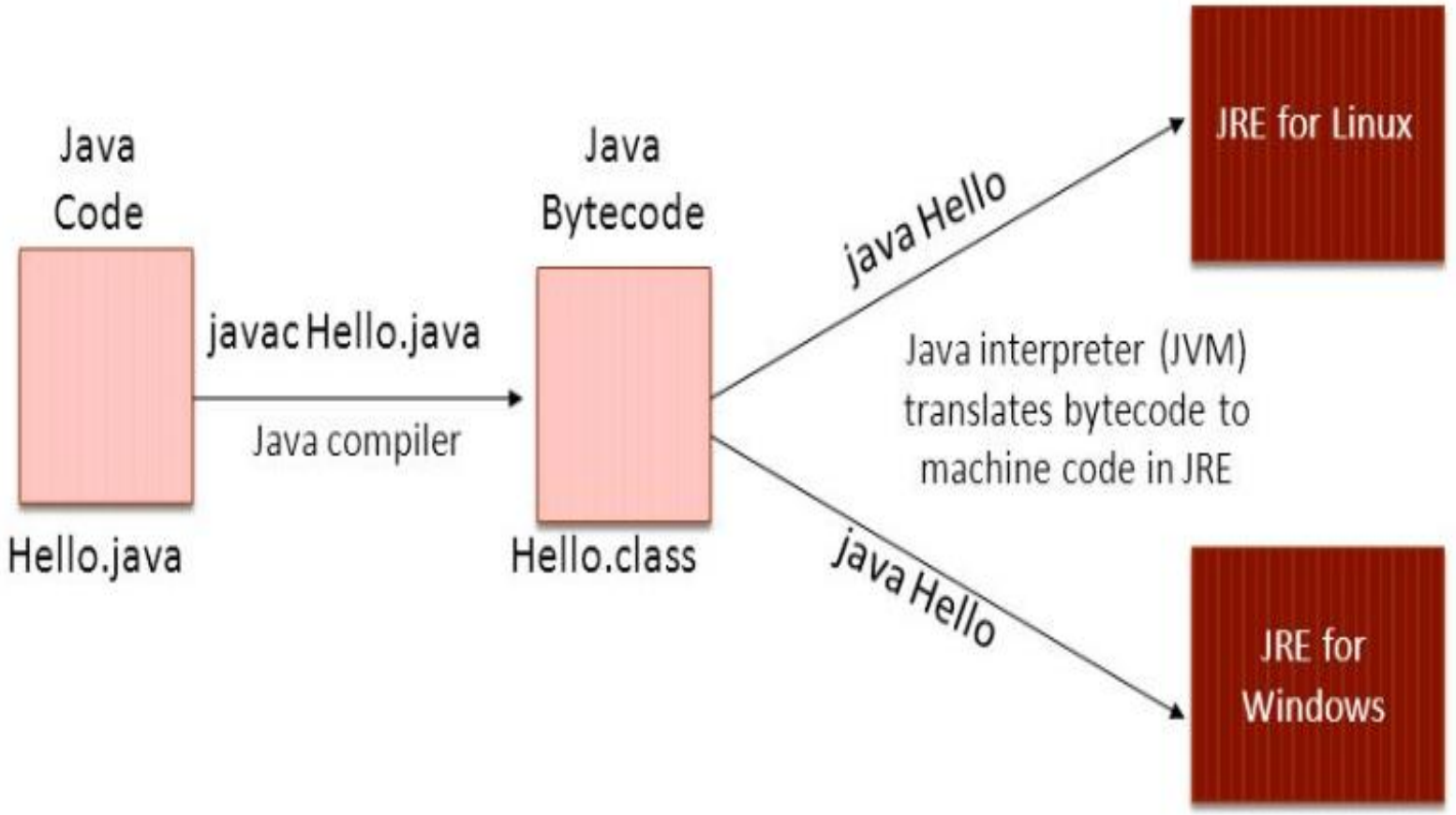


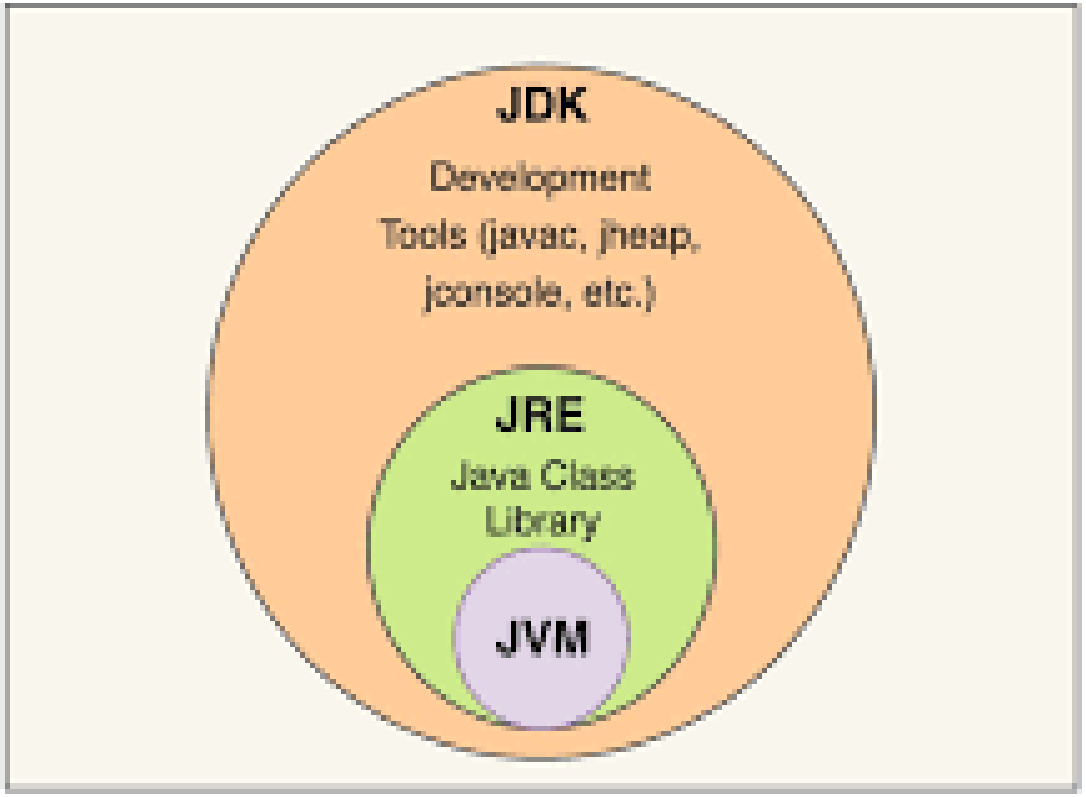
Java Virtual Machine (JVM)

Bytecode is platform independent

JVM is unique to each platform

Compiling and Running Java




JDK
Development
Tools (javac, jheap,
jconsole, etc.)
JRE
Java Class
Library
JVM

Java Development kit

- JDK is a software development environment used for making java applications & applets.
- It is possible to install more than one JDK version on the same computer.
- JDK contains tools required to write Java program & JRE to execute them.
- It includes compiler, Java application launcher, Appletviewer etc.

NOTE: Compiler converts source code to the bytecode & App Launcher opens JRE, loads the necessary class & execute its main method

Java Runtime Environment

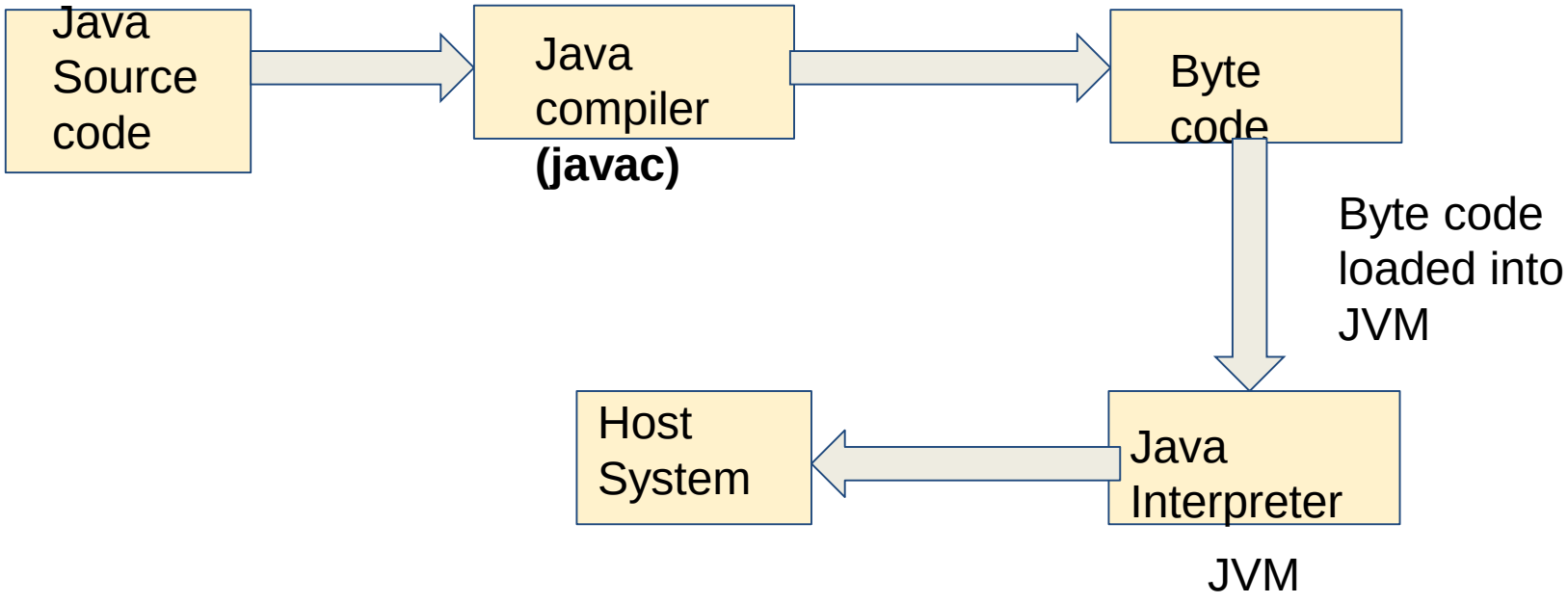
- JRE is a piece of software that is designed to run other software.
- It contains the class libraries, loader class & JVM.
- JRE does not include any tool for java development like debugger, compiler etc.
- It uses important package classes like math, swing, util, lang, awt & runtime libraries.

NOTE: For developing the java programs we need JDK but to run java program only JRE is needed

Java Virtual Machine

- JVM is an engine that provides a runtime environment to drive a java code or application. It is a part of JRE which converts **bytecode** to **machine language.**
- JVM provides the platform independent way to execute java source code.

.NOTE: Although JVM provides platform independent way to execute source code but itself is platform dependent. It also allocate & deallocate the memory space


Java Source code
Java compiler (javac)
Byte code
Byte code loaded into JVM
Java Interpreter
JVM
Host System

Phases of Program Creation and Execution

- **Edit :**
  - Creating a Java program consists of using an editor program.
  - Java source code files are saved with the file extension "**.java**".
  - Source files can be created using a simple text editor, or an IDE (Integrated Development Environment), such as JCreator, Eclipse, JBuilder, etc.
  - IDEs provide tools to support the development process, including editors for writing programs and debugging for locating logic errors in programs.

Phases of Program Creation and Execution

- **Compile**
  - During this phase, the programmer compiles the program using a command at the command line, or tools from the IDE.
  - At this step, the Java source code is translated into **bytecodes**.
  - If you are running the Java compiler from the command prompt, the command to enter is:
    - **Command: javac Hello.java***
  - Running this command would compile the Java source file, Hello.java, and generate a bytecode class file named, Hello.class.
  - Compiler is available as part of the Java Development Kit (JDK)

Phases of Program Creation and Execution

- **Load**
  - The program must be placed in memory before it can execute.
  - In loading, the class loader takes the ".class" files, created in the previous step, and transfers them to primary memory.
  - The class loader also loads the .class files provided by Java, that your program uses.

JRE / JDK

- **JRE**: Java Runtime Environment. It is basically the Java Virtual Machine where your Java programs run on. It also includes browser plugins for Applet execution.

- **JDK**: It's the full featured Software Development Kit for Java, including **JRE**, and the compilers and tools (like JavaDoc, and Java Debugger) to create and compile programs.

- Usually, when you only care about running Java programs on your browser or computer you will only install **JRE**. It's all you need. On the other hand, if you are planning to do some Java programming, you will also need **JDK**.

JDK / JRE File Structure

- **c:\jdk1.7.0** Root directory of the JDK software installation. Contains copyright, license, and README files. Also contains src.zip, the archive of source code for the Java platform.

- **c:\jdk1.7.0\bin** Executable files for the development tools contained in the Java Development Kit. The **PATH** environment variable should contain an entry for this directory.

- **c:\jdk1.7.0\jre** Root directory of the Java runtime environment used by the JDK development tools. The runtime environment is an implementation of the Java platform.

- **c:\jdk1.7.0\jre\bin** Executable files and DLLs for tools and libraries used by the Java platform.

- **c:\jdk1.7.0\jre\lib** Code libraries, property settings, and resource files used by the Java runtime environment.

Getting started

- In order to get started in Java programming, one needs to get a recent copy of the Java JDK.
- This can be obtained for free by downloading it from the Sun Microsystems website, http://java.sun.com/
- Once you download and install this JDK, you need a text editor to write your source file.
- Save the file as Hello.java
- The name of the program has to be similar to the filename.
- Java is case-sensitive: You cannot name a file "Hello.java" and then in the program you write "public class hello".

Getting started

- In order to get the output we have to first compile the program and then execute the compiled class.
- The applications required for this job are available as part of the JDK:
  - javac.exe – compiles the program
  - java.exe – the interpreter used to execute the compiled program
- In order to compile and execute the program we need to switch to the command prompt.
- On windows systems, this can be done by clicking *Start>Run>cmd*

Java Platforms

- J2ME(Micro Edition):

  Specifies several different sets of libraries (known as profiles) for devices with limited storage, display, and power capacities. Often used to develop applications for mobile devices, PDAs, TV set-top boxes, and printers.

- J2EE(Enterprise Edition):

  Java SE plus various APIs useful for multi-tier client–server enterprise applications.

- J2SE(Standard Edition):

  For general-purpose use on desktop PCs, servers and similar devices.

- Java FX:

  JavaFX is a platform for developing rich internet applications using a lightweight user-interface API.

My First program in Java

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

My First program in Java

- **Understanding first java program**
  - **class** keyword is used to declare a class in java.
  - **public** keyword is an access modifier which represents visibility, it means it is visible to all.
  - **static** is a keyword, if we declare any method as static, it is known as static method. The core advantage of static method is that there is no need to create object to invoke the static method. So it saves memory.
  - **void** is the return type of the method, it means it doesn't return any value.
  - **main** represents startup of the program.
  - **String[] args** is used for command line argument.
  - **System.out.println()** is used print statement.

Naming convention

- **Case Sensitivity**
  - Java is case sensitive
  - Example: **Hello** and **hello** would have different meaning in Java.

- **Class Names**
  - For all class names the first letter should be in Upper Case.
  - If several words are used to form a name of the class, each inner word's first letter should be in Upper Case.
  - Example *class MyFirstJavaClass*

Naming convention

- **Method Names**
  - All method names should start with a Lower Case letter.
  - If several words are used to form the name of the method, then each inner word's first letter should be in Upper Case.
  - Example *public void myMethodName()*

Naming convention

- **Program File Name**
  - Name of the program file should exactly match the class name.
  - When saving the file, you should save it using the class name and append '.java' to the end of the name.
  - If the file name and the class name do not match your program will not compile
  - Example : Assume 'MyFirstJavaProgram' is the class name. Then the file should be saved as 'MyFirstJavaProgram.java'

Java Identifiers

- All Java components require names. Names used for classes, variables and methods are called **identifiers**.
- In Java, there are several points to remember about identifiers.
  - All identifiers should begin with a letter (A to Z or a to z), currency character ($) or an underscore (_).
  - After the first character identifiers can have any combination of characters.
  - A key word cannot be used as an identifier.
  - Identifiers are case sensitive.
  - Examples of legal identifiers: age, $salary, _value, __1_value
  - Examples of illegal identifiers: 123abc, -salary

Java Keywords

- These reserved words may not be used as constant or variable or any other identifier names.

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

Comments in Java

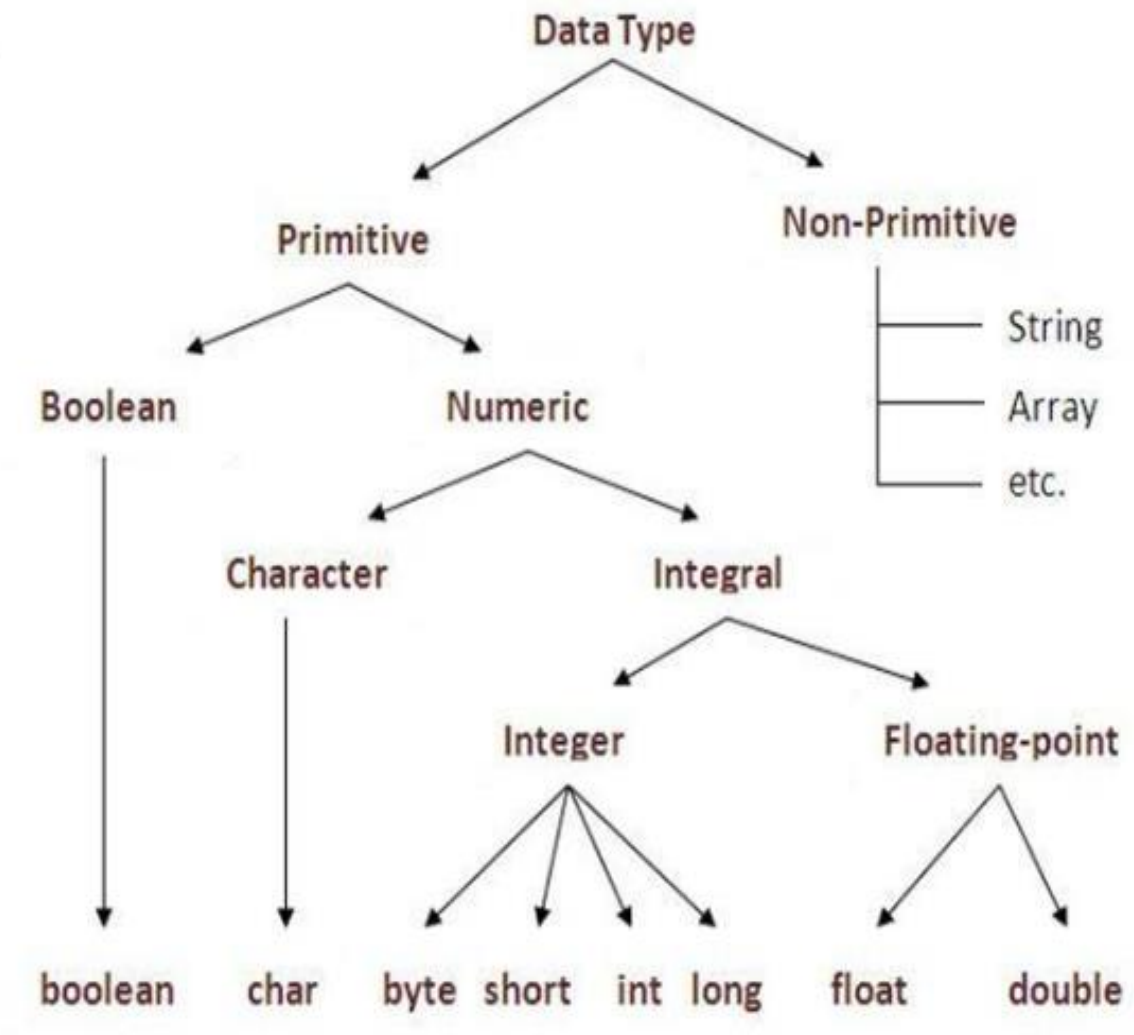
- Java supports single-line and multi-line comments very similar to c and c++.
- All characters available inside any comment are ignored by Java compiler.

```
public class MyFirstJavaProgram{
 /*This is my first java program.
*This will print 'Hello World' as the output
*This is an example of multi-line comments. */
 public static void main(String []args){
 //This is an example of single line comment
/*This is also an example of single line comment. */
 System.out.println("Hello World");
 }
 }
```

- A line containing only whitespace, possibly with a comment, is known as a **blank line**, and Java totally ignores it.

Java: Datatypes

- There are two data
  - Primitive Data Typ
  - Reference/Object



Java: Datatypes

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

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

➔Because java uses **unicode system** rather than ASCII code system.
\u0000 is the lowest range of unicode system.

Primitive Data Types

Data type	Range of values
byte	-128 .. 127 (8 bits)
short	-32,768 .. 32,767 (16 bits)
int	-2,147,483,648 .. 2,147,483,647 (32 bits)
long	-9,223,372,036,854,775,808 .. ... (64 bits)
float	$+/-10^{-38}$ to $+/-10^{+38}$ and 0, about 6 digits precision 4 Byte
double	1.7e-308 to 1.7e+308 and 0, about 15 digits precision 8 bytes
char	Unicode characters (generally 16 bits per char)
boolean	True or false 1 byte

Java: Datatypes

- **Examples:**

```
int a, b, c;  // Declares three ints, a, b, and c.
int a = 10, b = 10;  // Example of initialization
byte B = 22; // initializes a byte type variable B.
double pi = 3.14159; // declares and assigns a value of PI.
char a = 'a'; // the char variable a is initialized with value 'a'
```

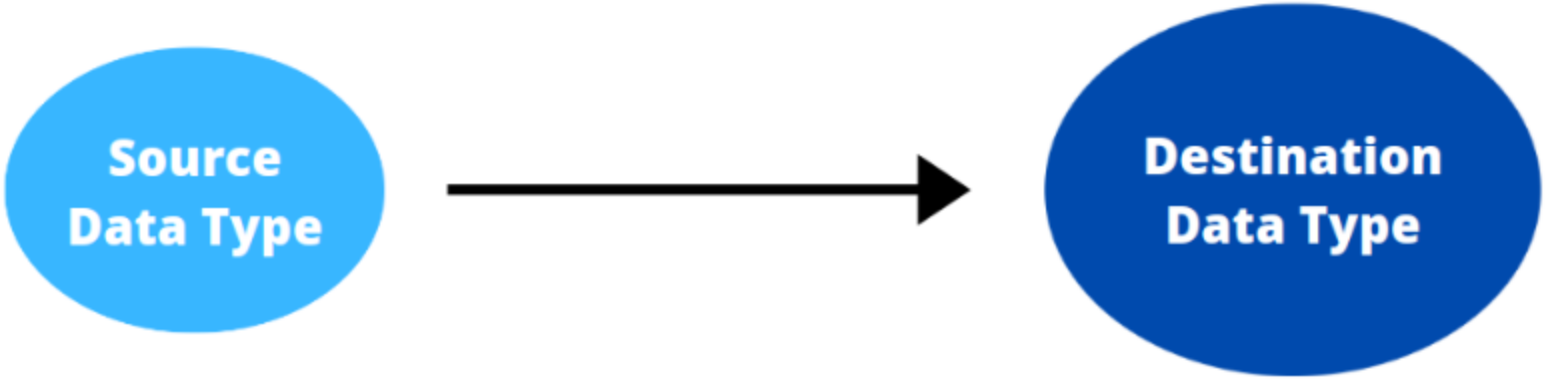
Type Conversion & Type Casting

- A significant part of programming involves playing around with data.
- Sometimes data type has to be changed from one type to another.
- Calculating the area of a circle using Pi, we get a decimal number, and that needs to be converted to integer if we desire to obtain the answer in integer format.
- Since Java is strongly typed, one often needs to convert from one data type to another. The programmer or the compiler can easily do this by using type casting and type conversion concepts.

Type Conversion

- **Type conversion** is a process in which the data type is automatically converted into another data type.
- The compiler does this automatic conversion at compile time.
- For type conversion to take place, the destination data type must be larger than the source type.
- This type of type conversion is also called **Widening Type Conversion/ Implicit Conversion/ Casting Down.**
- In this case, as the lower data types with smaller sizes are converted into higher ones with a larger size, there is no chance of data loss.

Type Conversion



Flow chart for Type Conversion

Type Conversion

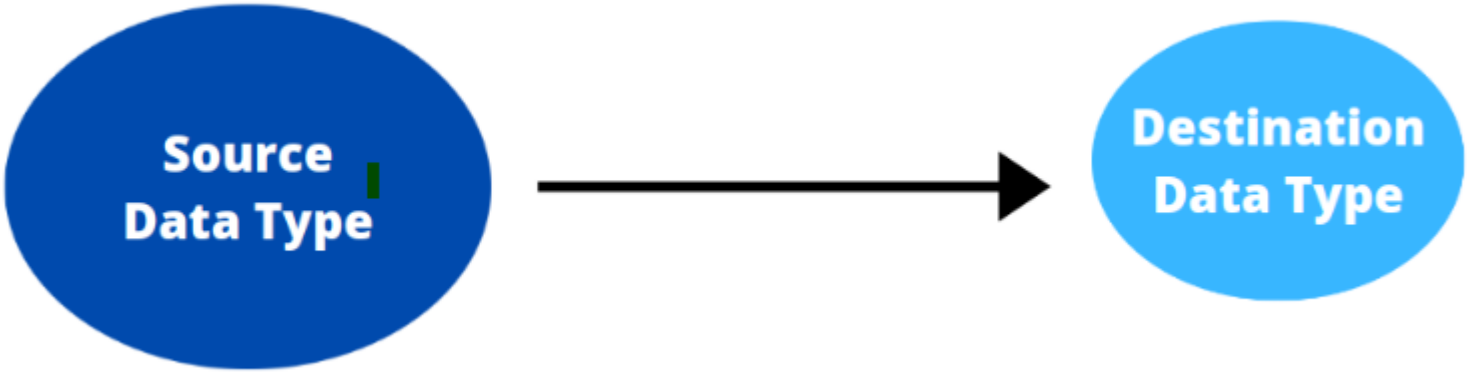
```
class Main {
        public static void main(String[] args) {
                int a= 10;
                float b=a;
                System.out.println( “value of a after type conversion:” + b);
    }
}
```

```
PS C:\Users\HP\Desktop\itm\JAVA\practicals_2023> javac Main.java
PS C:\Users\HP\Desktop\itm\JAVA\practicals_2023> java Main
value of a after type conversion:10.0
```

Type Casting

- **Type casting** is a process in which the programmer manually converts one data type into another data type.
- For this the casting operator (), the parenthesis is used. Unlike type conversion, the source data type must be larger than the destination type in type casting. The below flow chart has to be followed for successful type casting.
- Type casting is also called **Narrowing Type Casting/ Explicit Conversion/ Casting Up**. In this case, as the higher data types with a larger size are converted into lower ones with a smaller size, there is a chance of data loss. This is the reason that this type of conversion does not happen automatically.

Type Casting



Flow chart for Type Casting

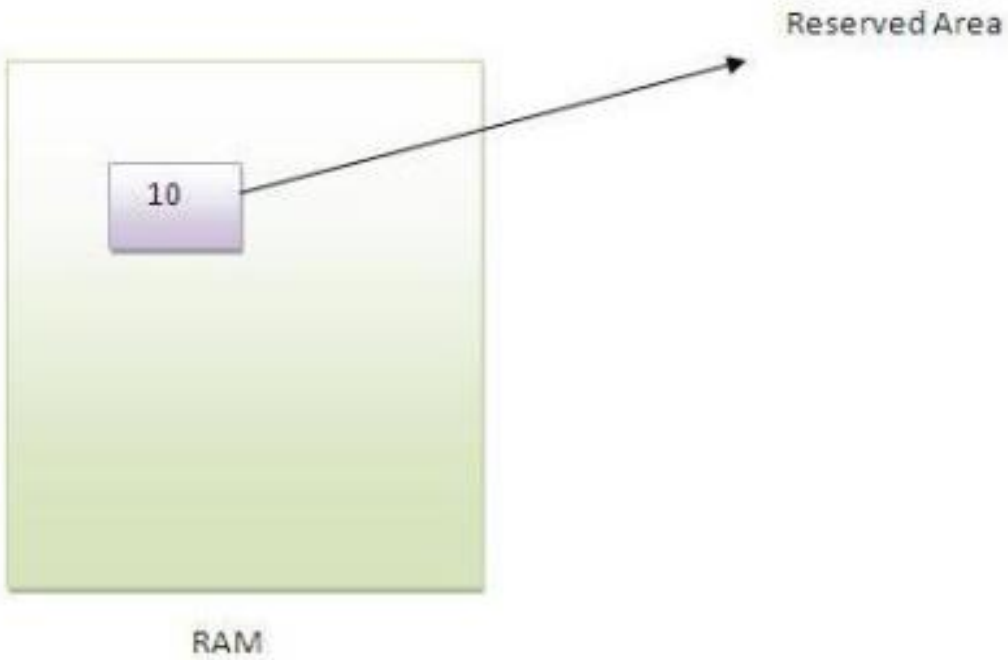
Type Casting

```
class Main {
        public static void main(String[] args) {
                float a= 10.56f;
                int b=a;
        System.out.println( “value of a after type casting:” + b);
    }
}
```

```
PS C:\Users\HP\Desktop\itm\JAVA\practicals_2023> javac Main.java
PS C:\Users\HP\Desktop\itm\JAVA\practicals_2023> java Main
value of a after type casting:10
```

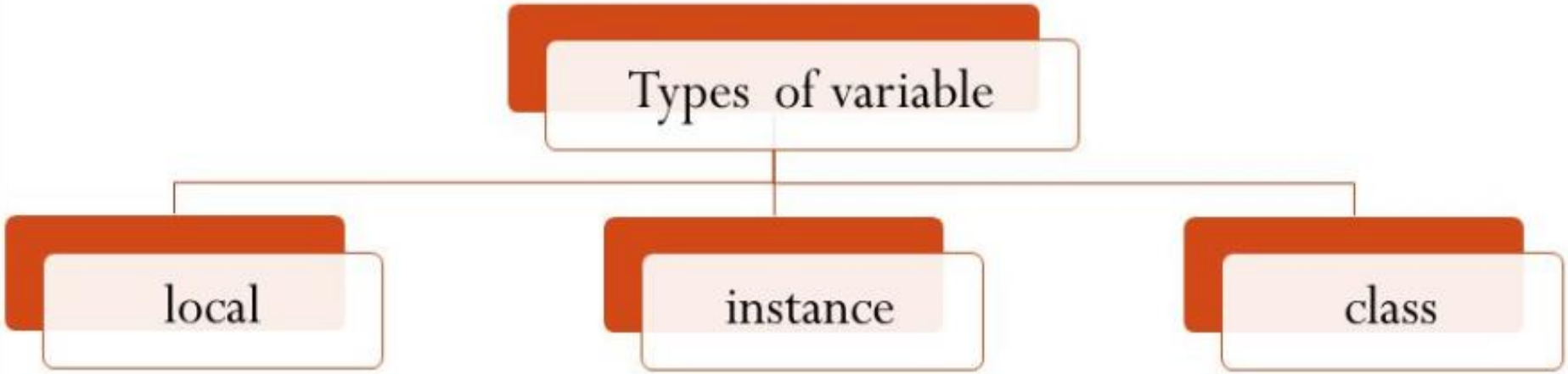
Java: Variables

- Variable is name of reserved area allocated in memory.



- int data=50;//Here data is variable

Java: Variables



A variable that is declared inside the method is called local variable.

A variable that is declared inside the class but outside the method is called instance variable . It is not declared as static.

A variable that is declared as static is called static (also class) variable. It cannot be local.

Java: Variables

Example to understand the types of variables:

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

Constant in Java : literals

```
public class Test {
public static void main(String[] args)
{
    final int a=10;
    System.out.println(a);
    a=33; //error as a is defined final
  System.out.println(a);
}
}
```

Java : Operators

- Java provides a rich set of operators to manipulate variables:
  - Arithmetic Operators
  - Relational Operators
  - Bitwise Operators
  - Logical Operators
  - Assignment Operators
  - Conditional Operator

Java : Operators

- **Arithmetic Operators:**

Arithmetic operators are used in mathematical expressions in the same way that they are used in algebra.

Operator	Description
+	Addition - Adds values on either side of the operator
-	Subtraction - Subtracts right hand operand from left hand operand
*	Multiplication - Multiplies values on either side of the operator
/	Division - Divides left hand operand by right hand operand
%	Modulus - Divides left hand operand by right hand operand and returns remainder
++	Increment - Increases the value of operand by 1
--	Decrement - Decreases the value of operand by 1

Java : Operators

- **Relational Operators:**

There are following relational operators supported by Java language

Operator	Description
==	Checks if the values of two operands are equal or not, if yes then condition becomes true.
!=	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.

Java : Operators

- **The Bitwise Operators:**
  - Java defines several bitwise operators, which can be applied to the integer types, long, int, short, char, and byte.
  - Bitwise operator works on bits and performs bit-by-bit operation.
  - Assume if a = 60; and b = 13; now in binary format :
    - a = 0011 1100
    - b = 0000 1101

    - a&b = 0000 1100
    - a|b = 0011 1101
    - a^b = 0011 0001
    - ~a = 1100 0011

Java : Operators

- **Logical Operators:**

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

Java : Operators

- **The Assignment Operators:**

Operator	Description
=	Simple assignment operator, Assigns values from right side operands to left side operand
+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand
*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand
/=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand
%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand

Operators

1. subscript **[ ]**, call **( )**, member access **.**
2. complement **~**, unary **+ -**,
3. type cast **(type)**,
4. binary **+ -** (**+** also concatenates strings)
5. class test **instanceof**

Java : Operators

- **Conditional Operator (? :)**
  - Conditional operator is also known as the ternary operator.
  - The goal of the operator is to decide which value should be assigned to the variable.
  - Syntax:

    variable x = (expression) ? value *if true* : value *if false*

  - Example:

```
int  a , b;
a = 10;
 b = (a == 1) ? 20: 30;
System.out.println( "Value of b is : " + b );
 b = (a == 10) ? 20: 30;
System.out.println( "Value of b is : " + b );
```

➔ Value of b is : 30

Value of b is : 20

Control Statements / Structures

- Code execution from top to bottom.
- The statements in the code are executed according to the order in which they appear.
- Control structures can be used to control the flow of Java code.
- Decision Making statements
  - if statements
  - switch statement
- Loop statements
  - do while loop
  - while loop
  - for loop
  - for-each loop
- Jump statements
  - break statement
  - continue statement

Decision-Making statements:

- Decision-making statements evaluate the Boolean expression and control the program flow depending upon the result of the condition

Two types of decision-making statements

- If statement
- switch statement.

1) If Statement:

- In Java, the "if" statement is used to evaluate a condition. The control of the program is diverted depending upon the specific condition. The condition of the If statement gives a Boolean value, either true or false. In Java, there are four types of if-statements given below.
    - Simple if statement
    - if-else statement
    - if-else-if ladder
    - Nested if-statement

```
if(condition) {

    statement 1; //executes when condition is true
}
else  {
     statement 2; //executes when condition is false
}
```

```
if(condition 1) {
    statement 1; //executes when condition 1 is true
}
else if(condition 2) {
        statement 2; //executes when condition 2 is true
          }
     else {
    statement 2; //executes when all the conditions are false
}
```

```
public class Student {
public static void main(String[] args) {
Int  a = 10;
    if ( a < 100)
     {   if( a%2 == 0)
            {   System.out.println(" a is < 100 and even ");
            }
            else
               {   System.out.println("a is  < 100 and odd");
                    }
           else
    {   System.out.println("a is not < 100");
    }
} }
```

- Check the no if it is < 100 and even

```
if(condition 1) {
    statement 1; //executes when condition 1 is true
if(condition 2) {
    statement 2; //executes when condition 2 is true
    }
    else{
    statement 2; //executes when condition 2 is false
    }
}
```

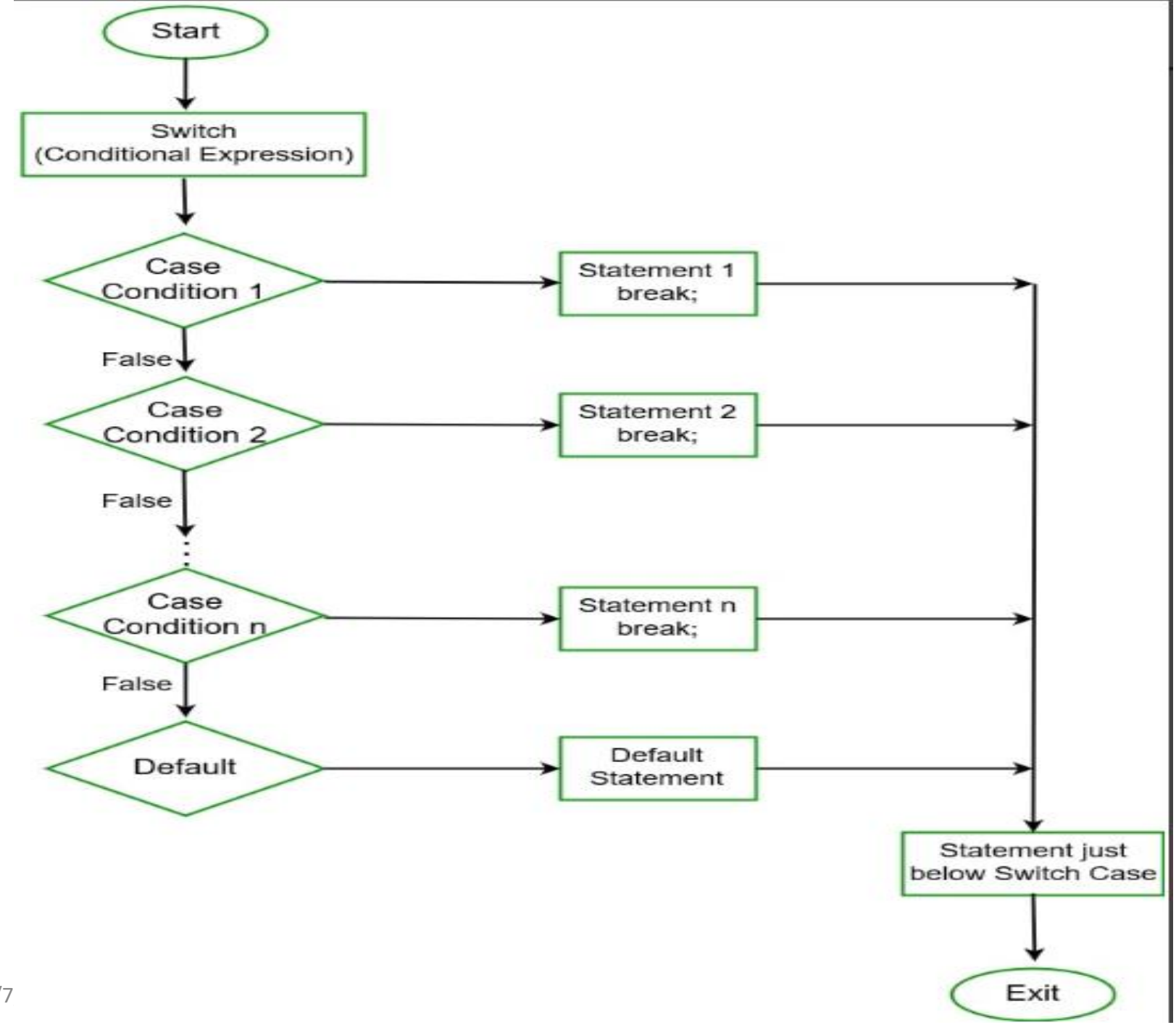
Switch Statement:

- are similar to if-else-if statements.
- The switch statement contains multiple blocks of code called cases and a single case is executed based on the variable which is being switched.

- The switch statement is easier to use instead of if-else-if statements. It also enhances the readability of the program.
- Each case value must be a unique literal or constant expression.
- **Points to be noted about switch statement:**

1) The case variables can be int, short, byte, char, or enumeration. String type is also supported since version 7 of Java.
2) Default statement is executed when any of the case doesn't match the value of expression. It is optional.
3) Break statement terminates the switch block when the condition is satisfied.
   It is optional, if not used, next case is executed.
4) While using switch statements, we must notice that the case expression will be of the same type as the variable. However, it will also be a constant value.

```
switch (choice)
{
  case 1:
            . . .
            break;
  case 2:
              . . .
              break;
  case 3:
               . . .
              break;
  case 4:
              . . .
              break;
  default:
            // bad input
              . . .
            break;
}
```


Start
Switch
(Conditional Expression)
Case
Condition 1
Statement 1
break;
False
Case
Condition 2
Statement 2
break;
False
Case
Condition n
Statement n
break;
False
Default
Default
Statement
Statement just
below Switch Case
Exit

• SWITCH STATEMENT IN JAVA WITH STRINGS

```
public class Main {
public static void main(String[] args) {
 String dayOfWeek = "Tuesday";
  switch (dayOfWeek) {
     case "Monday":
          System.out.println("Start of the work week");
               break;
    case "Tuesday":
          System.out.println("Time for meetings");
               break;
 case "Wednesday":
               System.out.println("Midweek work");
               break;
 case "Thursday":
 System.out.println("Almost there");
               break;
    case "Friday":
          System.out.println("End of the work week");
               break;
     case "Saturday":
     case "Sunday":
               System.out.println("Weekend!");
               break;
      default:

System.out.println("Invalid day");
        }
    }
}
```

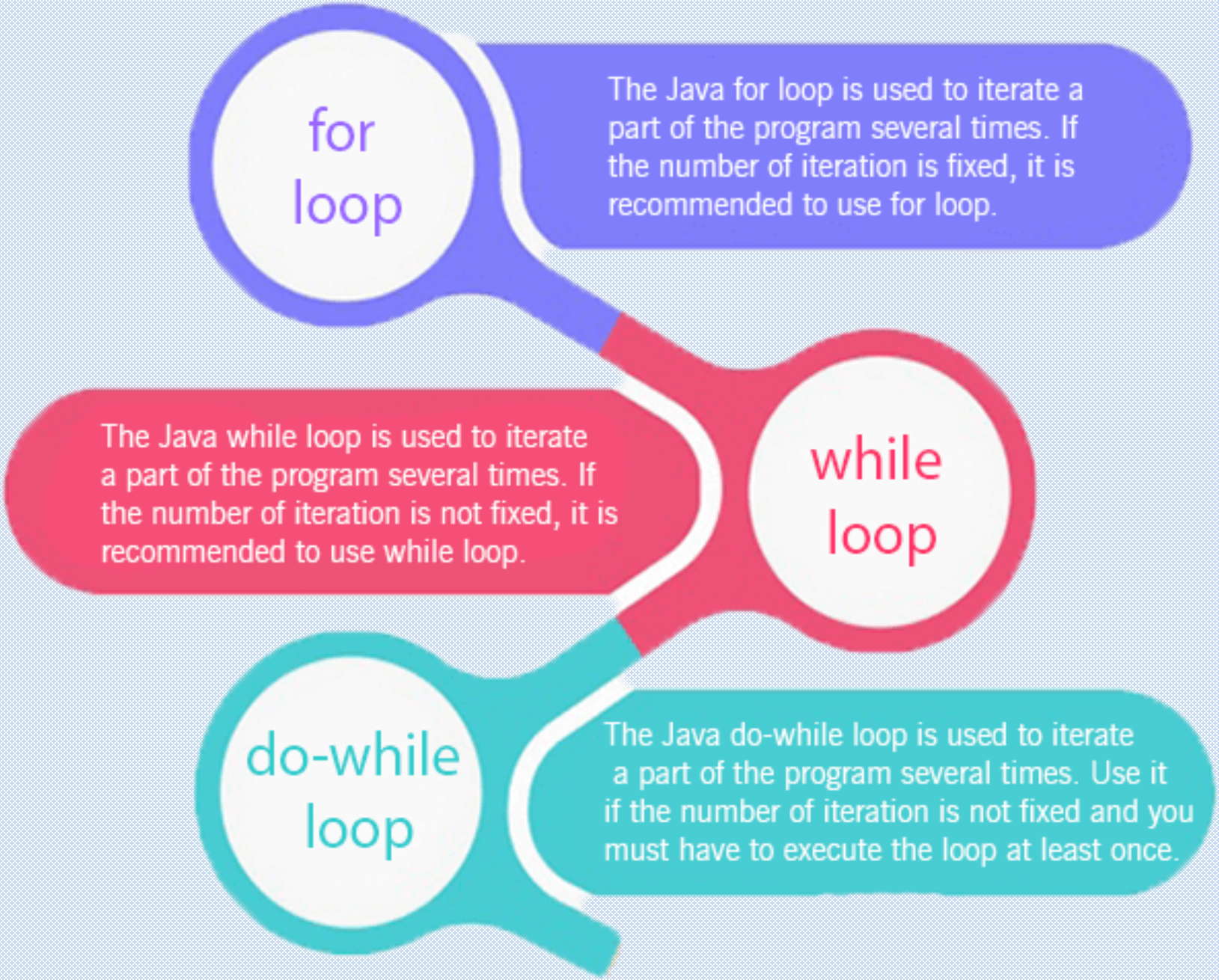
```
public class EnumSwitch{
enum Bikes { Honda, Pulsar,Suzuki; }
 public static void main(String args[])
   {  // Declaring the Enum variable
      Bikes b;
      b = Bikes.Pulsar;
     switch (b) {
       case Honda:
          System.out.println("You have chosen Honda !");
          break;
      case Pulsar:
          System.out.println(" You have chosen Pulsar !");
          break;
      case Suzuki:
          System.out.println(" You have chosen Suzuki !");
      default:
          System.out.println(" Oops ! Sorry not in the list. ");
          break;
      }
   }
}
```

Nested-Switch statements refers to Switch statements inside of another Switch Statements.

```
public class Main {
 public static void main (String[] args)
   {  int x = 1;       char y = 'b';

    switch (x)  {    // Outer Switch

          case 1:                      // If x == 1
               switch (y) {                      // Nested Switch

             case 'a':                        // If y == a
                System.out.println("Choice is 1-a");
                  break;
             case 'b':       // If y == b
                   System.out.println("Choice is 1-b");
                    break;
                   }
        break;
         case 2:       // If x == 2
                   System.out.println("Choice is 2");
                   break;
          case 3:      // If x == 3
                   System.out.println("Choice is 3");
                   break;
       default:
        System.out.println("Choice is other than 1, 2 3, ");
    } } }
```

Iteration Statement:

- The process of repeatedly executing a statements and is called as looping.
- The statements may be executed multiple times (from zero to infinite number).
- If a loop executing continuous then it is called as Infinite loop.
- Looping is also called as iterations.
- In Iteration statement, there are three types of operation:
- for loop, while loop, do-while loop


for loop
The Java for loop is used to iterate a part of the program several times. If the number of iteration is fixed, it is recommended to use for loop.
while loop
The Java while loop is used to iterate a part of the program several times. If the number of iteration is not fixed, it is recommended to use while loop.
do-while loop
The Java do-while loop is used to iterate a part of the program several times. Use it if the number of iteration is not fixed and you must have to execute the loop at least once.

- **while loop:**
- The while loop is entry controlled loop statement.
- The Java *while loop* is used to iterate a part of the program repeatedly until the specified Boolean condition is true.
- As soon as the Boolean condition becomes false, the loop automatically stops.

- Syntax:

```
Counter initialization;
While(condition/expression)
        { Statement block;
            Counter updation;
            }
```

```
public class WhileExample {
public static void main(String[] args) {
    int i=1;
    while(i<=10){
        System.out.println(i*i);
       i++;
    }
}
}
```

do-while loop:

- In do-while loop, first attempt of loop is to execute then it check the condition.
- The benefit of do-while loop/statement is that we get entry in loop and then condition will check for very first time.
- In while loop, condition will check first and if condition will not satisfied then the loop will not execute.

- Syntax:

```
do{
    //code to be executed / loop body
     //update statement
}while (condition/expression);
```

for loop:

- The for loop is entry controlled loop.
- It means that it provide a more conscious loop control structure.
- Syntax:

```
for (initialization; condition; iteration)

//iteration means increment/decrement
 { Statement block;
}
```

```
public class Main{

   public static boolean isPalindrome(String str)
{
   int n = str.length();

   // Compare characters from beginning and end
       for (int i = 0; i < n/2; i++)
              if (str.charAt(i) != str.charAt(n-i-1))
                  return false;
        return true;
}
public static void main(String args[])
{   System.out.println(isPalindrome("racecar"));
}
}
```

Java For-each Loop | Enhanced For Loop

- It is used to traverse the array or collection in Java.
- It is easier to use than simple for loop because we don't need to increment value and use subscript notation.
- It works on the basis of elements and not the index.
- It returns element one by one in the defined variable.
- It is known as the for-each loop because it traverses each element one by one.
- The drawback of the enhanced for loop is that it cannot traverse the elements in reverse order.
- Don’t have the option to skip any element because it does not work on an index basis.

For (data_type variable : arrayName | collection)

{ //body of for-each loop }

//An example of Java for-each loop

class ForEachExample{

public static void main(String args[]){

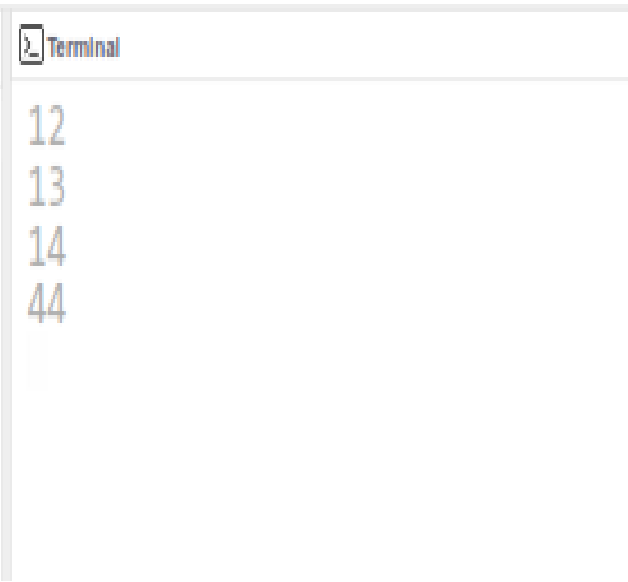
//declaring an array

int arr[]={12,13,14,44};

//traversing the array with for-each loop

for(**int** i:arr){

System.out.println(i);



```
public class ForEachExample{

public static void main(String args[]){

  //declaring an array
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
for (String i : cars) {
  System.out.println(i);
}
```

- Output :

```
Volvo
BMW
Ford
Mazda
```

Jump Statements

- Java supports three jump statements:
  - **break**
  - **continue**
  - **return.**

These statements transfer control to another part of the program.

Break Statement

- break can be used inside a loop to come out of it.
- break can be used inside the switch block to come out of the switch block.
- break can be used in nested blocks to go to the end of a block. Nested blocks represent a block written within another block.
- **Syntax:**

  break; (or)

  break label; //here label represents the name of the block.

Break: Example

```
class Forloop{

    public static void main(String args[]){
        int i;
        for(i=1;i<=10;i++){
            if(i==5)
                break;
            System.out.print(i+"\t");
        }
        System.out.println(x:"This method is after return");
    }
}
```

OUTPUT

```
PS C:\Users\HP\Desktop\itm\JAVA\practicals_2023> javac forloop.java
PS C:\Users\HP\Desktop\itm\JAVA\practicals_2023> java Forloop
1       2       3       4       This method is after return
```

Continue statement

- This statement is useful to continue the next repetition of a loop/ iteration.
- When continue is executed, subsequent statements inside the loop are not executed.

- **Syntax:** continue;

Continue: Example

```
class Forloop{

    public static void main(String args[]){
        int i;
        for(i=1;i<=10;i++){
            if(i==5)
                continue;
            System.out.print(i+"\t");
        }
    }
}
```

OUTPUT

```
PS C:\Users\HP\Desktop\itm\JAVA\practicals_2023> javac forloop.java
PS C:\Users\HP\Desktop\itm\JAVA\practicals_2023> java Forloop
1       2       3       4       6       7       8       9       10
```

Return statement

- return statement is useful to terminate a method and come back to the calling method.

- return statement in main method terminates the application.

- return statement can be used to return some value from a method to a calling method.

- **Syntax:**

  return; (or)

  return value; // value may be of any type

Return: Example

```
class Forloop{

    public static void main(String args[]){
        int i;
        for(i=1;i<=10;i++){
            if(i==5)
                return;
            System.out.print(i+"\t");
        }
        System.out.println(x:"This method is after return");
    }
}
```

OUTPUT

```
PS C:\Users\HP\Desktop\itm\JAVA\practicals_2023> javac forloop.java
PS C:\Users\HP\Desktop\itm\JAVA\practicals_2023> java Forloop
1       2       3       4
```

Array

- An array represents a group of elements of same data type.
- **Java array** is an object which contains elements of a similar data type
- Java array inherits the Object class

- Arrays are generally categorized into two types:
  - Single Dimensional arrays (or 1 Dimensional arrays)
  - Multi-Dimensional arrays (or 2 Dimensional arrays, 3 Dimensional arrays, …)

- Creating an array
  - Declare an array
  - Create memory location
  - Putting values to memory locations

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.

Declaring array variable

To declare an array, write the data type, followed by a set of square brackets[ ], followed by the identifier name.

```
datatype[ ] arrayName;  // preferred way.
                or
dataType arrayName[ ];  // works but not preferred way.
```

```
double[ ] myList;
        or
double myList[ ];
```

Array Creation

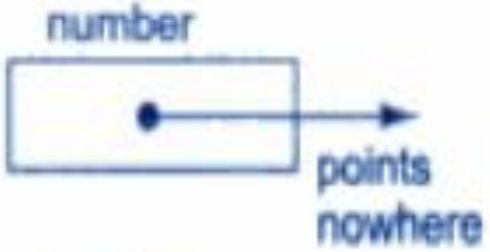
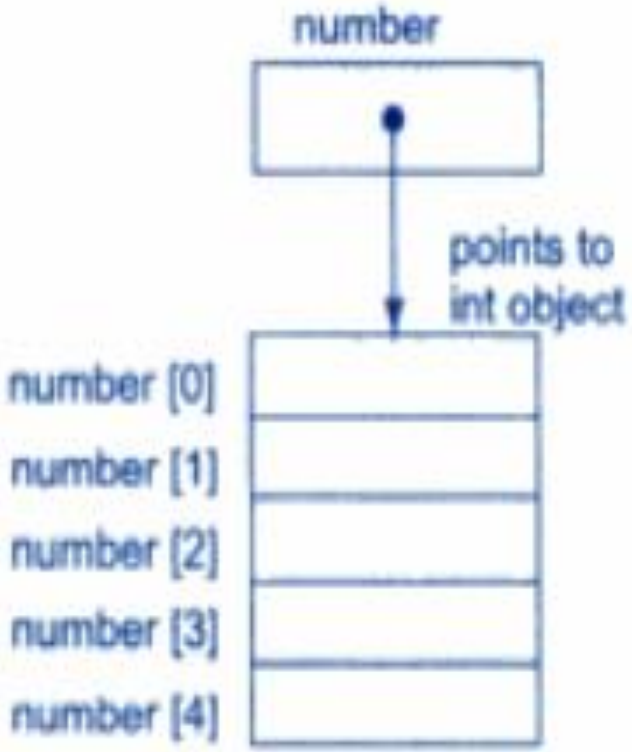
- To create array, write the **new** keyword, followed by the square[ ] brackets containing the number of elements in array.

For Example:

```
int arr[ ];          //declaration
arr[ ] = new int[10]; //object creation
```

Or

```
int arr[ ] = new int[10]; //declare & object creation
```

Statement	Result
`int number [ ];`	
`number = new int [5];`	

Array Creation

Another way to create array if the elements are known:

```
int arr[ ]={//array elements with comma operator };
```

Example:

```
int arr[] = {10,20,30,40,50};
```

Array Length

- to get the number of elements in an array, you can use the length field of an array.
- length field of an array returns the size of the array.
- Syntax:-

arrayName.length

```
public class Main{
      public static void main(String[] args) {
       int [] a={10,20,20};
       int x=a.length;
        System.out.println("length="  +x);
}}
```

Output :

length=3

Example 1

```
import java.util.Scanner;
class Arr{
    public static void main(String args[]){
        Scanner sc= new Scanner(System.in);
        int ar[]=new int[5];
        for(int i=0; i<ar.length;i++){
            ar[i] = sc.nextInt();
        }

        for(int i=0; i<ar.length;i++){
            System.out.print(ar[i]+"\t");
        }
    }
}
```

```
PS C:\Users\HP\Desktop\itm\JAVA\practicals_2023> javac Arr.java
PS C:\Users\HP\Desktop\itm\JAVA\practicals_2023> java Arr
10
20
30
40
50
10      20      30      40      50
```

Example 2

```
class Arr{

    public static void main(String args[]){
        int ar[]={10,20,30,40,50};

        for(int i=0; i<ar.length;i++){
            System.out.print(ar[i]+"\t");
        }
    }
}
```

```
PS C:\Users\HP\Desktop\itm\JAVA\practicals_2023> javac Arr.java
PS C:\Users\HP\Desktop\itm\JAVA\practicals_2023> java Arr
10      20      30      40      50
```

Multi-dimensional array

A multidimensional array is an array of arrays.

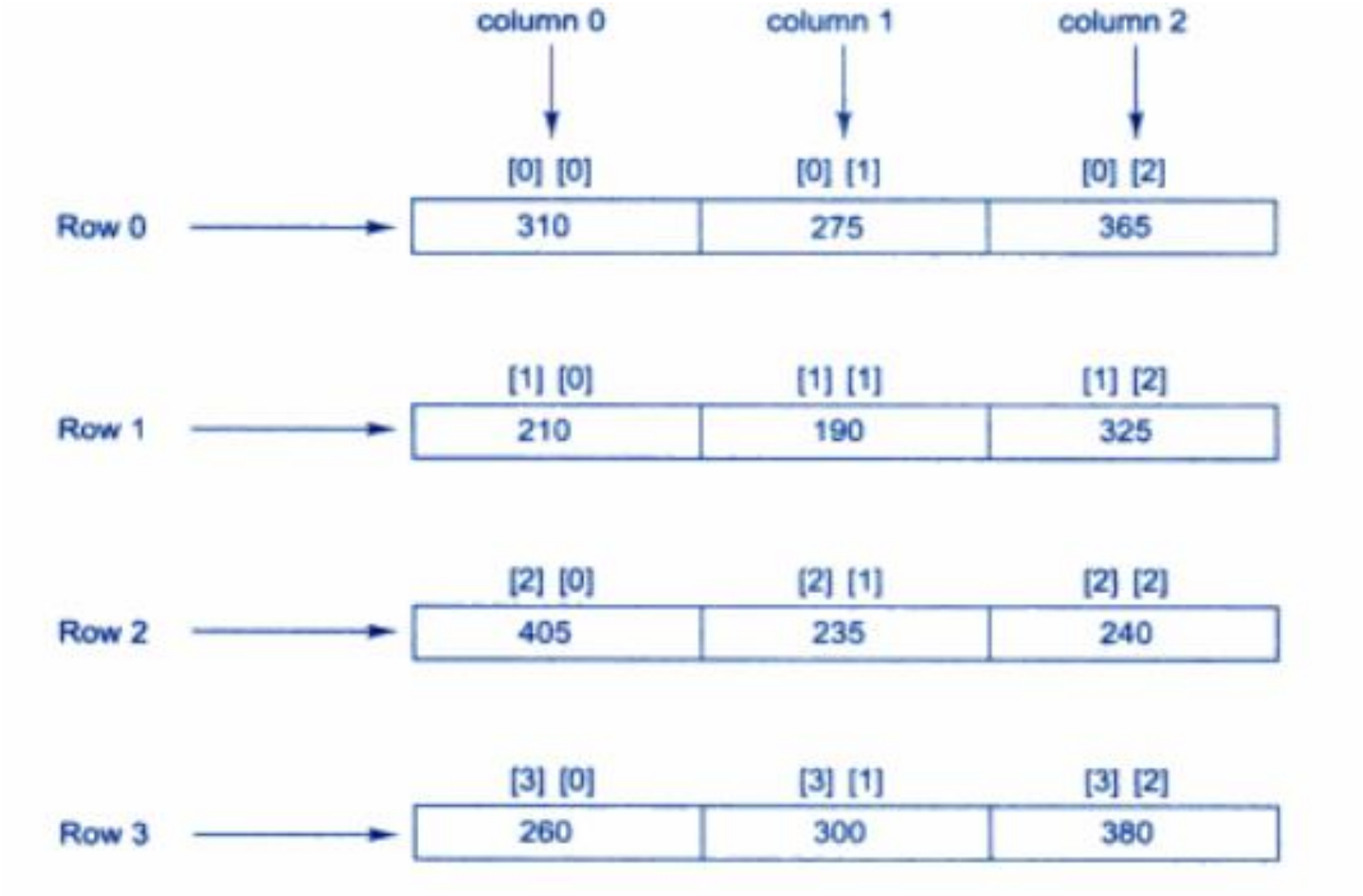
- dataType [][] arrayRefVar;
- dataType arrayRefVar[][];

- Each element of a multidimensional array is an array itself.

For example:

```
int[][] arr={{1,2,3},{4,5,6}}
```

To access the elements of the **arr** array, need to specify two indices: one for the array (row), and one for the element inside that array(column).


column 0
column 1
column 2
[0] [0]
[0] [1]
[0] [2]
Row 0
310
275
365
[1] [0]
[1] [1]
[1] [2]
Row 1
210
190
325
[2] [0]
[2] [1]
[2] [2]
Row 2
405
235
240
[3] [0]
[3] [1]
[3] [2]
Row 3
260
300
380

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


int x = a[i][j];

for ( int i=0; i<n ;i++)
{
    for (int j=0; j<n; j++)
    {
        a[i][j] = 0;
    }
}
```

Multi-dimensional array: using for loop

```
class Arr{

    public static void main(String args[]){

        int a[][]= {{1,2,3},{4,5,6}};

        for(int i=0;i<a.length;i++){
            for(int j=0;j<a[i].length;j++){

                System.out.print(a[i][j]+" ");
            }
            System.out.println(  " ");
        }
    }
}
```

```
PS C:\Users\HP\Desktop\itm\JAVA\practicals_2023> javac Arr.java
PS C:\Users\HP\Desktop\itm\JAVA\practicals_2023> java Arr
1 2 3
4 5 6
```

Multi-dimensional array: using for each loop

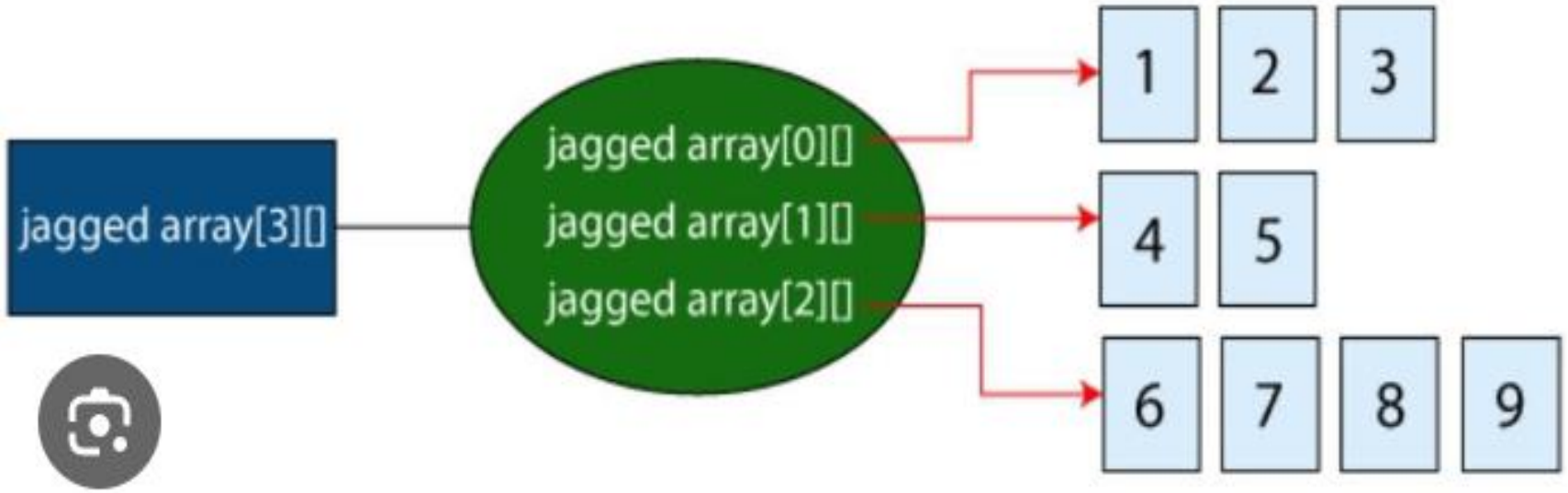
```
class Arr{
    public static void main(String args[]){

        int a[][]= {{1,2,3},{4,5,6}};

        for(int[] i:a){
            for(int j:i){

                System.out.print(j+"\t");
            }
            System.out.println(   "\n");
        }
    }
}
```

```
PS C:\Users\HP\Desktop\itm\JAVA\practicals_2023> javac Arr.java
PS C:\Users\HP\Desktop\itm\JAVA\practicals_2023> java Arr
1       2       3

4       5       6
```

OUTPUT

Jagged array/ Variable size array

Jagged array are the multidimensional arrays with different size



In Java, a jagged array can be declared and initialized using the following syntax:

Syntax:

```
datatype[][] arrayName = new datatype[numRows][];

arrayName[0] = new datatype[numColumns1];

arrayName[1] = new datatype[numColumns2];


arrayName[numRows-1] = new datatype[numColumnsN];

int[][] jaggedArray = { {1, 2}, {3, 4, 5}, {6, 7, 8, 9} };
```

Jagged array

```
import java.util.Scanner;
class Jaggedarray{

    public static void main(String args[]){
        Scanner sc= new Scanner(System.in);
        int a[][]= new int[3][];
        a[0]= new int[2];
        a[1]= new int[3];
        a[2]= new int[1];
        for(int i=0;i<a.length;i++){
            for(int j=0;j<a[i].length;j++){

                a[i][j]= sc.nextInt();
            }
        }
        System.out.println(x:"Jagged array is: ");

        for(int i=0;i<a.length;i++){
            for(int j=0;j<a[i].length;j++){

                System.out.print(a[i][j]+" ");
            }
```

OUTPUT

```
PS C:\Users\HP\Desktop\itm\JAVA\practicals_2023> javac Jaggedarray.java
PS C:\Users\HP\Desktop\itm\JAVA\practicals_2023> java Jaggedarray
10
2
1
3
4
3
Jagged array is:
10 2
1 3 4
3
```

String

- Strings are very important in the world of programming languages as they are used to process long textual/symbolic information.

- Strings are considered fundamental data types in almost all programming languages.
-
- These languages use character arrays to process long textual information.

- In java, string class is provided to work with strings along with the facility of character arrays.

- The strings are immutable as once an object of the String class is created & initialized with some value, it cannot be changed.

String

- To handle string data in Java, we require objects of the String class.
- The String class is included in the java.lang package.

How to create a string object?

There are two ways to create String object:
1) By string literal

Java String literal is created by using double quotes.

String s="welcome";

2) By new keyword

String s=**new** String("Welcome");

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

      String s1="java";//creating string by Java string literal

       char ch[]={'s','t','r','i','n','g','s'};
        String s2=new String(ch);//converting char array to string

        String s3=new String("example");//creating Java string by new keyword

        System.out.println(s1);

        System.out.println(s2);

        System.out.println(s3);
}}
```

- Output:

```
Java
Strings
example
```

Operation on String

The string class in java provides various methods

Convert a string into a character

1. Convert numbers into strings
2. Search strings
3. Create substrings
4. Change the case of string
5. Get a string length
6. Compare strings

1) int length() :

- It returns the length of the a string.
- This method calculates all the characters, symbols & numbers enclosed in double quotes (“”).
- The method returns an integer value

2) toLowerCase() :

- It returns a string after converting all uppercase letters of the invoking string into lowercase.

Syntax: stringname.toLowerCase();

3) toUpperCase() :

- It returns a string after converting all lowercase letters of the invoking string into Uppercase.

Syntax: stringname.toUpperCase()

4) substring():

- It returns a new String instance that is a substring of the original String instance.
- The substring() method is available in the following two forms:

1) substring(int begin)

- It takes a single parameter, which creates a new string that begins at the index specified by the argument value & ends at the last character of the string.

- **Example:**

```
String str = "EDUCATION"
system.out.println(str.substring(4));
```

Output :
ATION

2) substring(int begin, int end)

- This takes two parameters, the starting index and the ending index.
- The character at the start index is included in substring but the character at the end index is not.

Example:

```
String str = "EDUCATION"
System.out.println(str.substring(3,6));
```

Output:
CAT

5) concat() :

- Using addition (+) operator
- Using concat() method. str.concat(s1)

Example:

```
String str = “Hello”
String str1 = “Java”
System.out.println(str+str1);
System.out.println(str.concat(str1));
```

6) equals & equalsIgnoreCase() :

- These methods are used for comparing two strings.
- equals() compare the positions of the individual characters in a string.
- The equalsIgnoreCase() method does the same but ignores the case of characters
- Example:

```
String str = “Java is oopl”
String str1 = “JAVA IS OOPL”
System.out.println(str.equals(str1));
System.out.println(str.equalsIgnoreCase(str1));
```

Output:

false

true

```
class Stringop{

    public static void main(String[] args){
        String s= "Hello There";
        String str1="hello there";
        String str= ", how are you";
        System.out.println(s.length());
        System.out.println(s.toLowerCase());
        System.out.println(s.toUpperCase());
        System.out.println(s.substring(beginIndex:6));
        System.out.println(s.concat(str));
        System.out.println(s.length());
        System.out.println(s.equals(str1));
        System.out.println(s.equalsIgnoreCase(str1));
    }
}
```

OUTPUT

```
PS C:\Users\HP\Desktop\itm\JAVA\practicals_2023\string> javac Stringop.java
PS C:\Users\HP\Desktop\itm\JAVA\practicals_2023\string> java Stringop
11
hello there
HELLO THERE
There
Hello There, how are you
11
false
true
```

int indexOf(String substring)	It returns the specified substring index.
int indexOf(int ch)	It returns the specified char value index.
String replace(char old, char new)	It replaces all occurrences of the specified char value.
boolean isEmpty()	It checks if string is empty.

```
public class Main{
    public static void main(String []args){
        String s = new String("Welcome to java");
       System.out.println("Original String: " + s);
       System.out.println("Resulting String after replacing: " +
        s.toLowerCase());
        System.out.println("Original String: " + s);
 }}
```

Output :

Original String: Welcome to java
Resulting String after replacing: welcome to java
Original String: Welcome to java

Immutable String in Java :

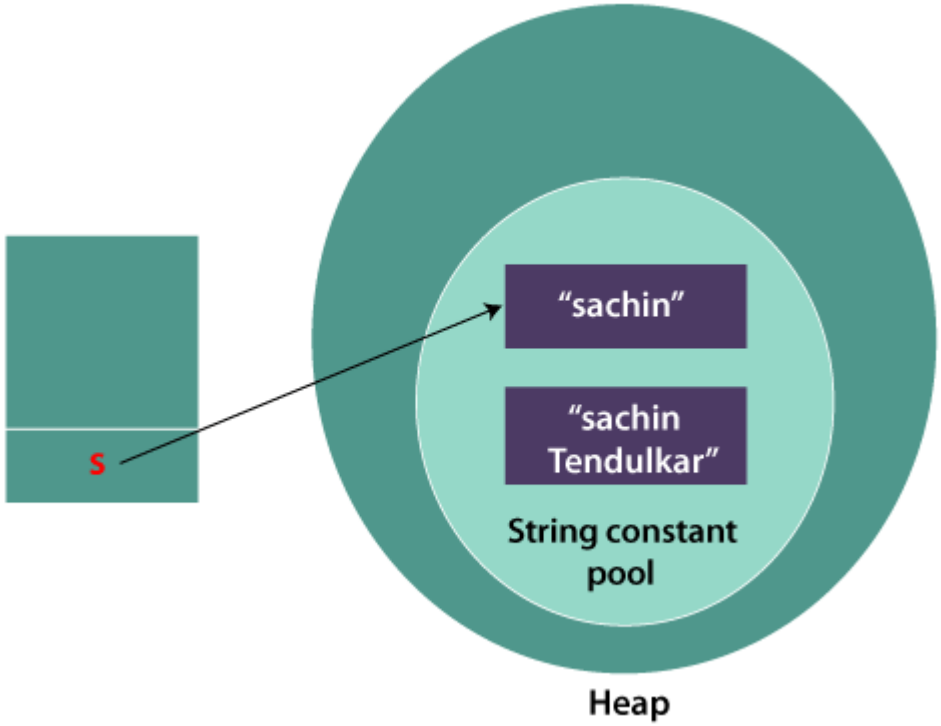
- **String objects are immutable**. Immutable simply means unmodifiable or unchangeable.

- Once String object is created its data or state can't be changed but a new String object is created.

```
class Testimmutablestring{
 public static void main(String args[]){
   String s="Sachin";
   s.concat("Tendulkar");//concat() method appends the string at the end
   System.out.println(s); //will print Sachin because strings are immutable objects
 }
}
```

Output :’

Sachin


"sachin"
"sachin Tendulkar"
String constant pool
s
Heap

Difference between String and StringBuffer

No	String	StringBuffer
1)	The String class is immutable.	The StringBuffer class is mutable.
2)	String is slow and consumes more memory when we concatenate too many strings because every time it creates new instance.	StringBuffer is fast and consumes less memory when we concatenate strings.
3)	String class overrides the equals() method of Object class. So you can compare the contents of two strings by equals() method.	StringBuffer class doesn't override the equals() method of Object class.
4)	String class is slower while performing concatenation operation.	StringBuffer class is faster while performing concatenation operation.
5)	String class uses String constant pool.	StringBuffer uses Heap memory

Stringbuffer class:

- StringBuffer class gives the facility to create a sequence of characters with mutable (modifiable) attributes.
- Syntax:

  StringBuffer str = new StringBuffer();
- The various StringBuffer methods are:
  1. length()
  2. capacity()
  3. reverse()
  4. append()
  5. insert()
  6. delete()
  7. deleteCharAt()
  8. replace()

Important Constructors of StringBuffer Class

Constructor	Description
StringBuffer()	It creates an empty String buffer with initial capacity of 16.
StringBuffer(String str)	It creates a String buffer with specified string..
StringBuffer(int capacity)	It creates an empty String buffer with specified capacity as length.

1) length() :

It defines the number of characters in the String.

Example:

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

      StringBuffer str = new StringBuffer("ContentWriter");
       int len = str.length();
       System.out.println("Length : " + len);
      }
      }
```

Output :

```
Length : 13
```

2) capacity()

- The capacity() method of the StringBuffer class returns the current capacity of the buffer.
- The default capacity of the buffer is 16.
- If the number of character increases from its current capacity, it increases the capacity by **(oldcapacity*2)+2.**
- if your current capacity is 16, it will be (16*2)+2=34.

2) capacity()

```
public class StringBufferExample{
   public static void main(String args[]){
       StringBuffer sb=new StringBuffer();
       System.out.println(sb.capacity());//default 16
       sb.append("Hello");
       System.out.println(sb);
       System.out.println(sb.capacity());//now 16
        sb.append("java is my favourite language");
       System.out.println(sb.capacity());//now (16*2)+2=34 i.e (oldcapacity*2)+2
       System.out.println(sb);
} }
```

Output :

```
16
Hello
16
34
Hellojava is my favourite language
```

3) append() :

The append() method is used to append the string or number at the end of the string.

Example:

```
StringBuffer str = new StringBuffer(“Hello");
str.append(“java");
System.out.println(str);
str.append(0);
System.out.println(str);
```

Output :

Hellojava

Hellojava0

4) insert() :

- insert() method is used to insert the string into the previously defined string at a particular indexed position.

- In insert method as a parameter, we provide **two-argument**, first is the **index** and second is the **string**.

```
public class HelloWorld{

    public static void main(String []args){

        StringBuffer s1 = new StringBuffer("Welcome");

         System.out.println(s1);

              s1.insert(7, " to java ");

                System.out.println(s1);

        s1.insert(3, " to java ");

        System.out.println(s1);     }}
```

Output :

Welcome

Welcome to java

Wel to java come to java

5)reverse() :

- reverse() method reverses all the characters of the object of the StringBuffe class.
- as an output, this method returns or gives the reversed String object.

Example:

```
public class Main{

public static void main(String []args){

StringBuffer sb = new StringBuffer("Welcome to Java");

System.out.println("Original String: " + sb);

sb.reverse();

System.out.println("Reversed String: " + sb);     }}
```

Output :

```
Original String: Welcome to Java
Reversed String: avaJ ot emocleW
```

6) delete() :

- delete() method deletes a sequence of characters from the StringBuffer object.

Syntax:

stringName.delete(int startIndex, int endIndex)

Example:

```
StringBuffer sb= new StringBuffer(“welcometojava");
System.out.println("Original String: " + sb);
sb.delete(0, 4);
System.out.println("String after deleting sequence of characters: "
+ sb);
```

Output :

```
Original String: welcometojava
String after deleting sequence of characters: ometojava
```

7) deleteCharAt() :

- deleteCharAt() method deletes only the character at the specified index

Syntax:

stringName.deleteCharAt(int index)

```
public class Main{
    public static void main(String []args){
        StringBuffer sb= new StringBuffer("Welcome to java");
        System.out.println("Original String: " + sb);
       sb.deleteCharAt(7);
      System.out.println("Resulting String " + sb);     }}
```

Output :

Original String: Welcome to java
Resulting String Welcometo java

8) replace():

- replace() method of the StringBuffer class replaces a sequence of characters with another sequence of characters.

Syntax:

stringName.replace(int startIndex, int endIndex, String string)

Example:

```
public class Main{
    public static void main(String []args){
        StringBuffer s = new StringBuffer("Welcome to java");
        System.out.println("Original String: " + s);
        s.replace(12, 15, "Tutorial");
      System.out.println("Resulting String after replacing: " + s);
  }}
```

Output :

Original String: Welcome to java

Resulting String after replacing: Welcome to jTutorial

8) replace():

```
public class Main{
  public static void main(String []args){
        StringBuffer s = new StringBuffer("Welcome to java");
        System.out.println("Original String: " + s);
        s.replace(7, 12, "Tutorial");
        System.out.println("Resulting String after replacing: "
          + s);
}
}
```

Output :

Original String: Welcome to java

Resulting String after replacing: WelcomeTutorialava

insert(int offset, String s) :

insert() method inserts the given String with this string at the given position

```
public class Main{
    public static void main(String args[]){
        StringBuffer sb=new StringBuffer("Hello ");
            sb.insert(1,"Java");//now original string is changed
            System.out.println(sb);//prints HJavaello
}
}
```

Output :

HJavaello

Example

```
class Stringbuffer{

        )ug

    public static void main(String[] args){

        StringBuffer sb= new StringBuffer(s   "Hello");

        System.out.println("Length of String is: "+ sb.length());

        System.out.println("Capacity of String is: "+ sb.capacity());

        System.out.println("Appended String is: "+ sb.append(str:" World"));

        System.out.println("String after using insert method is: "+ sb.insert(offset:11,str:", welcome to java"));

        System.out.println("String after using delete method is: "+ sb.delete(start:11,end:27));

        System.out.println("Use of deleteCharAt method: "+ sb.deleteCharAt(index:5));

        System.out.println("String after replace method is: "+ sb.replace(start:5,end:9,str:"Diksha"));

        System.out.println("Reversed String: "+ sb.reverse());


    }
}
```

```
PS C:\Users\HP\Desktop\itm\JAVA\practicals_2023\string> javac Stringop.java
PS C:\Users\HP\Desktop\itm\JAVA\practicals_2023\string> java Stringbuffer
Length of String is: 5
Capacity of String is: 21
Appended String is: Hello World
String after using insert method is: Hello World, welcome to java
String after using delete method is: Hello Worlda
Use of deleteCharAt method: HelloWorlda
String after replace method is: HelloDikshada
Reversed String: adahskiDolleH
PS C:\Users\HP\Desktop\itm\JAVA\practicals_2023\string>
```

Command line arguments :

- Sometimes we need to pass information into a program while it is running.
- This can be accomplished by passing command line arguments to **main()**
- The information passed is stored in the string array passed to the main() method and it is stored as a string.

- It is the information that directly follows the program's name on the command line when it is running.

Example-1

```
class Cla{
    Run | Debug
    public static void main(String[] args){
        System.out.println(args[0]);
        System.out.println(args[1]);
        System.out.println(args[2]);
    }
}
```

OUTPUT:

```
PS C:\Users\HP\Desktop\itm\JAVA\practicals_2023> javac cla.java
PS C:\Users\HP\Desktop\itm\JAVA\practicals_2023> java Cla Hello 10 14.3
Hello
10
14.3
```

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

            System.out.println("sum="+args[0]+args[1]);
            int i= Integer.parseInt(args[0]);
            int j = Integer.parseInt(args[1]);
             int sum=i+j;
             System.out.println("sum="+sum);
}
}
```

```
D:\Java-pgm>java Main 2 3
sum=23
sum=5

D:\Java-pgm>
```

Example-2

```
class Cla{

    public static void main(String[] args){
        for(String s: args){
            System.out.println(s);
        }
    }
}
```

OUTPUT:

```
PS C:\Users\HP\Desktop\itm\JAVA\practicals_2023> javac cla.java
PS C:\Users\HP\Desktop\itm\JAVA\practicals_2023> java Cla Hello 10 14.3
Hello
10
14.3
```

varargs

- If we don't know how many argument we will have to pass in the method, varargs is the better approach.

- varargs allows the method to accept zero or multiple arguments.

- Before varargs either we use overloaded method or take an array as the method parameter but it was not considered good because it leads to the maintenance problem.

Syntax: returntype method_name(datatype… value)

```
class VarargsExample2{
 static void display(String... values){
       System.out.println("display method invoked ");
       for(String s:values){
       System.out.println(s);
  }
 }
 public static void main(String args[]){
 display();//zero argument
 display("hello");//one argument
 display("my","name","is","varargs");//four arguments
 }
}
```

```
display method invoked
display method invoked
hello
display method invoked
my
name
is
varargs
```

Wrapper Class

- In java, wrapper classes are used to convert primitive types into class objects

.

- Basically, generic classes only work with objects and don't support primitives. As a result, if we want to work with them, we have to convert primitive values into wrapper objects.

- Each primitive type has corresponding wrapper classes.

- All wrapper classes belongs to the **java.lang** package

Wrapper Class

Primitive Type	Wrapper class
boolean	Boolean
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double

Need of Wrapper Class

- Whenever the primitive types are required as an object, wrapper classes can be used.
- Wrapper classes also include methods to unwrap the object and give back the data type.
- In java.util package, the classes handle only objects. In this case wrapper class are helpful as they convert primitive data type to objects.
- In the Collection framework, Data Structures such as ArrayList store data only as objects and not the primitive types.
- Wrapper classes have methods that support object creation from other object types such as string.
- Wrapper classes are also used for synchronization in multithreading. As in synchronization process, we try to achieve that the shared resource will be used by only one thread at a time. Objects are needed for this.

Wrapper Class

Creating wrapper object:

1. Using wrapper class constructor //deprecated

2. Using wrapper class only

3. Using valueOf() method

1) Using wrapper class constructor

- We can create a wrapper object using the wrapper class and its constructor by passing the value to it.

Syntax:

ClassName object = new ClassName(argument);

- **Example**:

Integer number = new Integer(58);

- way of creating an instance **of wrapper classes using constructor is deprecated** as of the latest version of JDK.
- This is because each time new memory is allocated in the heap when we create an object with the help of the constructor.
- **constructor Character(char) has been deprecated** since JDK version 9.

2) Using Wrapper Class

- We can create Wrapper class object without using constructor as well by using the wrapper class instead of the primitive type to create a wrapper object without passing the value to the constructor

- To get the value, we can print the particular object.

- **Syntax:**

  ClassName object = value;

- **Example:**

```
Integer intValue = 10;
System.out.println(intValue);
```

3) Using valueOf() method

By using valueOf Static method, a Wrapper object can be created.

- **Syntax :**

ClassName object = ClassName.valueOf(argument);

- Example:

```
int a= 10;
Integer num= Integer.valueOf(a);
```

Wrapper class methods

S. No.	Method	Method Description
1	typeValue()	Converts the value of this Number object to the specified primitive data type returned
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 data type value
5	toString()	Returns a String object representing the value of specified Integer type argument
6	parseInt()	Returns an Integer type value of a specified String representation
7	decode()	Decodes a String into an integer
8	min()	Returns the smaller value after comparison of the two arguments
9	max()	Returns the larger value after comparison of the two arguments
10	round()	Returns the closest round off long or int value as per the method return type

- The **wrapper class in Java** provides the mechanism to convert primitive data type into object and object into primitive data type.
- Since J2SE 5.0, **autoboxing** and **unboxing** feature convert primitives into objects and objects into primitives automatically.
- The automatic conversion of primitive into an object is known as autoboxing and vice-versa unboxing.

Autoboxing :

- Autoboxing is when the Java compiler performs the automatic conversion of the primitive data types to the object of their corresponding wrapper classes.
- Converting an *int* to *Integer*, a *double* to *Double*, etc.

Java compiler applies autoboxing when a primitive value is:

- Passed as a parameter to a method that expects an object of the corresponding wrapper class.
- Assigned to a variable of the corresponding wrapper class.

To explicitly convert primitive to object,we use **valueOf()** method.

Autoboxing: Example

```
class Autobox{

    public static void main(String[] args){
     int i=10;
     //auto-boxing primitive to Integer object conversion
     Integer I=i;
     System.out.println("Value of object I= "+I);
     //Explicitly wrapping (boxing)
     Integer A=Integer.valueOf(i);
     System.out.println("Explicit wrapping: "+A);
    }
}
```

```
PS C:\Users\HP\Desktop\itm\JAVA\practicals_2023\demo> java Autobox
Value of object I= 10
Explicit wrapping: 10
PS C:\Users\HP\Desktop\itm\JAVA\practicals_2023\demo>
```

```
//Java program to convert primitive into objects
//Autoboxing example of int to Integer
public class WrapperExample1{
    public static void main(String args[]){
        //Converting int into Integer
        int a=20;
        Integer i=Integer.valueOf(a);//converting int into Integer explicitly
        Integer j=a;
        //autoboxing, now compiler will write Integer.valueOf(a) internally

        System.out.println(a+" "+i+" "+j);
    }}
```

Output:

20 20 20

Auto-Unboxing

- Unboxing is automatically converting an object of a wrapper type (Integer, for example) to its corresponding primitive (int) value.
- The Java compiler applies unboxing when an object of a wrapper class is:
  a) Passed as a parameter to a method that expects a value of the corresponding primitive type.
  b) Assigned to a variable of the corresponding primitive type.
- To explicitly convert object to corresponding primitive type we use **xxxValue()** method.

Auto-Unboxing: Example

```
class Autounbox{

        atic void main(String[] args){
    Integer I=10;
    //auto-unboxing primitive to Integer object conversion
    int i= I;
    System.out.println("primitive value = "+i);
    //Explicitly un-wrapping (unboxing)
    int a= I.intValue();
    System.out.println("Explicit un-wrapping: "+a);
  }
}
```

```
PS C:\Users\HP\Desktop\itm\JAVA\practicals_2023\demo> javac Autoboxing.java
PS C:\Users\HP\Desktop\itm\JAVA\practicals_2023\demo> java Autounbox
primitive value = 10
Explicit un-wrapping: 10
```

toString() method

- toString() method returns the String representation of the object.
- If you print any object, Java compiler internally invokes the toString() method on the object.
- So overriding the toString() method, returns the desired output,

```
class Stringconvert{
    public static void main(String[] args){
     Integer I=100;
     String s=I.toString();
     System.out.println("Value of String s: "+s);

    }
```

```
PS C:\Users\HP\Desktop\itm\JAVA\practicals_2023\demo> java Stringconvert
Value of String s: 100
PS C:\Users\HP\Desktop\itm\JAVA\practicals_2023\demo>
```