



RAMESHSOFT

SOFTWARE TRAINING INSTITUTE

JOB CONSULTANCY

JAVA WITH SELENIUM PART-1 CORE JAVA

***RAMESH ANAPATI
RACHAMALLA SAITEJA***



ISBN : 978-93-92941-08-5

JAVA WITH SELENIUM

IN REAL TIME

CORE JAVA

PART-1

RAMESH ANAPATI

RACHAMALLA SAITEJA



Title of the Book: JAVA WITH SELENIUM IN REALTIME – CORE JAVA(PART-1)

Edition: First- 2022

Copyright 2022 © Authors

No part of this book may be reproduced or transmitted in any form by any means, electronic or mechanical, including photocopy, recording or any information storage and retrieval system, without permission in writing from the copyright owners.

Disclaimer

The authors are solely responsible for the contents published in this book. The publishers or editors don't take any responsibility for the same in any manner. Errors, if any, are purely unintentional and readers are requested to communicate such errors to the editors or publishers to avoid discrepancies in future. Few sentences in this book are written in general talkative way to understand the beginners. Few images are taken from internet sources.

ISBN: 978-93-92941-08-5

MRP Rs.

SR edu publications

Publisher and Distributor

12-51/d/13/a, Kalyan Nagar

Kalwakurthy, Telangana-509324

Phone: +919553211467

Email: sredupublications@gmail.com

Website: www.Sredupublications.com





Contents:

- Overview of java
- Installing of java Installing of Eclipse
- Features of java
- Data Types in java Primitive Data Types Non-Primitive Data Types
- String Class
- Conditional statements
 - a. Selection statements
 - i. If Statements
 - ii. If-else Statements
 - iii. Else-if Statements
 - iv. Switch
 - b. Iterative Statements
 - i. While
 - ii. do-while
 - iii. for
 - iv. foreach
 - c. Transfer statements
 - i. Break
 - ii. Continue
 - iii. Throw
 - iv. Throws
 - v. Finally
- Usage of Control Statements in selenium
- Variables
 - i. Static Variables
 - ii. Non-Static Variables
 - iii. Local variables
 - iv. Object type|reference type variables
- Methods
 - i. Static Methods
 - ii. Non-Static Methods
 - iii. Generic methods
- What is Main() Method
- What is Class
- What is Object

- Initialisation of Variables
 - i. Through variables
 - ii. Through console
 - iii. Through Object
 - iv. Through constructors
- What is Constructors
- What is Constructor Overloading
- Automatic promotions in methods & constructors
- What is this() constructor
- What is super() constructor
- What is this keyword
- What is super keyword
- Mutable vs Immutable
- Inheritance
 - i. Types of Inheritance Concepts
 - ii. WORA Mechanism
- Polymorphism
 - i. Overloading
 - ii. Overriding
- Data Hiding
- Encapsulation I
- S-A & HAS-A Relationships
- Data Abstraction
- Interfaces
- Marker interface
- Functional interface
- Partially abstract interface
- Fully abstract interface
- Default methods inside interface
- Static methods inside interface
- Private static & non static methods inside interface
- Adapter classes in java
- Abstract Methods
- Abstract Classes
- READ Only classes
- Interfaces java1.8 & 1.9
- Importance of OOPS Concept for selenium
- Accessing of Classes in different packages
- Arrays
 - i. Single Dimensional Arrays
 - ii. Two Dimensional Arrays
- Object Arrays
- Collections (java.util package in detailed)
 - List
 - Set

- ArrayList
- LinkedList
- HashSet
- TreeSet
- LinkedHashSet
- NavigableSet
- SortedSet
- Vector
- stack
- Classes
- ConcurrentHashMap
- WeakHashMap
- TreeMap
- SortedMap
- NavigableMap
- Comparable vs Comparator
- Queue
- Dequeue
- PriorityQueue
- cursors
- Comparable & Comparator for sorting
- Utility classes(Arrays,Collections classes)
- Map ,HashMap, Linked, HashMap ,TreeMap, NavigableMap ,SortedMap, Hashtable,Concurrent HashMap.

JAVA.LANG package

- Object class
- Object class child classes(String StringBuffer StringBuilder Number Throwable Error Exception Integer.....)
- Wrapper classes

JAVA LOMBOK

Auto Boxing and Auto Unboxing

CORE JAVA

What is java?

Java is a general-purpose, class-based, object-oriented programming language designed for having lesser implementation dependencies. It is a computing platform for application development. Java is fast, secure, and reliable, therefore. It is widely used for developing Java applications in laptops, data centres, game consoles, scientific supercomputers, cell phones, etc.

Or

One of the most widely used programming languages, Java is used as the server-side language for most back-end development projects, including those involving big data and Android development. Java is also commonly used for desktop computing, other mobile computing, games, and numerical computing.

How to execute a program in java.

1.Programmer:

- Write the program by programmer
- In notepad / notepad++/edit plus/Ide's
- Have to follow Rules Regulations Syntaxes

2. Validation

- Validation/compilation process
- compilation is done by compiler
- compiler is a software which is responsible for validation whether the programmer is following the rules and syntaxes

3.Execution

- in java the program will execute in java virtual machine (JVM)

.java

- After writing the program in notepad/ notepad++/ edit plus/ Ide's
- We are going to save the file as .java file
- example: Addition.java (file name)

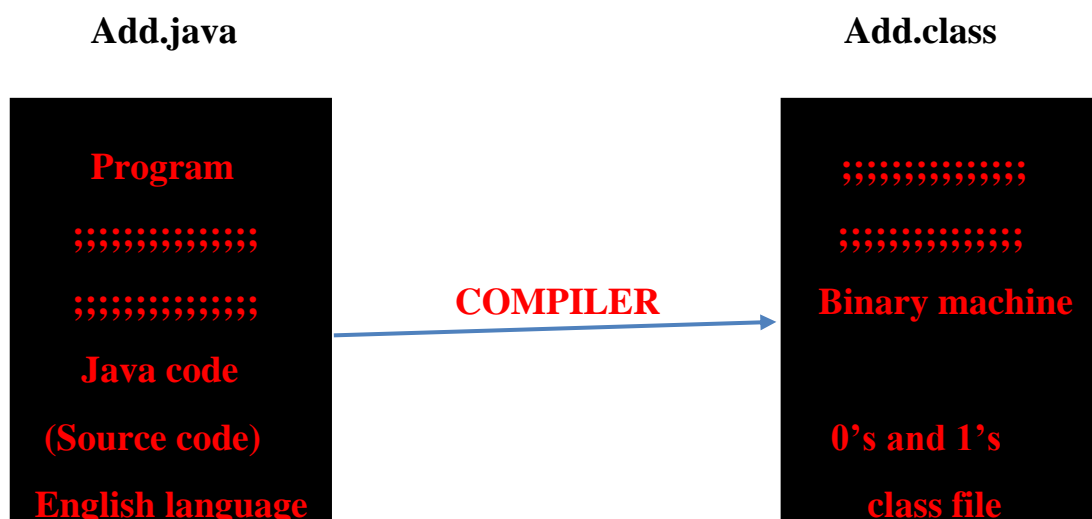
File Formats

abc.txt	abc.xls	test.csv	demo.Properties	demo.xml	demo.java
text	excel	comma	key val	tax	java

.class

After executing the program by JVM there are two cases occur

- case1: If any errors occur in the program while executing **.class file** is NOT going to generated.
- Case2: If there are NO ERRORS while executing the program (source file) **.class file** is going to generated



Class:

A Class is a container or place holder where we are going to write our java programs.

- To write something in java that something should be there inside a container called class.

Syntax for class:

```
<access specifier> class <class name>
{
    ;;;;;;;;;;
}
```

➡ access specifier: <as>

- specifying access of a class (whether accessible outside or not)
- In general way – To keep restrictions.
- <as> is optional, not mandatory.
- All access modifiers will be in lower case.

Types of access specifiers:

1. Public
2. Private
3. Protected
4. Default

➡ Class:

- class is fixed key word (reserve word or predefine word) in order to convey it as a class.
- Every program must start with class keyword

➡ Classname:

- It is Mandatory in java
- Classname can be anything but it should be very meaning full.
- It should not start with number and special characters.
- Classname should contain underscore for more readability
Example: Add_Test_Program
- Every class name should be capital letter for each word.

<access specifier> (Optional)

Class (Mandatory)

<classname> (Mandatory)

{ } (Mandatory)

Examples:

➤ Public class Addition_Program

```
{  
    ;;;;  
}
```

➤ Class Addition_Program

```
{  
    ;;;;  
} // <as> is not mandatory
```

➤ Private class Addition_Program **(invalid)**

```
{  
    ;;;;  
} //at class level we should not specify private
```

➤ Default class Addition_Program **(invalid)**

```
{  
    ;;;;  
} //at class level we should not specify private
```

➤ Protected class Addition_Program **(invalid)**

```
{  
    ;;;;  
} //at class level we should not specify private
```

In general, we define 5 kinds of data.

1. Whole numbers
2. Real numbers
3. Words
4. Alphabets
5. Logical values

Examples: in general way

Whole numbers === 456, 157, etc.

Real numbers === 456.456, 951.125 etc.

Words === java, selenium etc.

Alphabets === R, J, S etc.

Logical values === True, False

DATA TYPES:

- In java terminology.
- 1. whole numbers → integer data
- 2. real numbers → Floating data
- 3. words → String data
- 4. alphabets → character data
- 5. logical values → Boolean data

Q) How to define the data?

- Define the data (5 kinds of data).
- Give or assign some name to the data (variables)
- Tell what type of data it is? (Data type)
- Specify the scope of data (access specifier)
- If data is not going to be changes then the data is static.
- If data is going to changes then the data is non-static.

Examples:

- Static public integer number1 = 25;
 (scope) (datatype) (variable) (data/value)
- Static public floating number2 = 456.456;
- Static public string course = java;
- Static public char letter = A;
- Static public Boolean stmt = true;

Types of Data types:

There are two types of data types:

1. Primitive data type
2. Non-primitive data type

Using data type, we can tell what type of data it is.

Primitive Data type:

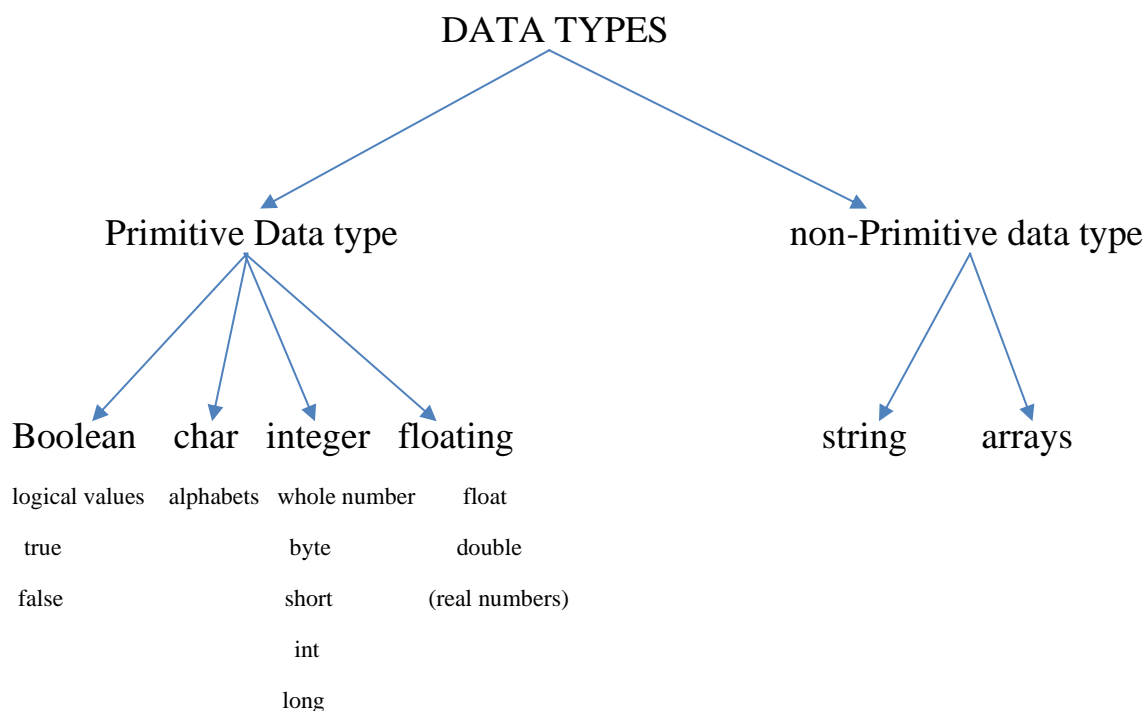
- These are system define data types
- These are fixed in their memory size
- These are 8 in numbers

Primitive data types are: byte, short, int long, float, double, char, boolean.

Non-Primitive Data type:

- These are not fixed in their memory size.

Non-Primitive data types are strings, arrays



Whole Numbers → Integer data type:

- byte → 1 byte (8bits) = 127 is max -128 is minimum.
- Short □ 2 byte (16bits) = 32,767 to -32,768
- Int □ 4 byte (32bits) = 2,147,483,647 is max
- Long □ 8 byte (64bits) = 33333333,6565655. . .
For long suffix l should be there while defining data type.

Real Numbers □ Floating point data type:

- float □ 4 bytes (32bits) = (0 to 6 decimal places)
- double □ 8 bytes (64bits) = (6+ decimal places)

for float we should mention suffix f

example:

456.456f □ float data type

0.456123456 □ double

2.0f □ float

456123458.4f □ float

Note: By default, every floating-point data will be double.

Logical values □ Boolean data type:

- True
- False

Alphabets □ char data type:

In java to convey char should be enclosed with single quotes ‘ ‘

- Char can be (a-z) and (A-Z)
- 0 to 65,535

Char ‘a’ (valid)

Char ‘ ‘ (invalid)

Char ‘ad’ (invalid)

Note: To convey it as a char the character or alphabet should be enclosed with a single quotes and no double alphabets.

Words □ String data type:

A group of characters is known as a String.

- In java in order to convey it as a String, it always should be enclosed with double quotes (“ ”)
- Whatever you write in double quotes (“ ”) by default it becomes string.
- In String S should be always capital letter.
Example: “java” , “automation”, “testing”, “selenium”.

Variables:

Variables is a container or place holder where we are going to define “data”.

- Variables should be declared inside a class.
- In java if we want to define some data that data should be inside a variable only.

Types of Variables:

1. Static variables (class level variables)
2. Non-static variables (instance variables)
3. Local variables (method level variables)
4. Object type | reference type variables
 - Ot | rt static variables
 - Ot | rt non- static variables
 - Ot | rt local variables

Static variables (class level variables):

Whenever the value /data is not going to changes from object to object if it is constant then we go for static variables.

□After starting the class immediately, we can declare the static variables.

Syntax:

Static <as> datatype variable name = value / data;

Or

<as> static datatype variable name = value/data;

Example:

```
Public static int numberOne = 951;
```

Note: This is called declaration and initialization in a single line.

Where- static □ mandatory (if value/data is constant)

<as> □ optional (specifying the access of variables)

datatype □ mandatory (defines the type of data)

variable name □ mandatory

= □ assignment operator

Value/ data □ five kinds of data.

; □ semicolon (end of the sentence)

Note: every variable starts with alphabets, first letter of first word is small and first letter of next word is capital.

□ should not start with number or special characters.

Ex: numberOne, variableName etc.

```
<as> static datatype variableName; ( declaration)
```

```
variableName = value/data; (initialization)
```

Note: the above lines represent the initialization and declaration in separate lines or not in a single line.

Non-static variables (instance variables):

Whenever the value/data is going to changes from object to object if it is not constant then we go for non-static.

□ After starting the class immediately, we can declare the static variables.

Example: Bank name sbi □ static (won't change constant)

Customer name □ non-static (changes from customer to customer).

Syntax:

```
<as> datatype variableName = value / data;
```

(Declaration and initialization in a single line)

Or

```
<as> datatype variableName;
```


variableName = value/data;

(Declaration and initialization in two lines)

Note: initialization is not mandatory for static and non-static variables.

Default values:

For every data type there will be a default value:

byte short int long □ 0

float double □ 0.0

string □ null

boolean □ false

char □ ascii value

Write a program for addition between two umbers?

```
Public class Addition_Program
{
    Public int num1 = 25;
    Public int num2 = 65;
    int result = num1+num2;
}
```

Methods:

A method is a container or place holder where we are going to write our business implementation or business logics of business requirements.

For example in above mentioned program we have declared and intialised

```
int result = num1+num2;
```

the above line is the implementation or logic of the required addition between two numbers program.

Types of Methods:

1. Static methods
2. Non-static methods

Static Methods:

- Whenever business implementation are constant (not going to change) then that method is known as static method.
- If you want to access without creating an object then we go for static method.
- If you want to create only read only then we go for static method.

Syntax

```
<as>  static void | return type  methodname(---)
{
; ; ; ; ; ; ;
return statement;
}
```

Or

```
Static  <as>  void | return type  methodname(---)
{
; ; ; ; ; ; ;
return statement;
}
```

➔ whereas

<as> - specifying the scope of business method (optional)

Static - keyword in order to convey it as a static method (mandatory)

Void - it returns nothing (just business implementation will execute)

{ } - mandatory

Note:

- Per method only one return statement should be there that too end statement.
- If any statement taken or written after return statement then compile time error will occur.

Example:

Business requirement: cash deposit in atm.

- Insert card
- enter pin
- enter account number
- insert the cash
.....
- once deposited 600rs
- your balance will be 25000+600

void

return type

-return 25600(display on screen with your total balance)

Case1: once cash is deposited don't return to end user (void)

Case2: once cash is deposited returns the total amount to end user (return type)

Write a program for addition between two numbers using void?

```
Public class Addition_Program
{
Public static num1= 25;
Public static num2 = 65;
Public void addition ( )
{
int result = num1 + num1;
}
}
```

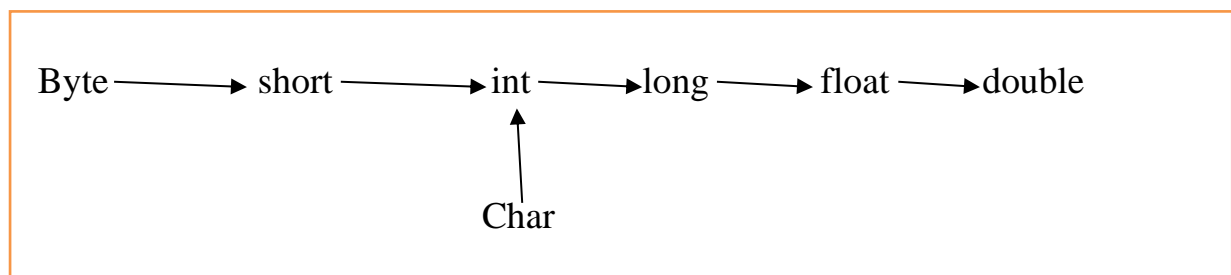
Write a program for addition between two numbers using return type?

```
Public class Addition_Program
{
Public static num1= 25;
Public static num2 = 65;
Public int addition ( )
{
int result = num1 + num1;
return result;
}
}
```

Here at end return result statement is the end statement.

Result is type of int data type. So, return type should be int.

(Whatever the Result type is of which data type then the return type should be that only)



- Here double can hold all data types.
- Float can hold long, int, char, short, byte.
- Long can hold int, char, short, byte.
- Int can hold char, short, byte.
- Short can hold only byte.

PRINTING STATEMENT

- In order to print some information on the console

System.out.println();

Whereas □ system-----> it is a pre define class [class System]

. -----> it is a dot operator, any word after it can be reference variable/object or method

out-----> it is an object (predefine)

println()----> it is a method (predefine)

What ever you write in double quotes it will print as it is

□ println()----->print next message in new line

□ print()----->print next message in same line

Example: system.out.println();

System.out.println(string messege);

System.out.println("hello world");

In above example hello world is going to print as it is in console.

Example program:

```
public class Career
{
    public static void main(String args [ ] )
    {
        System.out.println("hello world");
        System.out.print("learn java ");
        System.out.println("along with selenium.");
        System.out.print("to get knowledge on automation testing");
    }
}
```

Output:

hello world

learn java along with selenium.

to get knowledge on automation testing

Write a program for addition between two numbers?

```
Public class Addition_Program
{
Public static num1= 25;
Public static num2 = 65;
Public int addition ( )
{
int result = num1 + num1;
system.out.println("result is: "+result);
return result;
}
}
```

Main Method:

main ();

→Main method in java is required in order to execute the program.

□JVM always starts execution from main method

□it is the starting point of execution

□inside the main () we do testing/ accessing the defined part.

□we can define the main method after starting a class immediately or also we can define in between the class or we can define main method before end of the statements.

Syntax:

```
public static void main(String args[ ] )
{
// we do testing | accessing the defined part
}
```

Or

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

Or

```
public static void main(String .... args )  
{  
    ;;;;;;;;;;  
}
```

In place of args we can write anything

Or

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

Example program:

```
public class AboutMe  
{  
    public static void main(String args[ ])  
    {  
        System.out.println("My Name is: Paul ");  
        System.out.println("I am 22 years old");  
    }  
}
```

Output

My Name is: Paul I am 22 years old

Write a program for addition between two numbers?

```
Public  class  Addition_Program
{
Public  static  num1= 25;
Public  static  num2 = 65;
Public  int    addition ( )
{
int    result  = num1 + num1;
system.out.println("result is: "+result);
return result;
}
}
Public  static  Add_Demo
{
public static void main(String args[ ])
{
System.out.println("jvm is started");
}
}
```


JAVA SOURCE FILE STRUCTURE:

- In a single source file we can have multiple classes

Class A

```
{  
}
```

Class B

```
{  
}
```

Class C

```
{  
}
```

- If no class is declared as public we can save that file with any name
Example: A.java, B.java, c.java, Demo.java
- Declare one class as public, the class which you declare as public the classname and file name should be same otherwise you will get a compile time error.
- We should not take more than one class as public. If you declare more than one class as public you will get a compile time error.
- The class which you declare as public, the public class name and file name must and should be same.
- Compulsory we should take the main () inside the public class only.

How to access | test static variables and static methods?

1. Directly by calling their names.
2. Through class name.
3. Through object.

How to access | test non-static variables and non-static methods?

1. Through object.

➔ Directly by calling their names:

datatype varname = static_variable_name;

example : string nameresp = name;

datatype varname = static_method_name; (if the method is having return type)

static_method.name; (if the method is not having return type)

example:

```
public class TestDemo
{
    int a=10;
    static int b= 20;
    system.out.println("Hello");
}

Public static void display()
{
    System.out.println("welcome to rameshsoft");
}

Public static void main (String[ ] args)
{
    System.out.println("the value of b is: "+b);
    display();
}
}
```

Output.

The value of b is: 20

Welcome to rameshsoft

Write a program for addition between two programs?

```
public class addition {  
    int num1=15;  
    int num2=15;  
    public int addition()  
    {  
        int result = num1+num2;  
        System.out.println("result is;"+result);  
        return result;  
    }  
    public static int additionOne()  
    {  
        int resultOne = 25+25;  
        System.out.println("resultOne is;"+resultOne);  
        return resultOne;  
    }  
    public static void main(String[] args) {  
        System.out.println("rachamalla");  
        int addresp = additionOne();  
        System.out.println("additionOne()resp is;"+addresp);  
    }  
}
```

Output:

```
rachamalla  
resultOne is;50  
additionOne()resp is;50
```

➔Through classname:

`datatype varname = classname.static_variable_name (static variables)`

`datatype varname = classname.static_method_name (static methods)`

whenever in program we have separated the execution part with separate class error will shown due to directly calling. So, to overcome this issue through classname was implemented.

To resolve we need to write

`String nameresp = classname.variablename`

`Int addresp = classname.methodname`

example program:

```
public class TestDemo
{
    int a=10;
    static int b= 20;
    Public void hello( )
    {
        system.out.println("Hello");
    }
    Public static void display()
    {
        System.out.println("display");
    }
    Public static void main (String[ ] args)
    {
        Int c= Testdemo.b;
        System.out.println( c);
        TestDemo.display( );
    }
}
```

}

Output:

20

Display

□Through object:

- 1) static level
- 2) Dynamic level

Static level:

- Using new Operator
- Using new instance of
- Using Deserialization
- Using clone mechanism

Dynamic Level:

- Using class.forName(-)

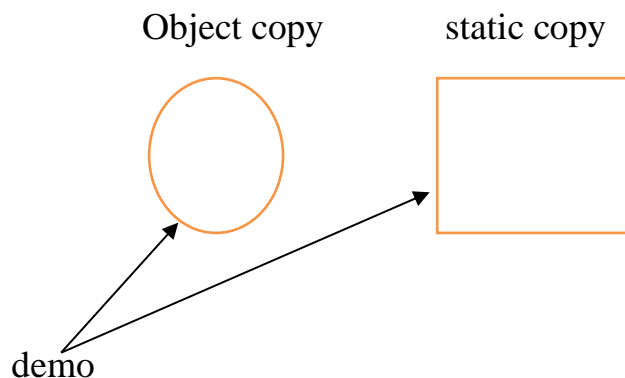
New Operator

How to create object:

Syntax:

Classname varname = new classname();

Ex: Add_Demo demo = new Add_Demo();



example program:

```
public class TestDemo
{
    int a=10;
    static int b= 20;
    Public void hello( )
    {
        system.out.println("Display");
    }
    System.out.println("display");
}
Public static void main (String[ ] args)
{
    Testdemo testdemo = new TestDemo( );
    Int c = testdemo.a;
    System.out.println©;
    testDemo.hello( );
    testDemo.display ( );
}
}
```

Output:

10

hello

Display

Example program:

```
public class add {  
  
    public void hello() {  
        System.out.println("hello()");  
    }  
  
    public static void main(String[] args) {  
        System.out.println("java");  
        add demo = new add();  
        demo.hello();  
    }  
}
```

Output:

```
java  
hello()
```

➔ How to access non-static variables and non-static methods:

We can access non-static variables and methods by using only creating an object.

Ex: Testdemo testdemo = new TestDemo();

In the above program, display() method is a static method and hello () is a non-static method..

Static Vs Non-Static:

- Inside non-static area i.e., non-static method, we can access both static and non-static variables and methods.
- Inside static area i.e., static method. We can access only static variables and methods.
- We cannot access non-static variables or methods inside static method.

- If we want to access, it will show compile time error as cannot make reference non-static fields in to static area.

Local Variables:

Whenever you want to restrict the data to certain portion of area within the class then we can go for local variables.

- Whenever you declare the variables inside the method such kind of variables are called local variables.
- Local variable names and class level variable can be same.
- Use **this**. Keyword to take the class level variables. If not, priority always goes to local variables.
- Use **final** keyword to stop re assignment.

Scope:

- within the method only we can access.
- outside the method we cannot access.

For Local variables no <as> (access specifier) is allowed only **final** is allowed. Initialisation of local variables is mandatory when ever we are using variables.

ObjectType | reference type variables:

Addition_Program test = new Addition_Program (Non-static ot|rt variables)

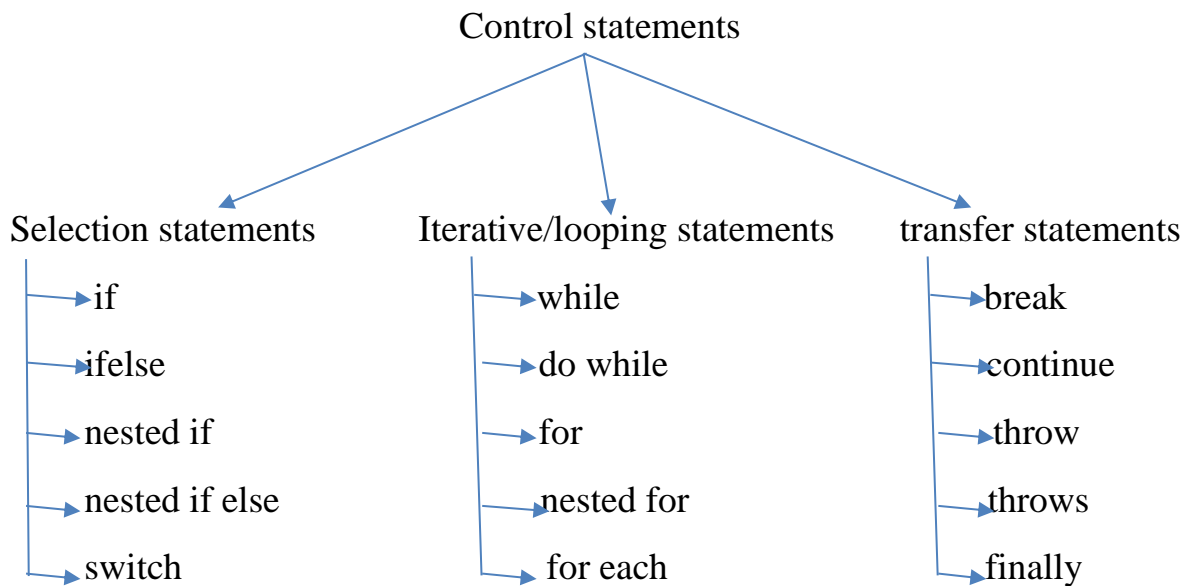
Static Addition_Program test1 = new Addition_Program (static ot|rt variables)

Addition_Program test = new Addition_Program (if it is in method then it is local variable)

Control statements:

By using control statements, we can tell the flow of execution in which order statement should execute

(Flow of execution)



Selection statements:

if: -

If you want to execute the statements based on some conditions then we go for if statements.

Syntax:

```
if (condition)
{
    ;;;;;;
}
;;;;;;
;;;;;;
```

if block

normal statements

- Condition should be Boolean either true or false
- if condition is true, program will execute if block then follows by normal statements.
- if condition is false it won't execute if block.

if else: -

If you want to execute the statements based on either based condition then we go for if else statements.

Syntax:

```

if (condition)
{
    ;;;;;;
}
else
{
    ;;;;;;
}

```

- if if condition is true, if block will executed followed by normal statements
- If if condition is false, else block will executed.
- For if and if else ({ }) not mandatory

nested if: -

```

if ( condition)           (outer if condition)
{
    ;;;;;;
    If (condition)       (inner if condition)
    {
        ;;;;;;
    }
}

```

- If outer condition is true then outer condition is going to executed.

- If inner condition is true then inner condition will going to executed.

Nested if else:

```

if(condition)
{
    ;;;;;
    If (condition)
    {
        ;;;;;
    }
    else
    {
        ;;;;
        if(condition)
        {
            ;;;;
        }
        ;;;;
    }
}

```

Switch: -

Using switch, we can execute single condition or multiple conditions.

Syntax:

```

Switch (condition)
{
    Case    case1:  ;;;;;;;;;
                Break;

```

```

Case    case2:  ;;;;;;
                Break;

Case    caseN   ;;;;;;
                Break;

;;;;;;;;;;;;;

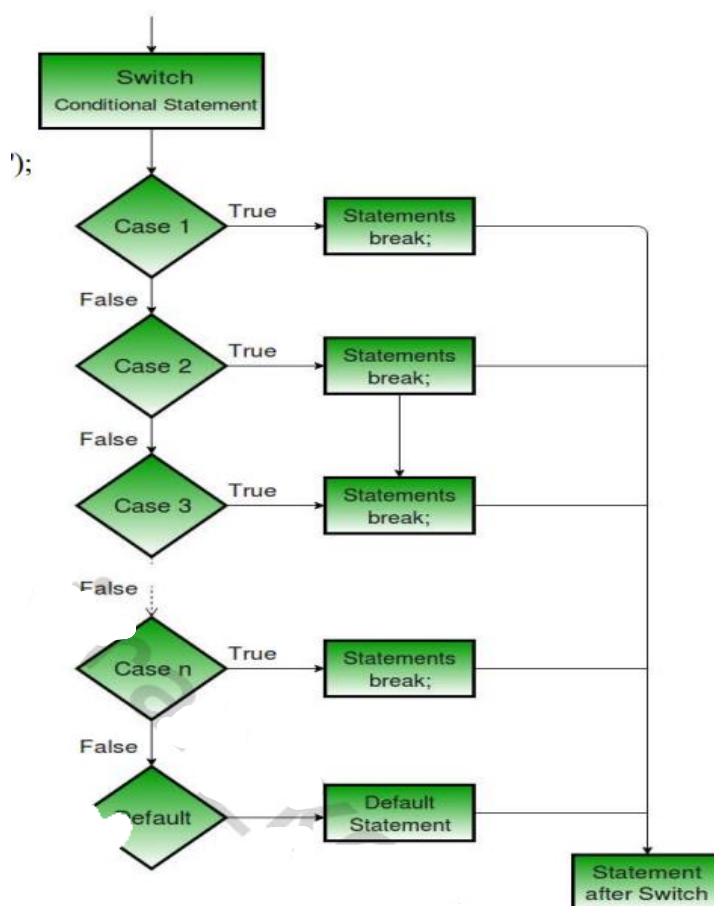
Default ;;;;;;;;;;

Break;

}

```

- Switch is a key word to convey it as switch.
- Switch condition always should be byte short int long string Enum other than these if you specify you get a compile time error.
- Case is a fixed key word to convey it as switch case.



Looping Statements:

Whenever you want to execute same statement multiple times then we go for looping statements.

Types of looping/iterative/repetitive statements:

1. while
2. do while
3. for
4. nested loops
5. for each loop

➔ While loops:

If you don't know No. of iterations in advance then we go for while loop

Syntax:

Initialisation section;

While (condition)

{

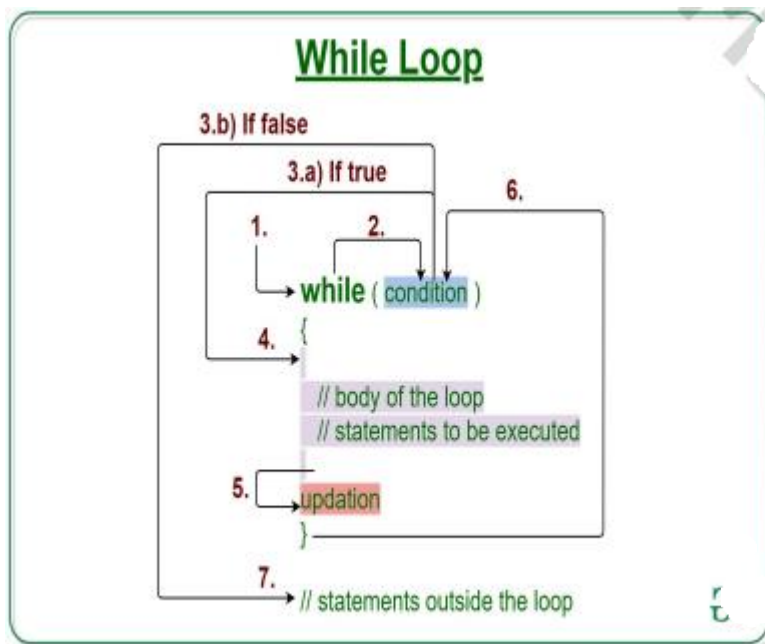
;;;;;;;

Inc | dec section;

}

;;;;;;;

- Condition always Boolean if condition is false never executed.
- If condition is true then it executes until the condition becomes false.



➔do while loops:

If you want to execute loop body at least once irrespective of condition then go for do while.

Syntax:

Initialisation section;

do

{

.....

Inc | dec section;

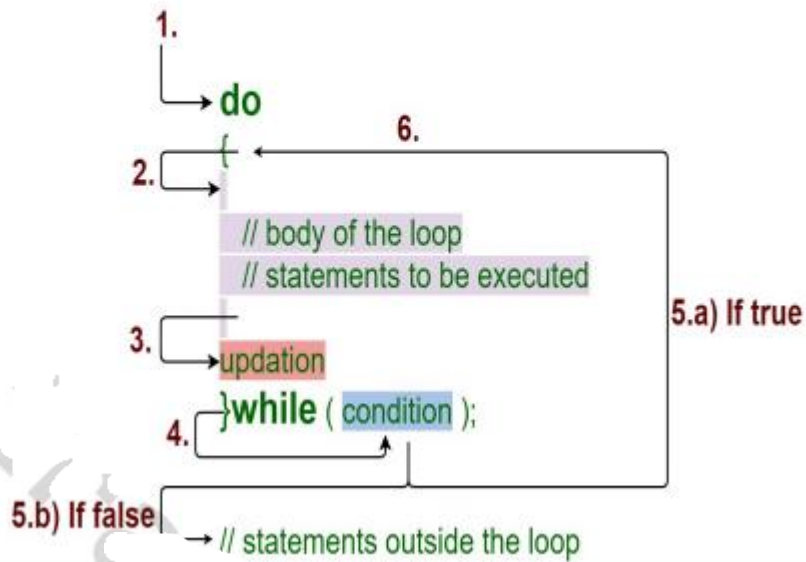
}

While (condition);

.....

- If do while is true or false the loop body executes irrespectively.

Do - While Loop



➔for loop: -

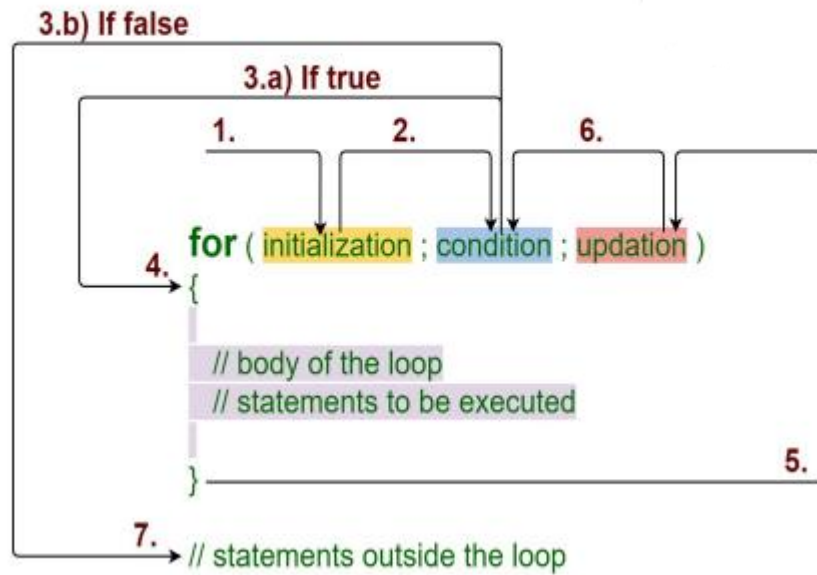
If you know no. of iterations in advance then go for **for loop**.

Syntax:

```
for (initialisation section; condition; increment/ decrement section)
{
    ;;;;;;
}
```

➤ For (; ;) empty conditions is also valid.

For Loop



➔nested loop: -

A loop inside another loop is called nested loop.

Syntax:

```
for (initial sec; condition; inc | dec section)
{
    ;;;;;;;;;;
    for (initial sec; condition; inc | dec section)
    {
        ;;;;;;
    }
}
```


Programs

1. write a program for addition between two numbers?

```
public class addition {  
    int num1=15;  
    int num2=15;  
    public int addition()  
    {  
        int result = num1+num2;  
        System.out.println("result is;" +result);  
        return result;  
    }  
    public static int additionOne()  
    {  
        int resultOne = 25+25;  
        System.out.println("resultOne is;" +resultOne);  
        return resultOne;  
    }  
    public static void main(String[] args) {  
        System.out.println("rachamalla");  
        int addresp = additionOne();  
        System.out.println("additionOne()resp is;" +addresp);  
    }  
}
```

Output:

```
rachamalla  
resultOne is;50  
additionOne()resp is;50
```

2. write a program for factorial number?

```
public class factorial {  
    public static void main(String[] args) {  
        int n=4;  
        int i;  
        int fact=1;  
        for(i=1;i<=n;i++)  
        {  
            fact=fact*i;  
        }  
    }  
}
```

```

        System.out.println("factorial of "+n+"is: "+fact);
    }
}

```

Output:

factorial of 4is: 24

3. write a program for multiplication table?

```

public class multiplicationtable
{
    public static void main(String[] args)
    {
        int num=16;
        int k;

        for(int i=1;i<=10;i++)
        {
            k=num*i;
            System.out.println(num+"X"+i+"="+k);
        }
    }
}

```

Output:

```

16X1=16
16X2=32
16X3=48
16X4=64
16X5=80
16X6=96
16X7=112
16X8=128
16X9=144
16X10=160

```

4. write a program for Fibonacci series?

```
public class Fibonacci
{
    public static void main(String[] args)
    {
        int r=0;
        int s=1;
        int t;
        int i;
        int n=15;
        System.out.println(r+" "+s);
        for(i=2;i<n;i++)
        {
            t=r+s;
            System.out.print(" "+t+" ");
            r=s;
            s=t;
        }
    }
}
```

Output:

```
0 1
1 2 3 5 8 13 21 34 55 89 144 233 377
```

5.write a program to find odd numbers?

```
public class numbers
{
    public static void main(String[] args)
    {
        int n=15;
        for(int i=1;i<=n;i++)
        {
            if(i%2==1)
                System.out.println("list of odd numbers of 50 is: "+i);
        }
    }
}
```

Output:

list of odd numbers of 50 is: 1
list of odd numbers of 50 is: 3
list of odd numbers of 50 is: 5
list of odd numbers of 50 is: 7
list of odd numbers of 50 is: 9
list of odd numbers of 50 is: 11
list of odd numbers of 50 is: 13
list of odd numbers of 50 is: 15

6. write a program to draw a pattern?

```
public class pattern
{
    public static void main(String[] args)
    {
        int i;
        int j;
        int k=5; //no of rows
        for(i=0;i<k;i++)
        {
            for(j=0;j<=i;j++)
            {
                System.out.print("* ");
            }
            System.out.println();
        }
    }
}
```

Output:

```
*
* *
* * *
* * * *
* * * * *
```

7.write a program to find the given number is positive or negative?

```
public class postive_or_negative
{
    public static void main(String[] args)
    {
        int n=10;

        if(n>0) {
            System.out.println("the number "+n+" is positive");
        }
        else if(n<0)
        {
            System.out.println("the number "+n+"is negative");
        }

    }

}
```

Output:

the number 10 is positive

8. write a program for sum of natural numbers?

```
public class sum_of_natural_numbers {
    public static void main(String[] args) {
        int n=20;
        int sum=0;
        for(int i=0;i<=n;i++)
        {
            sum=sum+i;

        }
        System.out.println("the sum of "+n+"natural numbers is: "+sum);
    }
}
```

Output:

the sum of 20natural numbers is: 210

9.write a program to swaping the two number without third variable?

```
public class swap_wo_3rdvar {  
    public static void main(String[] args) {  
        int a=10;  
        int b=20;  
  
        System.out.println("number a is:"+a);  
        System.out.println("number b is:"+b);  
  
        a=a+b;  
        b=a-b;  
        a=a-b;  
  
        System.out.println("result of swap without 3rd var is:"+a+"a+"  
        "+"b="+b);  
    }  
}
```

Output:

number a is:10

number b is:20

result of swap without 3rd var is:a=20 b=10

10. write a program for swapping two numbers with third variable?

```
public class swapping  
{  
    public static void main(String[] args)  
    {  
  
        int a=5;  
        int b=10;  
        int temp;  
        System.out.println("number a is:"+a);  
        System.out.println("number b is:"+b);  
  
        temp=a;  
        a=b;  
        b=temp;
```

```
System.out.println("result of swap a and b is: "+a+" "+b);
```

```
}  
}
```

Output:

number a is:5

number b is:10

result of swap a and b is: 10 5

11.WAP to print sum of all even numbers from 1 to 15

```
class LoopEvenSum
```

```
{
```

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

```
{
```

```
int i,sum=0; for(i=1;i<=15;i++)
```

```
{
```

```
if(i%2==0) { sum=sum+i;
```

```
}
```

```
}
```

```
System.out.println("Final sum value is: "+sum);
```

```
}
```

```
}
```

12.WAP to check whether number 13 is prime number or not

```
class PrimeNumber
{
    public static void main(String args[ ])
    {
        int num = 13;
        boolean flag = true;
        for(int i = 2; i < num; ++i)
        {
            // condition for nonprime number
            if(num % i == 0)
            {
                flag = false;
                break;
            }
        }
        if (flag==true)
            System.out.println(num + " is a prime number. ");
        else
            System.out.println(num + " is not a prime number. ");
    }
}
```


OOPs

OOPs:-

To improve code readability and reusability by defining a java program efficiently.

Object-oriented programming (OOPs) is a methodology that simplifies software development and maintenance by providing some rules.

- 1 Data Hiding
2. Data abstraction
3. Encapsulation

Tightly encapsulation

4. Inheritance (IS-A relationship)
5. Has-A relationship (Aggregations | composition)
5. Polymorphism

Overloading

overriding

→ Data Hiding:

Data hiding means our internal data should not go out directly, after validation only if the user is valid then allowed to access the data is the mechanism of data hiding.

In real time we can implement data hiding using private modifier.

→ Data abstraction:

Data abstraction means by hiding internal implementations just highlighting the set of business services called data abstractions.

→ Encapsulations:

It is a process of wrapping a data into a container called class.

→ Tightly encapsulation:

A class is said to be tightly encapsulation if and only if all variables in that class should be of type private called tightly encapsulation.

→ Inheritance:

In java inheritance is also called as IS-A relationship.

- It is the First Pillar of OOPS.
- Inherit means acquire, possess, access, take, etc.
- One class acquiring the properties of another class is called as Inheritance

Or

- One class is accessing the properties of another class is called as Inheritance.
- Here properties is defined as methods and variables.

→ **WORA MECHANISM: -**

- Write Once Read/ Reuse Any (no. of times)
- We should not duplicate code
- Code reusability.

→ In java we can implement inheritance by using **extends** Key word.

Using inheritance, we can get data from one class to another class.

Class classname extends classname (classname of other)

→ This inheritance is applicable at two levels.

1. Class level

2. interface level

To get code reusability.

Through inheritance whatever the parent having by default available to every child.

- Whenever we create the parent class object on that parent class reference we can create or we can access parent class method.
- Whenever we create child class object on that reference we can access both parent and child class method.
- Creating child class object using parent class reference

Fp_men fp_men = new Fp_women()
(parent class) (child class)

Whenever we are creating child class object using parent class reference on that parent class reference, we can access only parent class method but not child class methods.

Note: Creating parent class object by using child class references is not possible

Types of Inheritance: -

- 1.single level inheritance
- 2.multilevel inheritance
- 3.multiple level inheritance
- 4.hybrid inheritance

→ single level inheritance:

If a class is extending only one class at a time

Example: class a extends class b

→ Multilevel inheritance:

If a single class extending multi classes at a time

Example: class a extends class b extends class c extends class c

→ Multiple level:

A single class extends more than one class but not possible in java

A class cannot extends more than one class.

Polymorphism:

Representing one thing in many forms is a mechanism of polymorphism.

- 1.Overloading
- 2.over riding

Over Loading:

Overloading is also called as static polymorphism.

Early binding.

Overloading is applicable at two levels

- 1.Method level (same method name with different parameters)
- 2.constructor level.

In overloading method resolution happens over on reference type and method execution happens on reference type.

(method resolution)	(method execution)
<code>overload test = new overloadtest();</code>	
(reference type)	(runtime object)

overloading is a concept which is applicable in the part of polymorphism.

over Riding:

It is also called as runtime polymorphism or dynamic polymorphism.

--> If the child is not satisfying with parent parent class implementation, then child class has the ability to override the methods with their own implementations is the mechanism of overriding.

- we cannot override the static methods.
- method signature should be always same.

Annotations:

In java annotations is a program which performs some specific tasks.

In java every annotation starts with @

@Override

@supress warnings

@Functional interface etc...

- We cannot override final method.
- we can override a non final method as final.
- we cannot override static method(static to non-static cannot override).

@override annotation indicates the method or statement is overriding.

@supress warnings annotation is used to ignore warnings.

@functional interface:

An interface is having exactly one abstract method, and it can have any no.of implementations.

- If you want to convey or restrict this to functional interface we use annotations @functional interface
- If you are trying to take more than one method after annotation you will get a compile time error.

final finally finalize:

final is a key word which is applicable at

1. class level
2. variable level
3. method level
4. enums level

Whenever we declare final of above levels we cannot extends or override or change or reassignment.

Method Hiding:-

If you want to implement method hiding in java both methods should be static (parent and child).

Method hiding is used to stop the overriding.

In method hiding method resolution happens over on reference type method execution also happens at reference type.

Abstract: -

Abstract is a fixed key word which is applicable at class level, method level and interface level.

- whenever you declare a class as abstract, then we cannot create object for that class.
- If we try to create object for that class, we get compile time error.
- we cannot override the static methods.
- By declaring class as abstract then that class is known as read only class.
- We can access through only class name.

When do we go for abstract at method?

whenever we know the service but not implementation then we go for abstract to implement the service in future.

Syntax:

```
abstract <as> void | return type method name(---);
```

Note: it doesn't contain any method body.

- whenever a class is having minimum one abstract method then compulsory we must and should declare the class as abstract.
- child class is responsible to provide implementations of abstract methods.

Syntax to create object:

```

1      classname  variableName  = new classname( );
2

```

1==> classname can be abstract /non abstract.

2==>should be non-abstract.

Interface: (1.7 and below versions)

Interface is a service copy where we are going to maintain all business requirements or business services in the form of abstract methods.

Syntax:

```
interface    <interface name>
{
}

```

- By default every interface becomes abstract.
- By default every method becomes public abstract.
- Interface can contains methods and variables.
- By default every variable becomes public static final.

implements is a keyword to use the implementation in a child class.

From **Java 1.8 version** inside the interface we take implementation in the form of default methods and static methods.

Interface static method:

Whenever the business implementations are constant which are not going to be changes and no body allowed to change the behaviour of that then we go for static method.

Syntax:

```
<as> static void | return type methodname(--)  
{  
    ;  
    ;  
    ;  
    return statement;  
}
```

- interface static methods will not available for child classes.
- we cannot override these static methods inside the implementation classes.
- we can access these methods only through classname or interface name.

Interface default method:

Whenever the business implementations are not constant which are going to be changes then we go for default method.

- default Methods available for child classes.
- we can override the default methods.

Syntax:

```
default void | return type methodname(---)  
{  
    ;  
    ;  
    ;  
    return statement;  
}
```

- **default** is a key word which introduced in java 1.8 version which is only applicable at interface level but not class level.

- we can access default methods through child class.
- we cannot access default methods through interface name.
- while overriding replace default key word with **public**.

From **Java 1.9 version** onwards inside the interface we can take private methods which is of type static | non-static.

Types of interfaces: -

1. Marker interface
2. functional interface
3. partially abstract interface
4. fully abstract interface

Marker Interface:

An interface is said to be marker interface if the interface is having 0 (zero) methods.

> we have predefined marker interfaces.

- cloneable
- serializable
- Random access

example: interface Demo

```
{
  

}
```

In above example we have zero methods so it is markers interface.

Advantage: - *By implementing the marker interface in our jvm will get the capability. like whenever our class is implementing cloneable interface that class will get the capability to create the duplicate object.*

Functional Interface:

An interface is having exactly one abstract method, and it can have any no. of implementations.

- If you want to convey or restrict this to functional interface, we use annotations @functional interface
- If you are trying to take more than one method after annotation you will get a compile time error.

Partially abstract interface:

An interface is having more than one abstract method. Means it can have any no. of methods and implementations.

Fully abstract interface:

An interface is having only one abstract method such kind of interface is called as fully abstract interface.

It never talks about implementation.

How to perform initialization of variables:

1. Direct initialization
2. Through Object
3. Through console
4. Through constructor

Direct Initialization:

Declaration and initialization in a single line called Direct initialization.

```
int num1 = 8;
```

Through Object:

```
Class name demo/variable = new classname( )
demo. var;
```

Through Console:

As an end user I'm going to enter the values and then I'm going to read those values from the console and assign it to variables.

To do that we have a predefined class called **scanner class**

Syntax:

```
Class scanner
{
```

```
Public scanner (-)
```

```
{  
}
```

From console to get the string value we are using next () method i.e.,

```
Public string next ( );
```

```
Public int nextInt( ); // To get int values.
```

```
Public float nextfloat ( ); // To get float values.
```

```
Public double nextdouble ( ); // To get double
```

```
Scanner scanner = new scanner (system.in);
```

```
System.out.println(“ ”);
```

```
int value1 = scanner.nextInt( );
```

Through Constructor:

We can perform initialization of variable

There are two types of constructors.

1. Default constructors
2. Parameterized constructor

Default constructors:

A constructor with zero parameters are called default constructor.

```
<as> constructorname()
```

```
{
```

```
;;;;;;;;;;
```

```
}
```

Parameterized constructor:

A constructor with parameters is called parameterized constructor.

```
<as> constructorname(datatype varname1, datatype varname2)
```

```
{
.....
}
```

When the constructors are going to be executed?

Whenever you create the object automatically the constructors are going to be executed.

- Whenever our class is having zero constructors or not having at least one constructor compiler generates default constructor.
- To define parameterized constructor the constructor's name should be same as classname.

this, super

this

- this is a key word
- By using this we can refer current class variables and current class methods.
- This can be used anywhere but except inside a static context means we should not use inside the static blocks and static methods.

this() constructors

- this() ➔ which refers current class default constructor.
- this(5) ➔ which refers current class 1 int parameterized constructor.

this("java",56.23) ➔ which refers current class 1 string1 double constructor.

Super

- super is a key word
- Which refers parent class variables and parent class methods.
- We can use anywhere except static context like static blocks and methods.

Super () constructors

- Super() ➔ which refers parent class default constructor.
- Super(5) ➔ which refers class1 int parameterized constructor.

Super("java",56.23) ➔ which refers parent class 1string 1 double constructor.

Assume that we have a class like

Class Demo

```
{  
}
```

If we compile the above class after compilation

Class demo extends object

```
{
```

Public Demo()

```
{
```

Super ();

```
}
```

```
}
```

- By default, every java class extends object.
- Inside the class if we don't have any constructor, then our compiler is going to place the default constructor.
- In side the constructor the first statement should be always either super () or this ()
- If you don't place anything by default super(); is placed.
- We should not write both this() and super() constructor, if we specify both we get compile time error.

Instance Block and static Block:

Static Block:

Before executing the program if you want to perform pre-execution activity then we can go for static block.

- Static block is executed when class loading by JVM automatically executed.
- In a class we can write any no. of static blocks.

Syntax:

```
Static  
  
{  
  
; ; ; ; ;  
  
}
```

Instance Block:

If you want to perform other than initialization activity then we go for instance.

- Whenever we create the object automatically instance blocks are going to executed.

Inner classes and Inner Interfaces:

When you are want to group all the related things as a single entity then we go for inner class and inner interfaces.

- Weather you declare or not by default every interface becomes public static.

Arrays

By using arrays we can hold or represent or we can define group of elements as a single entity.

- If you want to represent a group of homogeneous (group of elements as a homogeneous) as a single entity then we go for arrays.

For example: if we want to define 10 variables like

num1=1, num2=5, num3= 4,.....num10= 27;

(it is difficult to define length of code will be large)

Generally, Array index starts with 0 (zero).

Different types of arrays: -

1. Single dimensional array
2. Two-dimensional array
3. Multi-dimensional array

➔Single dimensional array:

Single dimensional arrays can hold group of elements(homogeneous).

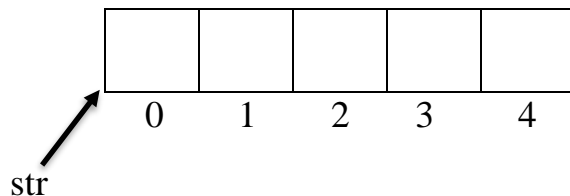
Syntax:

Datatype [] variablename = new datatype [size];

Whereas - [] ➔indicates single dimensional array.

String [] str = new string[5];

Whereas- string object is created with size is 5 (0 to 4)



If value is not assigned default values executed.

Example:

```
Public class classname
{
Public static void main(string[] args)
{
String[ ] str = new string[5];
for(int i=0; i<str.length; i++)
{
System.out.println(str[i]);
}
}
}
```

- Arrays can hold homogeneous. If we assign heterogeneous elements we get compile time error.
- In array we can have only one method length
Str.length or int.length
- In arrays we don't have any predefined API's
- In arrays we have fixed size we cannot change as per our requirements.

Example:

```
int[ ] in = new int[9];          //declaration
in[0] = 456;
in[1] = 91456;
in[2] = 1;
in[3] = 24;
in[4] = 91456;
in[8] = 25;
for(int i=0; i<in.length; i++)
{
```

```
System.out.println(in[i]);\n
}
```

Now we want to do it initialization and declaration in a single line.

```
datatype[ ] varname = {list of values separated with comma}
string[ ] str1 = {"java","selenium","hardwork","job"}
```

➔ *Two dimensional array:*

If you want to represent data in rows and columns then we go for two dimensional array

Syntax:

```
datatype[ ][ ] varname = new datatype[rows][columns]
```

➔ [][]—indicates two dimensional arrays

4 combinations (00,01,10,11)

00	01
10	11

Example: `string[][] str = new string [2] [2];`

`Str[0][0] = "java";`

`Str[0][1] = "selenium";`

`Str[1][0] = "hardwork";`

`Str[1][1] = "job";`

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

`{`

`for(int j=0; j<str.length; j++)`


```

        {
            System.out.println(str[i][j]);
        }
    }

```

- Either it is single or multi-dimensional problems are same
- It can hold same type or homogeneous.
- Size is fixed based on requirements we cannot change.
- Do not have any API's
- Only one method length.

➔ **object arrays:**

It holds both homogeneous and heterogeneous

By using object arrays we can use both homogenous and heterogeneous elements.

```

Object[ ] obj = new object[4];
Obj[0] = 456;
Obj[1] = "job";
Obj[2] = 1;
Obj[3] = "hard work";

```

But rest of problems are same

To overcome all these problems, we go for collection frameworks.

FRAME WORKS

Collection Framework

Collection Framework:

To overcome all the problems in arrays we are going to collection framework.

- Collection framework is coming from **java.util package**.

Advantages:

- It can hold both homogeneous and heterogeneous elements.
- Size is dynamic, based on requirements we can increase or decrease.
- We have bunch of API's and no. of methods.
- Bounded and unbounded.

➔Collection framework is a group of classes and interfaces. Which is coming from java.util package.

➔Our collection frameworks start with one root interface called collection interface.

➔It contains three child interfaces

1. List(I)
2. Set(I)
3. Queue(I)

➔Collection is an interface which is root for collection framework.

List (I):

➔List is an interface which is child interface of collection.

➔List can hold list of groups of values that can be both homogeneous and heterogeneous with duplicates.

SET(I):

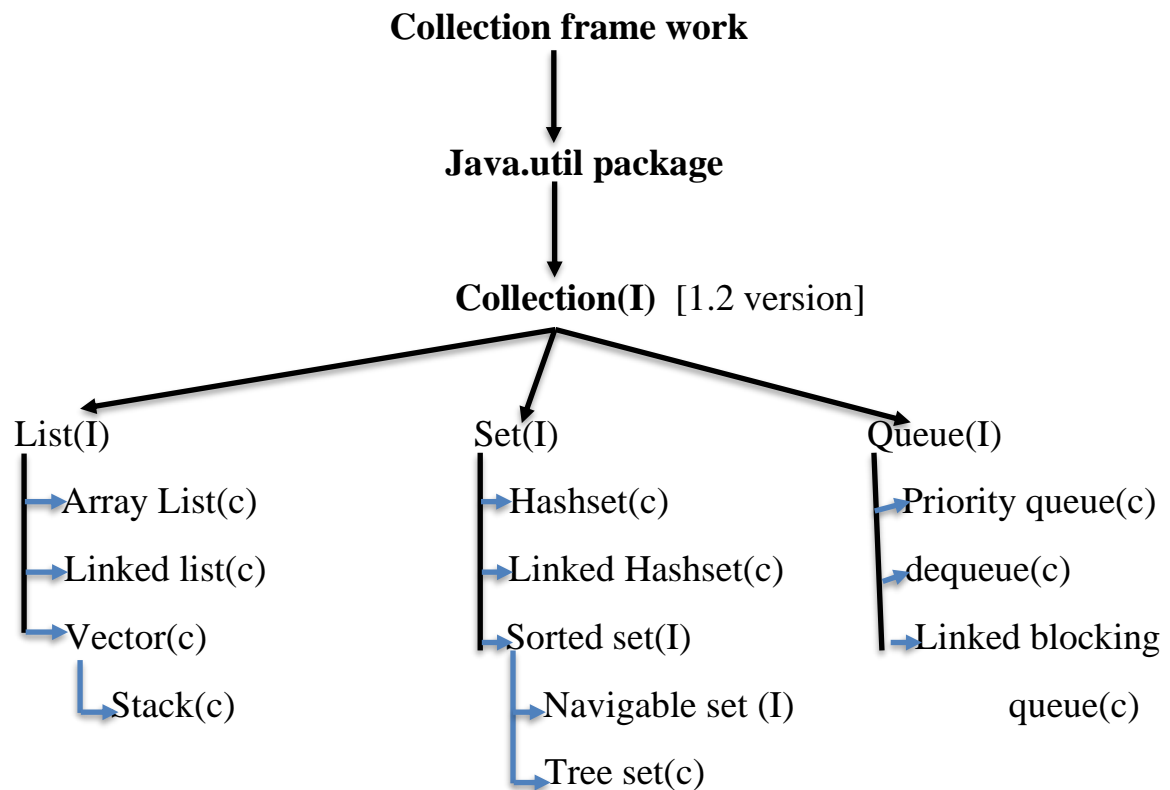
➔set is an interface which is child interface of collection(I)

➔set can hold set of groups of values that can be both homogeneous and heterogeneous without duplicates.

QUEUE(I):

→Queue is an interface which is child interface of collection(I)

→Queue can hold set of groups of values that can be both homogeneous and heterogeneous with first come first out priority.



Versions:

- Collection interface → 1.2 version
- List interface → 1.2 version
 - Array list class → 1.2 version
 - Linked list class → 1.2 version
 - Vector class → 1.0 version
 - Stack class → 1.0 version
- Set interface → 1.2 version
 - Hash set class → 1.2 version
 - Linked Hash set → 1.4 version
 - Sorted set interface → 1.2 version
 - Navigable set interface → 1.4version
 - Tree set Class → 1.2 version
- Queue interface → 1.2 version

Interface Collection:

```
interface collection
{
    boolean add(object obj);          // To add an element to object if it is added
                                     // successfully return true or else return false.
    boolean addAll(collection c);     // To add some group of elements
                                     // if it is successfully added return true if not false.
    boolean remove(object obj);       //remove element
    boolean removeAll(collection c);  //remove all elements.
    int size( );                      // to find how many elements(To find size)
    boolean isempty( );               // to check element is empty or not
    void clear( );                    // to clear
    boolean cotains(object obj);      // to check element is there or not.
    boolean containsAll(collection c); //to check all elements is there or not.
    Iterator iterator ( );            // in collection object if we have n no.of
                                     // elements to get one by one
    Object [ ] to array( );           // convert object array to array.
    ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
}
```

List interface

```
Interface list extends collection
{
    Void add(int index, object obj);  //to add element in index
    boolean addAll(int index, collection c); // to add group of elements
    object remove(int index);         //to remove elements in index
    object set(int index, object new); // to replace element in index
    List Iterator(is iterator());     // to get one by one element in index
}
```

```

Object get(int index);           //to get one element in index if you pass
                                elements it will give in form of object
}

```

- We can differentiate duplicate by index mechanism
- 1st implementation class of list is array list(c)

Arraylist:

Array list is class which is implementation class of list.

- It allows both homogeneous and heterogeneous elements
- It allows duplicate values.
- We differentiate duplicate values by index mechanism.
- Allows null values
- In array list insertion order is preserved
Insertion order means how you added element in same way we get value called insertion order mechanism.
- In array list we can pick elements with same speed.

Class ArrayList implements List, cloneable, serializable, random access

```

{
Public arraylist ()
{
// it creates empty array list object with size 10.
}

```

```

Public ArrayList (int initialcapacity)
{
// it creates empty array list object with size 10.
}

```

```

Public arraylist (collection c)
{
,,,,,,,,,,,,,
}

```

- ArrayList is a class which implements list.
- Whenever it implements cloneable interface that class will get the capability to hold duplicate object
- Whenever class implements serializable interface that class will get the capability to save the state of an object.
- Whenever class implements random access interface we can pickup elements with same speed.

➔ we have constructors in arraylist

Whenever my intension is to hold a group of elements where I want to maintain duplicate with insertion order then go for arraylist.

➔ ArrayList is the best choice whenever our frequent operation is retrieval or storage with homogenous elements with duplicates.

Linkedlist:

It is a class which is implementation class of list interface.

- It allows both homogeneous and heterogeneous elements
- It allows null values
- It also allows duplicate values based on insertion.
- Linked list is the best suitable whenever our operation is deletion or insertion in the middle at runtime we can go for linked list.
(But mostly we use array list)

[Vector and stack are not using now as it is old versions (1.0)]

Vector:

Vector is a class which is implementation class of list interface.

- It allows both homogeneous and heterogeneous elements.
- It allows duplicate values
- Duplicate values are differentiate based on index mechanism
- Vector version is 1.0

Syntax:

```
Class vector
{
Public vector ( )
{
// size is 10
}
```

```

Public vector(int initial capacity)
{
//10
}

```

```

Public vector (int initial capacity ; int incremental capacity)
{
//capacity size is 10
}

```

```

Public vector (collection c)
{
//size 10
}

```

In vector we have some specific methods.

```

Void addeElement(object obj);
Void removeElement(object obj);
Void removeElementAt(int index);
Void removeElement( );
Object firstElement( );
Object lastElement( );
:::
}

```


This Vector having one more child class called Stack

Stack:

Stack is a class which is child class of vector

- Follows Last in First Out mechanism

Class stack extends vector

```
{  
//LIFO  
Public stack()  
:  
;;;;  
}
```

In stack we have few methods

Object push (object obj)

Object pop () //remove and returns top of the stack

Object peek () //returns top of the stack without removal

boolean empty ()

```
.....;  
}
```

Set Interface:

- Set is an interface which is the child interface of collection interface.
- Whenever you want to hold the group of elements or group of objects that can be both homogeneous and heterogeneous and without duplicates.
- Set interface does not have any methods we need to use collection interface methods only.

First implementation class of set interface is HashSet(c)

HashSet: (1.2 version)

→ HashSet is a class which is implementation class of set interface.

Interface set extends collection

```
{  
    // no methods  
}
```

→ HashSet allows both homogeneous and heterogeneous elements.

→ HashSet doesn't allow duplicate elements.

→ If you're trying to take any duplicate we won't get any compile time error. Simply the add method returns the false.

→ In HashSet insertion order is not preserved.

→ HashSet allows null values.

→ Default initial capacity is 16 and default load is 75%

To create object

```
Set set = new Set(); // invalid
```

```
Set set = new HashSet<>(); // valid (initial capacity is 16 and load is 75)
```

```
HashSet hs = new HashSet();
```

Linked HashSet:

- It is child class of HashSet
- Introduced 1.4 version
- Linked HashSet allows both homogeneous and heterogeneous elements.
- It doesn't allow duplicate values.
- Insertion order is preserved.
- It allows null values.

Sorted Set interface:

- It is child interface of set interface
- Sorted set is specially designed for sorting techniques that can be ascending order or descending order.

Navigable set interface:

- It is a child interface of sorted set
- Specially designed to perform navigation operation.

Treeset (c)

- TreeSet is a class which is implementation class of navigable set, sorted set and set interface.
- It allows only homogeneous elements.
- It doesn't allow heterogeneous element.
- Specially designed to perform sorting techniques.
- It doesn't allow null values (if you enter any null values , then compilee will shows you the null pointer exception.)

Four constructors in Tree set

Class TreeSet

```
{  
    Public TreeSet()  
    {  
        //which follows ascending order(comparable)  
    }  
    Public TreeSet (int init)  
    {  
        //which follows ascending order  
    }  
    Public TreeSet (collection c)  
    {  
        //which follows ascending order  
    }  
    Public TreeSet (comparator comparator)  
    {  
        //which follows customised sorting order  
    }  
    .....  
}
```

→ whenever you execute default constructor ascending order will performed.

Comparable interface:

- It is an interface coming from java.lang package

```
interface comparable
{
    int compareTo(Object obj);
    boolean equals(-)
}
```

- Having two methods : CompareTo and equals method.
- Specially designed to perform sorting techniques with ascending nature.
- Every predefined class implemented this comparable interface.

Comparator interface:

- It is an interface coming from java.util package
- In comparator, we have method compare inside we write custom logic
- Whenever the interface is having one abstract method such kind of interface is called functional interface.
- If you want custom sorting nature your class must and should implement this comparator interface

```
Interface Comparator
{
    int compare (Object obj1, Object obj2);
}
```

Queue interface:

It is an interface which is child interface of collection interface. If you want to hold a group of objects, prior to processing we can go for queue

➔ Queue doesn't allow heterogeneous elements.

➔ Follows First in first order (FIFO)

```
interface Queue
{
    boolean offer(Object obj);    // we can add one element(object to queue)
    Object remove();    // To remove elements & returns head of element
    // Queue. If queue is empty return no element exception
    Object peak();    // It returns head of element of queue. if queue is empty
    // Returns null.
```

Object element(); // it returns head of element of queue.if queue is

Empty Shows no such element

```
.....  
}
```

Priority Queue:

- It is a class which is implementation class of Queue Interface.
- We can add both homogeneous and heterogeneous element.
- Duplicates are not allowed.
- Insertion order is not preserved.
- Null insertion also not at all possible.

We cannot create object for interface

By using interface reference name we can create object

//Queue queue = new Queue(); NOT VALID

//Queue queue = new priorityQueue(); VALID

PriorityQueue queue = new priorityQueue();

queue.add(456);

queue.offer(91456);

queue.add(1);

CURSORS:

In java we have a concept called cursors to iterate the elements / to get the elements one by one from the collection object.

Types:

1.Enumeration

2.List Iterator

3.Iterator

Enumeration:-

Enumeration is an interface.

- By using enumeration cursor, we can retrieve the elements one by one from collection object. But enumeration cursor is only applicable for legacy classes (1.0 version)
- Using enumeration cursor, we can retrieve the elements only from legacy collection objects.

```

Interface Enumeration
{

}

```

From_which_legacy_collection_objet. [you want to retrieve the object on that we call elements]

```
Enumeration enumeration = from_which_legacy_collection_object_elements();
```

```
Vector vector = new vector();
```

```
Vector.add(456);
```

```
Vector.add(1);
```

```
Vector.add(91456);
```

```
Enumeration enumeration = vector.elements();
```

```
While[enumerations.hasMore Elements()]
```

```
{
```

```
Object obj = enumeration.nextElement();
```

```
Integer I =(Integer) obj;
```

```
If(i==1)
```

```
{
```

```
System.out.println(“we are no1”);
```

```
}
```

```
System.out.println(i);
```

```
}
```

Problem of enumeration is applicable for legacy classes.

LIST ITERATOR:

List iterator is an interface

- Iterating the values one by one from list types that can be ArrayList, Linked List, Vector, Stack but not from set, Queue,maps.
- Limitations with list iterator is we can retrieve the elements from only List Types.

```
ListIterator itr = from_which_list_collection_object.listIterator();
```

```
Interface ListIterator
```

```
{  
    boolean hasNext();  
    object next();  
    ;;;;;;;  
}
```

ITERATOR:-

Iterator is an interface.

- By using iterator we can retrieve the elements one by one from any collection object.

```
From_which_collection_object.iterator();
```

```
Iterator itr = From_which_collection_object.iterator();
```

```
Interface Iterator
```

```
{  
    boolean hasNext(); //To check whether element is there or not  
    object next(); // To give element  
    ;;;;;;;  
}
```

For each loop

- We can retrieve the elements one by one from arrays and any collection objects.
- It is also called as enhanced for loop

Normal for loop

```
for(init section; condition; inc|dec section)  
{  
    ;;;;;  
}
```

Enhanced for loop

```
for(source)
{
    ;;;;;
}
```

(Source = what type of elements in source name : source)

Source – arrays/any collection object.

Whenever you add an element to the collection object it become of object.

- Object can hold anything
- Object obj = 456; valid
- Object obj1 = “java”; valid
- String str1 = “selenium”; valid
- Object obj2 = str1; // implicit type casting. Compiler will take care of it.
- String str2 = (string) obj1; //Explicit type casting programmer will take care of it.

TYPE CASTING:-

Converting from one form to another form is known as type casting.

There are two types type casting.

1.implicit type casting

2.explicit type casting.

IMPLICIT TYPE CASTING

- Compiler will take care of implicit type casting
- Whenever we are assigning smaller data to bigger datatype then implicit type casting will come into picture.
- No loss of data.

EXPLICIT TYPE CASTING

- Programmer is responsible for explicit type casting
- Whenever we are assigning bigger data to smaller data type then explicit type casting takes place.
- Loss of data.

String Str2 = (String) Obj1
 1 2 3 4

Rule1: 3 and 4 should have some relationship either child to parent or [aren't to child.

Rule2: 1 and 3 should be same type or 1st should be the parent of 3rd one.

MAP:-

- It is an interface. Which is not child interface of collection.
- By using map we can represent or hold data in the form key value pairs.

Key	=	value	
Practise	=	job (Entry1)	} Entry set
Salary	=	120000 (Entry2)	
Name	=	Ramesh (Entry3)	

Interface Map

```
{
Object put(Object key, object value) // to add an element
Void putAll(Map map); // to add group of elements
Object get(Object key); //to get one value if you pass value it will get in
                           form of object. If no key = null
object getDeafultvalue(object key, object defaultvalue);
object remove(Object key); // to remove key
boolean containskey(object key); //to check key is there or not
boolean containsvalue(Object value) //to check value is there or not
void clear();
;;;;;;;;;
boolean is Empty()
int size()
set keyset()
set entryset()
```

```

collection values();
,,,,,,,,,
interface Entry //public static
{

}

}

```

HashMap:- (1.2v)

First implementation class of map interface

- In HashMap class we have constructors
- Default constructor, capacity class HashMap implements MAP

```

Public HashMap ()
{
    // one HashMap object is going to created with size 16 / capacity 75%
}

```

```

Public HashMap (int Initial capacity)
{
    //To specify my own capacity object
}

```

```

Public HashMap (Map map)
{

}

```

```

Public HashMap (int initial capacity, float loadfactor)
{

}

```

- It allows both homogeneous and heterogenous elements
- Takes the values in the form of key value pairs
- Keys cannot be duplicate but values can be duplicate.

- If you are trying to duplicate the keys old values are going to replace with new values.
- Hash Map allows null values.
- Insertion order is not preserved.

Linked HashMap:-

- It is a class which is child class of HashMap
- Introduced in java 1.4 version
- Allows both homogenous and heterogeneous elements.
- It takes values in the form of key value pairs; keys cannot be duplicate values can be duplicate.
- Allows null values
- Insertion order is preserved.

Concurrent Hash Map:-

- It doesn't allow null values
- Insertion order is not preserved
- In concurrent HashMap null is not allowed in any keys/ values.

WeakHashMap:-

- Allows both null values

Sorted Map:-

- It is an interface
- Specially designed to perform sorting techniques (ascending or descending)

Navigable Map:-

- Navigable is an interface.
- Specially designed to perform navigation operation.

TreeMap:-

- Tree map is an implementation class of map interface.
- Allows only homogeneous type of elements
- Specially designed to perform sorting techniques.

Hashtable:-

- It is a class which is implementation class of map interface
- Introduced in 1.0 version
- Allows both homogeneous and heterogeneous elements.
- It doesn't allow null values
- Allows duplicate values but not keys (old value replaced with new)

Utility classes:-

In list we don't have sorting mechanism so that we use utility classes.

- Arrays
 - collections
- } Java.util package

Arrays: - in arrays utility class we can perform sorting techniques that can be ascending or descending and we can also perform searching techniques.

```
public class classname {  
    public static void main(string[] args) {  
        String[] str={"hello","java","job"}  
        Arrays.sort(str);  
        For(String str1: str)  
        {  
            System.out.println(str1)  
        }  
        System.out.println(arrays.binarysearch(str,"java"));  
        Arrays.asList(str);  
    }  
}
```

In order to get elements in a ascending or descending order. If we perform iteration it will get one by one to get ascending pre-defined class Array.sort .

For searching elements binary search if elements present returns +ve if not -ve.

Collections:- we have method called sort method based on comparable

```
Collection.sort(list)
```

- in collections framework we can add any type of elements
- those are not type safe
- while retrying we should do typecasting.

To overcome minor problem in collections like we can add any type of elements and while retrieving we should do type casting.

Generics:

- to get type safe (means homogeneous)
- To resolve typecasting issues.

`anycollectionobj<type> varname = new anycollection obj<type> ();`

`anycollectionobj<type> varname = new anycollection ();`

`anycollectionobj<type> varname = new anycollection obj< > ();`

ex:

`List<String> list1 = new Array list<String>();`

`List<integer> list1 = new Array list<integer>();`

`List<integer> list1 = new Array list();`

`List<integer> list1 = new Array list<>();`

`Map<String, float> map1 = new HashMap<String, float>();`

Java Lombok

Java Lombok is a java API

Suppose in a program we have 200 variables for those 200 variables we need to write 200 getter methods and setter methods.

To overcome that we use Lombok

Download the Lombok jar using the following link

<https://projectlombok.org/download>

Now at the project configure the build path add Lombok jar file.

Lombok is having annotations

@Data

Whenever we annotate a class with **@Data** how many variables are there in the class for all those variables it's going to generate getter and setter methods.

For static variables it's not going to generate the setter methods.

So, for static variables we should specify explicitly

@Getter, @setter

Example: **@Getter @setttter** private static string hardwork;

Suppose we want to generate NoArgs constructor then we need to annotate at class level

@NoArgsConstructor then it's going to generate default constructor.

@AllArgsconstructor then it's going to generate parameterised constructor.

Java.lang PACKAGE

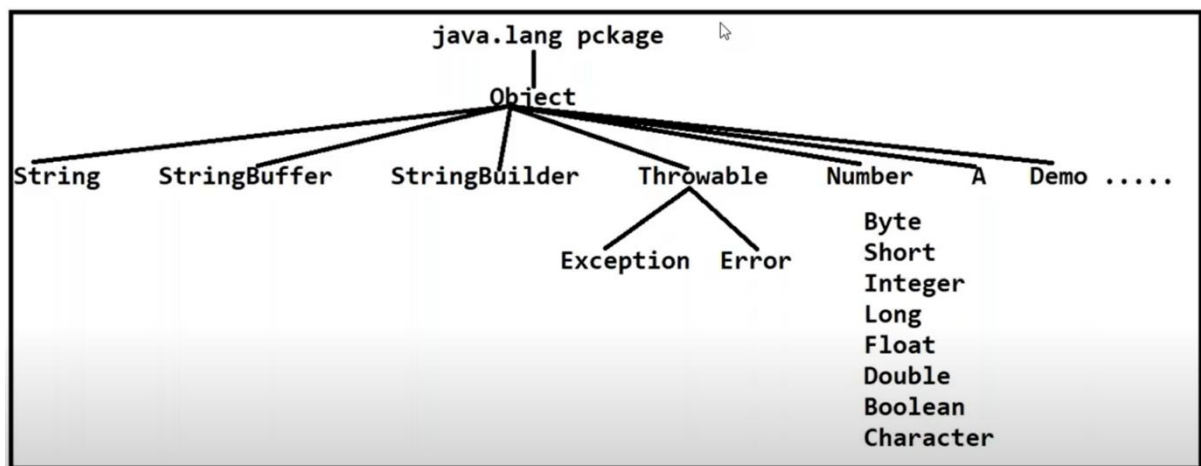
In java default package is java. lang package.

Whatever the classes you are using from java. lang package we are not required to import that class.

Other than java. lang package we have N no. of packages like:

Java.io
java.util
java.util.regex
java.time
.....
etc.....

If you are using the classes from above listed packages, we need to import compulsory.



In java.lang package our class is object. If you extend or not by default every java class extends the object class. For all java classes object is the parent class.

This object class is having number of child classes like String, String Buffer, StringBuilder, Throwable, Number etc...

In this object class we have 11 methods

```
java.lang package

class Object
{
public int hashCode()
public String toString()
public boolean equals(Object obj)
public void wait(-)
public void wait(--)
public void wait(---)
public void notify()
public void notifyAll()
public void finalize()
public Object clone()
public Class getClass()
}
```

in this object class we have 11 methods are there.

hashCode ():

In Java whenever we create the object. Internally for every object a unique number is going to be assigned that unique number is hash code. That returns in the form of integer.

toString ()

If you want to get string representation of an object, we have method called toString () method.

equals (Object obj)

to check whether two objects are equal or not we have equals () method.

Wait(-), wait(--), wait(---)

To wait for a thread, we have wait () method.

notify (), notifyAll ()

to get notification we have method notify () [but we are not going to use it]

finalize ()

In java programmer is responsible only for creation of object.

So, who is going to destroy the object is optional. If we want, we can destroy the object or by default jvm is going to destroy.so, jvm called a component called garbage collector before destroying an object jvm calls finalize method.

Clone ()

To create a duplicate object, we have method called clone () method.

getClass()

if we want runtime definition of a class, we have method called getClass() Method.

```
1 package secondapp;
2
3 public class ObjectDemo1 {
4
5     @Override
6     public String toString()
7     {
8         return "do hardwork";
9     }
10
11 public static void main(String[] args) {
12
13     ObjectDemo1 demo1 = new ObjectDemo1();
14     //System.out.println(demo1.hashCode());
15     //System.out.println(demo1.getClass().getName());
16     System.out.println(demo1.toString());
17     System.out.println(demo1);
18
19
20
21 }
22 }
23 }
24 }
```

```
1 public class CloneDemo implements Cloneable{
2     int id;
3     String name;
4
5     public void hello() {
6         System.out.println(id+" "+name);
7     }
8
9 public static void main(String[] args) throws CloneNotSupportedException {
10
11     CloneDemo demo = new CloneDemo();
12     Object obj = demo.clone();
13     CloneDemo dup = (CloneDemo) obj;
14     dup.id = 91456;
15     dup.name = "JOB";
16     dup.hello();
17
18
19     System.out.println(demo.hashCode());
20     System.out.println(obj.hashCode());
21     System.out.println(dup.hashCode());
22
23 }
24 }
25 }
26 }
```

we perform cloning mechanism to maintain Backup.

Instead of performing operation on original object we do on duplicates.

Cloneable is an interface which is a marker interface (means if the interface is having zero methods such kind of interface is called marker interface). By implementing this interface our jvm will get the capability to hold duplicate object.

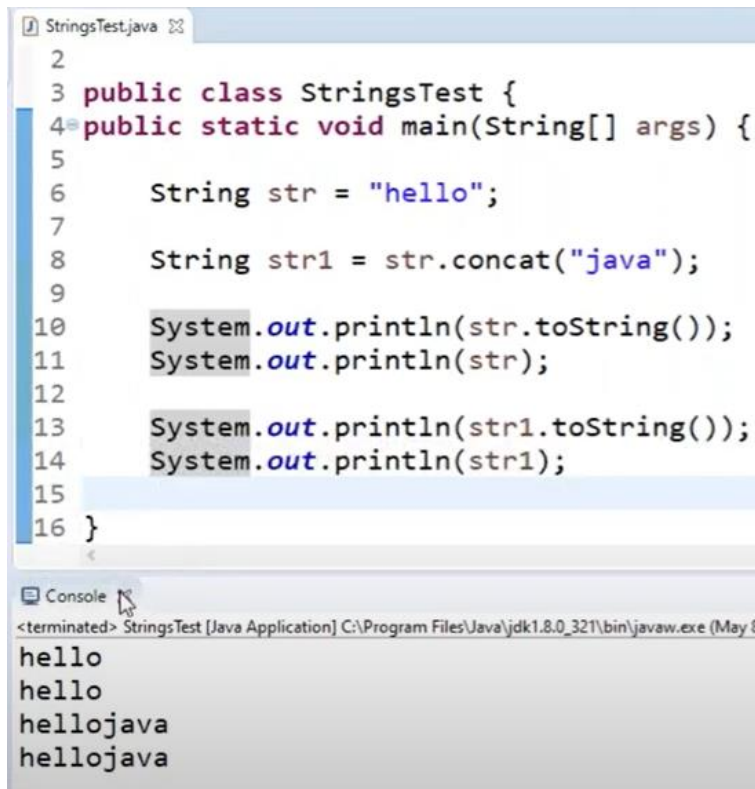
Mutable vs immutable

Mutable

- Mutable means change able
- String Buffer and StringBuilder are mutable.
- We can perform any changes in string Buffer and string Builder

Immutable

- Immutable means not changeable.
- Strings are immutable.
- Once we create string object in existing object, we cannot perform any changes. If we are trying to perform any changes with those changes one more new object is going to be created that mechanism is called immutable mechanism



```
1 StringsTest.java
2
3 public class StringTest {
4     public static void main(String[] args) {
5
6         String str = "hello";
7
8         String str1 = str.concat("java");
9
10        System.out.println(str.toString());
11        System.out.println(str);
12
13        System.out.println(str1.toString());
14        System.out.println(str1);
15
16    }

```

Console

```
<terminated> StringTest [Java Application] C:\Program Files\Java\jdk1.8.0_321\bin\javaw.exe (May 8
hello
hello
hellojava
hellojava

```

```
16
17
18
19     StringBuilder sb = new StringBuilder("java");
20     StringBuilder sb1 = sb.append("selenium");
21
22     System.out.println(sb);
23     System.out.println(sb1);
```

Console

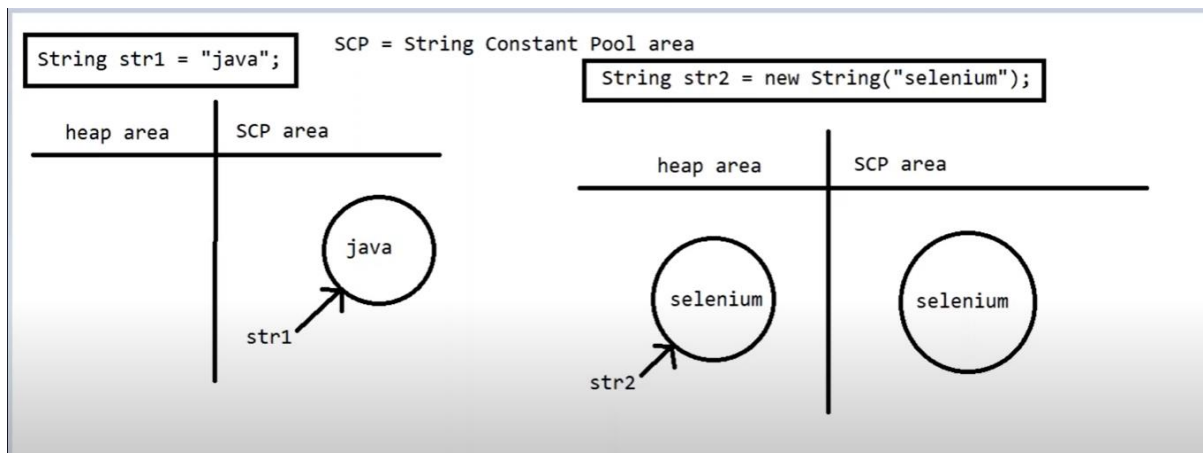
<terminated> StringsTest [Java Application] C:\Program Files\Java\jdk1.8.0_321\bin\javaw.exe (May 8, 2022, 11:01:20 AM)

hello
hellojava
hellojava
javaselenium
javaselenium

What is the difference between

String str1 = "java";

String str2 = new String("selenium");



String Class

String is a class which is child class of object. We have N number of methods.

```
public class String extends Object
{
    public char charAt(int index);
    public int compareTo(String s);
    public int compareToIgnoreCase(String s);
    public String concat(String s);
    public boolean contains(CharSequence cs);
    public boolean contentEquals(CharSequence cs);
    public boolean endsWith(-)
    public boolean equalsIgnoreCase(String a);
    public int indexOf(char c)
    public int indexOf(String a)
    public boolean isEmpty();
    public int length()
    public boolean matches(String regex)
    public String replace(char a,char b)
    public String replaceAll(String a,String b)
    public String replaceFirst(String a,String b)

    public String toLowerCase()
    public String toUpperCase()
    public String trim();
    public String substring(int index)
    public String substring(int startindex,int endindex)
    public char[] toCharArray();
    public String[] split(String regex);
    ;;;;;;;;;;;;;;
}
```

Now we will see how we can use the above methods

```
StringsDemo.java
2
3 public class StringsDemo {
4 public static void main(String[] args) {
5
6
7     String str1 = " hello java selenium demo ";
8
9     String str2 = str1.trim();
10    System.out.println(str1);
11    System.out.println(str2);
12
13    char ch = str2.charAt(2);
14    System.out.println(ch);
15
16    String[] str3 = str2.split(" ");
17    for(String str4 : str3)
18    {
19        System.out.println(str4);
20    }
21
22
23    char[] ch1 = str2.toCharArray();
24    for(char ch2 : ch1)
25    {
26
27
28
29
30    String str5 = str2.substring(3);
31    System.out.println(str5);
32
33    String str6 = str2.substring(3, 6);
34    System.out.println(str6);
35
36    String str7 = str2.replace('l', 'j');
37    System.out.println(str7);
38
```

```

37     System.out.println(str7);
38
39     String str8 = str2.replaceAll("hello", "JAVA");
40     System.out.println(str8);
41
42     String str9 = str2.toUpperCase();
43     System.out.println(str9);
44
45

```

Write a program for reverse of a string str =java

```

6
7     String str = "JAVA";
8
9     StringBuffer sb = new StringBuffer(str);
10    StringBuffer sb1 = sb.reverse();
11
12    System.out.println(sb1);
13
14
15
16
17
18
19
20

```

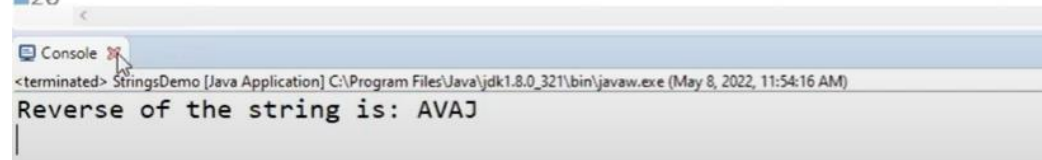
Console

<terminated> StringsDemo [Java Application] C:\Program Files\Java\jdk1.8.0_321\bin\javaw.exe (May 8, 2022, 11:49

AVAJ

Write a program for reverse of a string str =java without API

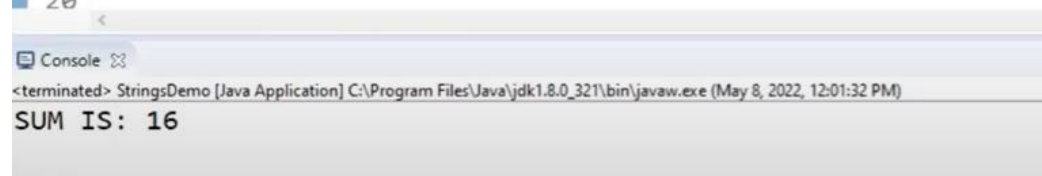
```
6
7   String str = "JAVA";
8
9   String reverse = ""; //AVAJ
10
11  for(int i=str.length()-1; i>=0; i--)
12  {
13      reverse = reverse + str.charAt(i); //AVA + J
14  }
15
16  System.out.println("Reverse of the string is: "+reverse);
17
18
19
20
```



The screenshot shows a Java IDE with a code editor and a console window. The code in the editor is a Java program that reverses the string "JAVA" by iterating from the end to the beginning and building a new string. The console window shows the output: "Reverse of the string is: AJAV".

Write a program to extract the digits from a string and do the sum

```
6
7   String str = "1ADP456";
8
9   char[] ch = str.toCharArray(); // [1 A D P 4 5 6]
10
11  int sum = 0; //16
12
13  for(char ch1 : ch)
14  {
15      if (Character.isDigit(ch1)) { // '5'
16          int val = Character.getNumericValue(ch1); //6
17          sum = sum + val; //16
18      }
19  }
20
```



The screenshot shows a Java IDE with a code editor and a console window. The code in the editor is a Java program that extracts digits from the string "1ADP456" and calculates their sum. The console window shows the output: "SUM IS: 16".

Write a Program to split the given String

```
String str1 = "java-selenium-demo-job";
String[] str3 = str1.split("-");

for(String str4 : str3)
{
    System.out.println(str4);
}
```

These are called as Primitives

byte short int long float double char boolean

These are called as Wrapper classes. All First letter should be capital

Byte Short Integer Long Float Double Character Boolean

By using wrapper classes, we can hold the data in the form of primitives and object.

In every wrapper class we have two methods

valueOf()

parseXXX()

```
WrappersDemo.java
5
6 String str = "456";
7
8
9 int i = Integer.parseInt(str);
10 System.out.println(i);
11
12 float f = Float.parseFloat(str);
13 System.out.println(f);
14
15 char ch = '2';
16 int i1 = Character.getNumericValue(ch);
17 System.out.println(i1);
18
19 Double d = Double.valueOf(str);
20 System.out.println(d);
21
22 Double d1 = Double.valueOf(d);
23 System.out.println(d1);
24
25 Integer i4 = Integer.valueOf(i);
26 System.out.println(i4);
27
28
```


What is Auto Boxing and Auto Unboxing?

Auto boxing: The automatic conversion of primitive data types into its equivalent wrapper type is known as Auto boxing.

(Primitive to wrapper object)

Auto unboxing: Converting wrapper to primitive is known as Auto unboxing.

(Wrapper object to primitive)

```
int a = 9;
Integer i = a; //auto boxing

int i1 = i; //auto unboxing
System.out.println(i1);
```

HOW TO DOWNLOAD JAVA

Open Google > enter “download java” > click on Java SE Download/oracle technology network/oracle link > click on ‘java downlaod” icon > accept liscense agreement > if your os is window x64 then download windows 64-bit JDK or if your os is windows X32 then download windows 32 JDK > once click on that you will get ‘.exe’ file > copy that file from downlaods and paste in a folder or desktop.

HOW TO INSTALL JAVA

Double click on my computer > properties > advanced system settings > environment variables > click new for user variables > enter variable name as JAVA_HOME > Enter variable value as path of JDK i.e., c:\program files\java > click on ‘ok’ > under system variables select ‘path’ variables > click “edit” > at the end of existing value put and write path of JDK/bin; > click OK.