JAVA COMPLETE SYLLABUS

follow By - @ Python_World_in

Introduction

- Prerequisites of Learning Java
- Necessity of Programming
- What is java?
- What is platform Independence?
- Where java Stands Today?
- Important Features
- History of Java
- Editions of Java
- JDK, JRE and JVM
- Compiling And Executing The Code

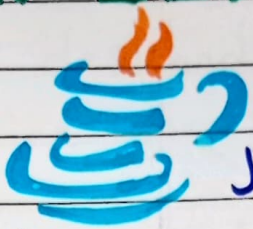
Basic Concepts in Java

- Data types
- Type Conversions
- Command line arguments
- Wrapper classes

- Control Statements
- if; if-else, nested if
- Switch, Ternary operator
- Loops Structures
- Arrays
- Types of arrays
- Classes and objects.
- Initializing objects
- Constructors
- Key words
- Static methods and blocks
- Inheritance
- Overriding and Overloading
- Base class and Derived class
- Polymorphism
- Interfaces
- Packages
- Path and classpath
- Exception Handling
- Try and Catch
- Hierarchy, Throw
- Checked and Unchecked
- String Handling
- Collections.

Python_World_in

JAVA COMPLETE NOTES



PYTHON_WORID_IN

An Introduction to JAVA

* Prerequisties of learning Java
* Necessity of Programming
* What is JAVA?
* What is Platform Independence?
* Where Java Stands Today?

Pre Requisites of learning Java

→ To start learning Java, you should be familiar with basics of Programming.

→ And since Java itself has been build up using C/C++ language so just basic knowledge in C/C++ is more than sufficient

Why do We need Programming

→ To Communicate with digital machines and make them work accordingly.

→ Today in the programming world, we have more than 900 languages available.

→ And Every language is designed to fulfill a particular kind of requirement.

Brief History of Programming Lang

→ C language was primarily designed to develop "System Softwares" like operating Systems, Device Drivers etc.

→ To remove design problems with "C" language, C++ language was designed.

→ It is an Object Oriented Language which Provides data Security and Can be used to Solve real world problems.

→ Many popular Softwares like Adobe Acrobat, winamp media player, internet Explorer, Mozilla firefox etc were designed in C++

What is JAVA ???

→ JAVA is a Technology (not only a programming language) to develop object oriented and platform independent applications.

- Platform Independence
- Technology

PLATFORM Independence

- Platform.

→ A platform is the environment in which a program runs.

→ In simple terms it is Combination of operating System and processor.

Example :- Windows + Intel (i5), Ubuntu + AMD

Platform = operating System + Processor

Q) How many physical Machines are there in the figure? Ans :- 12 physical machines.

Windows (32 bit)	Mac	Linux	Windows (32 bit)
windows (64 bit)	Linux	Mac	Linux
Windows (32 bit)	mac	Windows (64 bit)	Linux.

Q) How many platforms are there in the figure?
Answer :- Only 4

Windows (32 bit)	Mac	Linux	windows (32 bit)
Windows (64 bit)	Linux	mac	Linux
windows (32 bit)	Mac	Windows (64 bit)	Linux.

Where Java stands Today?

→ To understand how Java has dominated the market from last 25 years, please See this video. • https://youtu.be/F-rk3UgSBs

→ 3 Billion devices run java as per Oracle. (1 Billion = 100 Crores).

→ 1 Billion Java downloads per year

Types of JAVA Applications

* Web-based Applications
* Cloud-based Applications
* Distributed Applications
* Mobile Applications
* Gaming and Animation
* Digital and Electronic Devices
* Desktop Applications
* Business Applications

Important features of JAVA

- Platform independent
- Automatic Memory management
- Secure
- Robust
- Simple
- Object oriented
- Multithreaded.

* Platform Independent

A platform is the environment in which an application runs.

In other Words it is the Combination of an OS and a CPU.

for Example :-

Windows 8 + Intel - Core i5 (is a diff. platform)

Linux + AMD - A6 (is another diff platform)

Mac + Intel - Core i3 (is yet another diff platform)

* Now being platform independent means that an application developed and Compiled over one platform Can be executed over any other Platform without any change in the Code.
* And, Java has this Capability using the Concept of "bytecode" and "JVM".
* Whenever we Compile a Java program, the Compiler never generates machine Code.
* Rather it generates a machine independent Code Called the "bytecode".
* This bytecode is not directly understandable by the platform (OS and CPU).
* So another special layer of Software is required to Convert these bytecode instructions to machine dependent form.
* This special layer is the JVM, that Converts the bytecode to Underlying machine instruction set and runs it.

JAVA program to print Hello World!

```
class HelloWorld {
public Static void main(string args[]) {
System.out.Println("Hello World!");
}
}
```

Output :- Hello World!

* Thus any Such platform from which a JVM is available Can be used to execute a java application irrespective of where it has been Compiled.
* This is how java makes itself "Platform Independent" and it also truly justifies java's slogan of "WORA" (Write Once Run any where).

* Automatic Memory Management:
* In languages like C and C++ any dynamic memory which the programmer allocates Using malloc() or new has to be deallocated by himself Using free() or delete.
* But java uses runtime automatic garbage Collection feature where the JVM itself deallocates any dynamic memory which Our Program allocated.

→ **Secure** :-

When it Comes to Security, Java is always the first choice. It enables us to develop virus free, temper free System.

JAVA is a more secure language as Compared to C/C++ because:

1. It does not allow a programmer to explicitly Create pointers.
2. Java program always runs in Java runtime environment with almost null interaction with System OS, hence it is more Secure.

→ **Robust** :-

Java has very strict rules which every Program must Compulsorily follow and if these rules are violated then JVM kills/ terminates the Code by generating "Exception".

To Understand java's robustness, guess the Output of the following C/C++ Code:

```
int arr[5];
int i;
for (i=0; i<=9; i++)
{
 arr[i] = i+1;
}                    // unpredictable, after i is 5
```

→ The Previous Code might show Uncertain behaviour in C/C++ i.e. if memory is available after arr[4], then the Code will run, otherwise it will generate error at runtime.

→ On the other hand if in Java this Code is executed, the JVM will kill the application as soon as it finds the Statement arr[5]=...

→ Reason is that in Java we are not allowed to access any array beyond it's upper / lower index.

* **Simple** :-

* Java borrows most of it's Syntax from C/C++ languages.
* Moreover it has inherited best points from these languages and dropped others.
* Like, it has removed pointers, multiple inheritance etc as developers of Java language found these features to be Security threat and Confusing.
* Thus if we have basic understanding of C/C++ languages it is very easy to learn Java.

→ **Object Oriented** :-

Java Supports all important Concepts of OOPs, like:

→ Encapsulation
→ Inheritance
→ Polymorphism
→ Abstraction

* **Multithreaded** :—

Multithreading means Concurrent execution.

* In Simple terms it means that we can execute more than one part of the same program parallelly / simultaneously.

To understand this feature Consider the code given below:

```
Main()
{
clrscr();
factorial(5);
Prime(8);
evenodd(4);
}
.
.
.
```

* In the Previous Sample Code all 4 functions. clrscr(), factorial(), Prime() and evenodd() are independent of each other but still they will run Sequentially i.e, one after the other.

* This can be improved in Java by using multithreading feature So that all of these

functions can run together.

* **Benefits**: Reduced execution time, full utilization of CPU
* Some Practical examples where multi-threading is used are:

We can open multiple tabs in the same browser window.

When we use a media player to listen to a song, then there are multiple activities which take place parallely like:

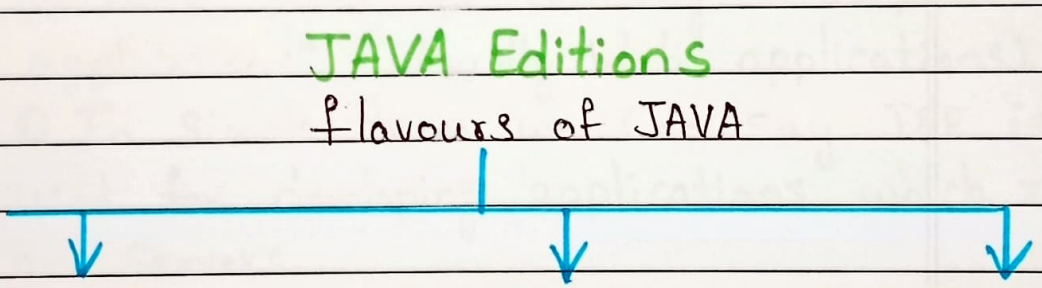
- Moving of a slider,
- elapsed time being shown,
- Volume adjustment,
- ability to add or remove songs from the playlist,
- Playing of the song etc...

History of JAVA

- Developed By: James Gosling
- Original Company Name: Sun microsystems
- Current Company: Oracle Crop
- Original Name: Oak but due to Copyright issues, changed to JAVA.
- First release: 23rd January 1996
- First version: JDK (Java development kit) 1.0
- Latest Version: JDK (17.0) released on 14th Sep 2021.

Editions / Flavours of Java

- When Java was originally released in 1996, it was just Java and no such thing as "editions" was there.
- But as use of Java increased, then in 1999, SUN Categorized it into 3 editions Called J2SE, J2EE and J2ME
- later on in 2006 they changed the naming and Called them as JSE, JEE & JME.
- These editions were named based on the kind of application which can be developed by learning that edition.



J2SE (Now known JSE)	J2EE (Now known JEE)	J2ME (Now JME)
Java Standard Edition	Java Enterprise Edition.	Java Micro Edition.
• Basics of Java • Language of Fundamental • Desktop Application.	• Web/Enterprise Application	• Applications run on mobile Phone, Pagers etc...

JSE (Java Standard Edition)

- This is the most basic version of Java

and it Provides us Core Concepts of Java language like the datatypes, operators, arrays, methods, OOP, GUI (Graphical User Interface) etc.

- Since it teaches us Core Concepts of Java that is why many people call it CORE JAVA although SUN never gave this name.
- Used for developing desktop applications like Calculators, media player, IDE etc.
- The **Java EE** which stands for (Java Enterprise Edition) is built on top of the Java SE platform and is a Collection of libraries used for building "enterprise applications" (Usually Web applications).
- In Simple term we Can say JEE is used for developing applications which run on Servers.
- Some popular applications developed using JEE are amazon.in, alibaba.com, irctc.co.in, ideacellular.com, airtel.in etc.

JME (Java Micro Edition)

- Java ME is the slimmer version of Java targeted towards Small devices such as Mobile phones.
- Generally People tend to think of the Micro edition as the mobile edition, while in reality, the micro edition is used not

just for mobile phones, but for all kind of devices, Such as television Sets, printers, Smart Cards and more.

- But as Smartphone technology arrived the use of JME has reduced as Android has Superseded it.

JDK (v/s) JRE (v/s) JVM

- Understanding difference between JDK, JRE and JVM is very very important in Java for interviews.
- These terms stand for :—

* JDK : JAVA Development Kit
* JRE : Java Runtime Environment
* JVM : Java Virtual Machine.

What is JVM?

* JVM is an abstract Machine that Can execute Precompiled Java programs.
* In Simple terms it is the code execution Component of JAVA.
* It is designed for each platform (OS+CPU) Supported by java and this means that Every platform will have a different Version of JVM.

QUIZ

Why JVM is Called a Virtual Machine?

JVM is Called Virtual machine because it is a Software layer but it behaves as if

it is a Complete machine (platform).

* That is all the tasks which are done by a machine while running a program in other languages like C, are actually done by JVM in Java.

For Example :—

Starting the Execution By Calling main(), Allocating memory for the Program, cleaning up memory cleanup etc...

What JVM Contains?

JVM Contains following important Components

→ Interpreter

→ Garbage Collector.

QUIZ

Are java Compiler and interpreter Same?

No, Not at all

* The java Compiler Converts Source Code to bytecode and is not a part of JVM, rather it Comes with JDK.
* the interpreter lives inside the JVM and Converts bytecode to Machine Understandable from.

What is JRE?

* JRE is an acronym for Java Runtime Environment.
* It Contains a JVM along with java classes/packages and set of runtime

libraries.

* So the JVM, while running a Java Program uses the classes and other libraries Supplied by JRE.
* If we do not want to write a java Program, and we just want to run it then we only need a JRE.

What is JDK?

JDK stands for Java Development kit and is a bundle of Software that we can Use to develop Java based applications.

It includes the JRE, Set of library class, Java Compiler, jar and additional tools needed while developing a Java application.

QUIZ

Q) Can I Compile a java application if I have a JRE?

- Yes
- No

Correct answer: No

JRE Can only be used to run a Java application. It doesn't Contain the javac tool which is used for Compilation.

Q) Which Component is Used to Compile, debug and execute java program?

a) JVM b) JDK c) JIT D) JRE

Correct Answer: B

Q) Which Component is Used to Convert bytecode to machine specific Code?
a) JVM b) JDK c) JIT d) JRE
Correct answer : A

Q) Which Component is used to provide a Platform to run a java program?
a) JVM b) JDK c) JIT d) JRE
Correct answer : D

Downloading JDK

1. Go to https://www.oracle.com/java/technologies/downloads/
2. Click on the tab of your OS and click on x-64 installer
3. You will be asked to login. So login to your Oracle account

Installing JDK

1. The download will begin
2. When the download Completes, you will get an exe file Called jdk-17.0.02_Windows-x64_bin.exe
3. Now, right click on this exe file and "Run as Administrator".
4. An installer window will pop up.
5. If everything goes perfectly you will receive a message Saying that installation is successful.

Verifying Installation

- Go to the Specified path, where you install your Java.
- By default Java gets installed in c:\Program files folder.
- There you will see a Java folder.
- Inside the Java folder you will see jdk-17.0.2 folder.
- Go to Start type Cmd and open it by clicking on it and type the following Command highlighted in red: (End files).

LTS And STS

* The last 4 years have Seen Some rapid changes in the way new Versions of the Java Development Kit is deployed and maintained.
* Traditionally, new java Versions were always released in a 2 to 4 year life cycle.
* Every 2 to 4 years, a new JDK would be released, Containing Some new features.

JDK release Timeline

JDK6 : launched in Dec, 2006
JDK7 : launched in July, 2011
JDK8 : launched in Mar, 2014
JDK9 : launched in Sep, 2017

* But from java 10, which was released in march 2018, Oracle changed the new release policy and decided to launch a new Version of Java every 6 months.
* Now, it became very difficult for Software Developers to keep their applications or Several hundred (thousand?) Servers up to date with the newest Java release.
* that is Why, the Concept of an LTS Was established.
* The next LTS Version is Java 11, which was released in 2018 and will Continue to receive updates until 2026, with a strong Possibility of date extension.
* The next 4 releases of Java, after Javava 11, which are Java 12, Java 13, Java 14 and Java 15 are all STS (Short Term Support)
* JDK 17, which has released in Sep 2021, is the latest LTS release and will be Supported as long as Sep 2029 with a Strong possibility of date extension.

Developing JAVA Programs

Java program Development

Using Notepad	Using an IDE like Netbeans/Eclipse/Intellij IDEA etc.
*Typing the code in notepad and running it through Command Prompt.	*This approach should be used after we have understood the basic working of java
*This approach is good for beginners for learning each step thoroughly.	*Not recommended for beginners because an IDE hides all the basic steps which are very important to understand.
*It is very helpful for better understand the steps clearly.	*It is also helpful and we have understood the basic working of Java.

Developing Java programs Using Notepad

Developing and running a java program requires three main steps:

1. Writing the Source Code
2. Compiling the Code.
3. Executing the Code.

Writing the Source Code

→ Select notepad.exe from the list shown in pop up menu of run Command.

→ Right click and choose "run as administrator" option.

→ Now type the Code given in next slide.

```
Class Test
{
  Public static void main(String [ ]args)
  {
    System.out.Println ("Hello User");
  }
}
```

Understanding The program

First of all we must remember that java is a highly Case Sensitive language.

* It means that we have to be very Careful about uppercase and lowescase letters while typing the Code.

For Example,

in previous Code three letters are Compulsorily in uppercase and they are

"T" of Test

"S" of string and

"S" of System.

This is because in java class names begin with upper Case.

The first Statement of Our Code is:

Class Test

Since java is an object oriented language and it strictly Supports Encapsulation So every java program must always Contain atleast one class and whatever we write must appear within the opening and closing braces of the class.

The Second statement is:

Public static Void main (string [] args)

In java also (like c/c++) the entry point of execution of our program is the method main() which is called by the JVM.

The words shown in blue are keywords and each has a different meaning and Purpose for the method main().

Why main() is public ?

* "Public" is an access modifier and in object Oriented Programming any method or Variable which is declared Public Can be accessible from outside of the class.
* Since main() method is Public in Java so, JVM Can easily access and execute it.

Why main() is static ?

→ Every class Can be have two kinds of methods nonstatic and static

→ A method which is nonstatic Can only be Called Using object of that class, while

a static method can be called without any object, simply using class name.
→ When the JVM makes a call to the main() method there is no object existing for that, therefore it has to have static method to allow invocation from outside the class.

Why Main() has return type Void?

* The keyword void is called return type which indicates that no value will be returned by the method to it's caller.
* Since main() method in Java is not supposed to return any value to the JVM its made void which simply means main() is not returning anything.

Can we change/remove the keyword used with main()?

→ No, Not at all.
→ This is because main() is called by JVM and to allow JVM to successfully call main() these keywords are important.
→ If we forget to write these keywords then although the code will compile but fail to run.
→ All we can do is change the order of public and static but we can't drop them.

What is string [ ] args?

* String is predefined class in Java
* So the statement string [ ] args is declaring args to be an array of string.
* It is called Command line argument and we will discuss it later.
* For now, just remember that the Statement string [ ] args. has to be present with main() otherwise Code will not run.

Can we change/drop string [ ] args?

- No, just like Keywords used with main() are Compulsory, Similarly string [ ] args is also Compulsory.
- All we can do is change the name from args to Something else.
- Also we can interchange the array name and [ ]
- For example: String args[ ], string [ ]args, String[ ]str, String str[ ] all are Valid.

Understanding System.out.Println()

- Now let's understand Code in the body of the main() method, which will print a message on the Console.

Syntax:

```
System.out.println("message");
```

- System is a predefined class
- Out is an object refrence (not object).
- Println() is a method.

- Together all three are used for displaying text on Console.
- We will discuss this part in detail Once we have Covered basics of JAVA

QUIZ

Q) Does every method has to be public, static and void?

- Yes
- No

No, it is not a Compulsion. This is only with the main() method that we have to make it public, static and void. All other methods have declarations are decided by the programmer.

Q) will a java program Compile without main()?

- Yes
- No.

Yes, Because main() is not needed for Compilation. But is used for execution of the Code. So we Can Compile a Java Program without main(), but we Cannot run it.

Saving the Source Code

→ Once we have written the Code, the next step is to Save and Compile it.

→ We Can Save Our Code in two locations:
Within "bin Subdirectory of jdk

OR

At any location in Our machine

→ (This requires setting "PATH" Variable also).

→ We will start with first approach and then migrate to Second approach while learning about packages.

→ To Save the Code in "bin", just choose jdk's bin as the "location to Save" in notepad.

→ In the "file name" option provide any name you like but with .Java extension.

→ Generally we Prefer giving the Same name to Our Code Source Code as the name of Our Class. (Remember it is a general choice not a rule!)

→ Also remember to give the filename in "double quotes" as otherwise notepad might add the extension.

→ Now Since our class name is Test, So we would Save Our file by the name. "Test.java"

Compiling the Source Code

* To Compile Our Code we have to do the following:
* Open the Command prompt by right clicking and selecting "run as administrator" option.
* Migrate to the jdk's bin folder.
* Type the Command to Compile the Code.
* The general Syntax of Compilation is:

<javac> <full name of .java file>

javac is the name of java's Compiler which takes the name of Our Source Code as argument and generates the bytecode.

For example:

```
javac Test.java
```

* Remember this Command has to be given from jdk's "bin" folder as we have saved the file there only!

What happens When we Compile Our Code?

* Whenever we Compile Our java Code, the Compiler does the following:
* It checks for Syntax error (like missing Semicolons, Wrong class or method names etc.
* If any Syntax error is found the Compilation Stops.

* Otherwise if no syntax errors are there the Compiler generates the "bytecode" of our "Source Code".

Points To Remember about "bytecodes"

* Bytecodes are generated as Seperate files.
* These files have the extension .class and their name is Same as the name of the class defined by the programmer.
* For example if class name is Text the bytecode name will also be "Test.class"
* Number of bytecode files generated is Same as number of programmer defined classes. So if our program Contains three class Called "College", "Faculty" and "Student", then three bytecode files would be generated Called:
  * College.class
  * faculty.class
  * Student.class

Executing the Code

* The general Syntax to run our Code is:

  Java < Name of the class Containing main method >

* java : is the java interpreter which takes .class file as argument (note: do not write the extension .class).
* This class file should Contain main() Method that is executed by the Java

Interpreter.

* For example, if class "Test has the main() method the our Command would be:

QUIZ

Q) How many class files would be generated for the following Code:

```
class A
{
 -----
}
class B
{
 -----
}
class C
{
 -----
}
```

Answer :- 3

Q) What should be the name of the Program:

```
class A
{
 ---
}
class B
{
 -----
}
```

```
class C
{
....
}
```

Answer:- Although we can give any name but it is preferred to give the same name as the class which contains main() method.

Q) What should be the name of the program:

```
class Indore
{
public static void main(String [Jargs)
{
System.out.Println("In Indore");
}
}
class Bhopal
{
Public static void main(string [Jargs)
{
Syetem.out.Print In("In Bhopal");
}
}
```

Answer:-

Can be either "Bhopal.java" or "Indore.java"

Q) In the previous Code Which main() method will be called by JVM if we run our Code?

Answer: It depends on how we run the code!
if we run it as:

java Bhopal

Then output would be

In Bhopal

And, if we run it as:

java Indore

Then output would be:

In Indore

Q) Suppose we write the following code:

```
Class Test
{
  Public Static Void main (String [] args)
  {
  System.Out.Println("Hello User");
  System.Out.Println("Welcome To Java");
  }
}
```

* Now when we will run it:

Java Test

* The output will be:

Hello User

Welcome to Java

* Did you notice Something?
* The line "Welcome To Java" automatically got displayed on second line. Why?
* Because the method println() implicity adds a new line at the end after displaying the Message.

*In Case we do not want the newline effect then we Can Use another method Called.

*So if we write:

```
System.out.print("Hello User");
System.out.print("Welcome To Java");
```

Then the Output would be:

Hello User Welcome To Java

Q) Would be the output of the following Code:

```
System.out.print("Hello User);
System.out.println("Welcome To Java");
```

Output :-

Hello User Welcome To Java

This is because the method println() method puts a newline after the message not before it.

Q) What would be the output of following code:

```
System.out.println("Hello User");
System.out.print("Welcome To Java");
```

Output :-

Hello User
Welcome To Java

Q) What would be the output of the code:

```
System.out.print("Hello User");
System.out.println();
System.out.print("Welcome To Java");
```

output :- Hello User
Welcome To Java

This is because the method print() always requires argument so. we Cannot Call it without arguments.

Some Common Errors!

*There are Some Very Common Mistakes which a Programmer might make in his Code due to which errors arise.

These are:

Forgetting to match number of opening and closing braces.

SOME More Concepts

*A. very Common doubt which might arise in our mind while learning java is that from where we are getting access of "String" and "System" Classes.

*We know that they are predefined classes but We haven't included any predefined file in our Code (like header files) but still we are able to use "String" and "System" class.

* In Java we don't have header files, rather we have packages.

*A package is just a folder which Containing java classes and as of java 14 there are 224 packages Containing 4569 classes.

* Amongest these packages there is a package Called java.lang which provides classes that are fundamental to the design of the Java programming language.

*Since these classes are so essential, the

Package java.lang is implicity added to our program by the java Compiler itself.
* The classes String, System, Math and many more Come from this package only.
*But if we want to add this package Ourself then we Can do so by writing "import" Keyword.
* In Java to add the Support of a package/class in our Code we use the Keyword import whose general syntax is:

import <package-name>.<class-name>;

OR

import <package-name>.*;

for example:

import java.lang.string;
import java.lang.System;

OR

import java.lang.*;

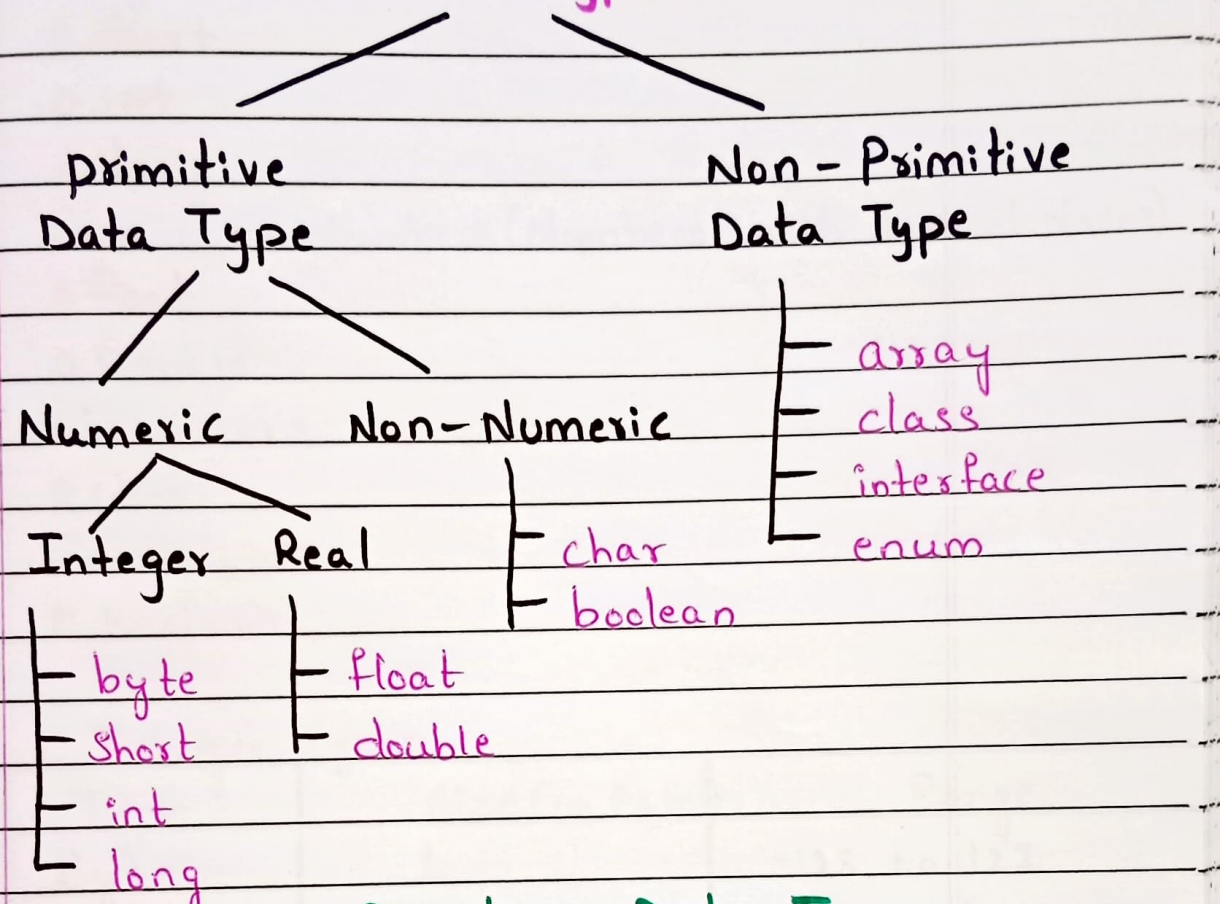
Data Types

Generally, data types are used to Create Variables where variables will hold values, as defined in the range of values of a data type.

Java Supports two Categories of data types.

- Primitive data types and
- Non primitive Data Types.

Data type Categories



Non Primitive Data Types

* Non primitive data types are also called as Reference data types.
* A Non primitive Data Type is used to refer to an Object.
* In java, Variables of type class, arrays, enums and interface are represented as objects.
* We will discuss this in later chapters like Arrays and classes and objects.

Non Primitive Data Types

* There are totally eight primitive data types in Java. They can be categorized as given below:

Integer types (Does not allow decimal places)

- Byte
- Short
- int
- long

Rational Numbers (Numbers with decimal places)

- float
- Double

Characters

- char

Conditional

- boolean

- Integer Types

Type	Size (in Bytes)	Range
Byte	1	-128 to 127
Short	2	-32768 to 32767
int	4	-2147483648 to 2147483648
long	8	-9223,372,036,854,775,808 to 9223,372,036,854,775,807

- Rational Numbers

Type	Size	Range
float	4	$-3.4 * 10^{38}$ to $3.4 * 10^{38}$
double	8	$-1.7 * 10^{308}$ to $1.7 * 10^{308}$

- Characters

Type	Size	Range
char	2	0 to 65535

Why Java Uses 2 bytes for characters?

* In Java almost 61 international languages are Supported.
* Now, the characters and Symbols of these languages Cannot be accommodated in 1 byte space in memory, So java takes 2 byts for characters.
* java Supports UNICODE but c language supports ASCII Code. In ASCII Code we Can represent characters of English language ; for storing all English latter and symbols 1 byte is Sufficient.

But UNICODE character set is superset of ASCII in which all the characters which are available in 61 international languages are Supported and it Contains 65536 characters ranging from 0 to 65535

To assign UNICODE values we have 2 options:

- Use the numeric value (or)
- use the format '\uxxxx' where xxxx is heradecimal from of the value.

For example :

- char ch=65;

(or) - char ch='\u0041';

Conditional

Type	Size	Range
Boolean	JVM dependent	true or false.

Type Conversion

What is type Conversion?

Whenever the Compiler encounters a statement where the value on right side of assignment is different than the variable on Left, then the Compiler tries to convert R.H.S to L.H.S and this automatic Conversion done by Compiler is Called as Type Conversion.

For example:

Consider the following statement:

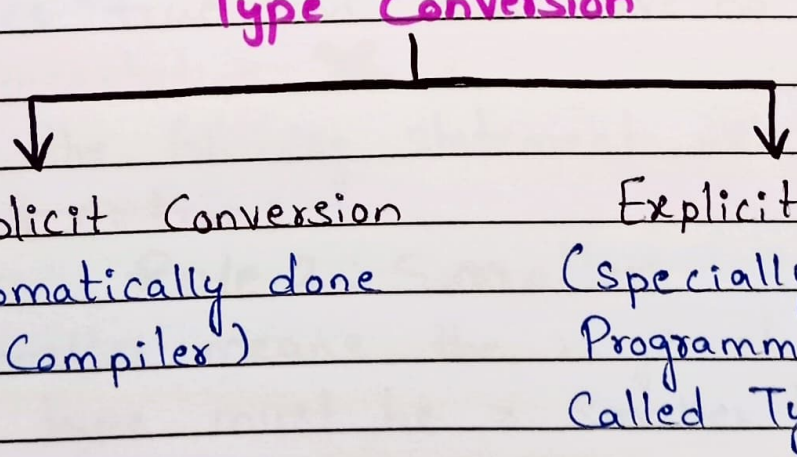
```
x = y;
```

In this Case two things Might happen:-

* If datatype of both variables are Same then Value of y will be assigned to the variable x.
* But if datatype of both Variables are different then the Value of Y needs to be Converted as per the datatypes of Variable X and this is Called "Type Conversion".

Forms of Type Conversion

In Java, type Conversion is of two types



Rules For Implicit ConVersion

* For implicit Conversion there are 2 Conditions which must be true:

- The values must be Compatible/Convertible.

AND

- The value on RHS of assignment must be Smaller than Varible on LHS.

* If both these rules are followed then Java will implicitly Convert the value otherwise Conversion has to do done by the Programmer.

Rule 1 : Convertible

→ Convertible means it must be possible for java to Convert a Value from one form to another

For example, It is possible for java to Convert a character to an integer Using it's UNICODE/ASCII Value.

So the following will Compile:

```
int x = 'A';
```

✔

→ But it is not possible for java to Convert the boolean value "true" to integer as the values "true" and "false" have no other Representation. ✘

So the following statement will not Compile:

```
int x = true;
```

Rule 2 : Smaller

* Smaller means the range of a Variable's data type must be a Smaller than other Variable's range. NOT THE SIZE.

* for example, a short data type Variable has a range of -32768 to 32767 Which is Smaller (proper Subset) of the range of an int Variable Whose range is -2147483648 to 2147483647, So "Short" is Considered to be Smaller than an "int".

* Another example, an int of 4 bytes and has a range of -2147483648 to 2147483647 While a float is also of 4 bytes but has a range of $-3.4 * 10^{38}$ to $3.4 * 10^{38}$ which is greater than the range of int, So int is Smaller than float.

Type Conversion In Expression

```
byte a=10, b=20;
byte c
c=a+b;
```

✗

Possible loss of Precision.
because both a and b are bytes java Converts them to int

Sol:- 1

```
c=(byte)(a+b);
```

✓

not reliable as byte is < int so there maybe rotation of value.

Sol:- 2

```
byte a=10;b=20;
int c;
c=a+b;
```

✓ More reliable

QUIZ

Which of these is necessary Condition for automatic type Conversion in Java?

a) The destination type is smaller than Source type.
b) The destination type is larger than Source type.
c) The destination type can be larger or Smaller than Source type.
d) None of the mentioned.
Answer :- B

Q) What is the error in this code?

```
byte b = 50;
b = b* 50;
```

a) b Can not Contain value 2500, limited by its range.
b) * operator has Converted b* 50 into int, which Can not be Converted to byte without Casting.
c) b Can not Contain value 50.
d) No error in this Code.
Answer :- B

Q) If an expression Contains double, int, float, long, then whole expression will promoted into which of these data types?
A) long
b) int
c) double
d) float
Answer :- C

Accepting Input

In Java there are 3 ways to accept input from User :-

→ Through Command line arguments
→ Using Scanner classes
→ Using GUI Components

This lecture we Cover Command line arguments and the other two methods will be Covered in future lectures.

Using Command line Arguments

Using Command line Arguments we can accept input when we are about to execute the program.

This is where the arguments of method main() Come in action.

Let us take an example to Understand this...

Case 1 :-

```
class Test
{
Public static void main( String [ ]args)
{
System.out.Println(" Hello " + args[0]);
}
}
```

```
... bin > Java Test Sachin
```

Case II :-

```
class Test
{
Public static void main(String [ ]args)
{
```

```
System.out.println("Hello" + args[0]);
System.out.println("Hello" + args[1]);
}
}
```

```
... bin > Java Test Sachin Amit
```

CASE - III :-

```
Class Test
{
Public static Void main(String [Jargs)
{
System.out.println("Hello" + args[0]);
System.out.println("Hello" + args[1]);
}
}
```

```
... bin > Java Test Sachin
```

CASE - IV

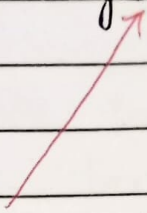
```
class Test
{
Public static Void main(String [Jargs)
{
System.out.println("Hello" + args[0]);
System.out.println("Hello" + args[1]);
System.out.println("Bye!");
}
}
```

```
... bin > Java Test Sachin Amit Sumit.
```

CASE - V

```
class Test
{
Public static Void main(string [Jargs)
{
```

```
System.out.println ("Hello" + args[0]);
System.out.Println ("Bye!");
}
}
```



```
... bin > Java Test "Sachin Amit"
```

CASE -VI (passing Integers)

```
Class AddNos
{
Public static void main(String [] args)
{
System.out.Println ("First number is "+args[0]);
System.out.Println ("Second number is "+args[1]);
System.out.Println ("Their sum is "+args[0]+args[1]);
}
}
```

```
... bin > Java AddNos 10 20
```

Why was the output 1020?

* Because anything which we pass from "Command prompt" is by default treated as a String by Java.
* Now since Java is Considering the values 10 and 20 as "10" and "20", so the operator + Concatenated them instead of adding them mathematically.
* To Solve this problem we have to Convert the values "10" and "20" from string to int and this is done Using Special classes in Java Called "Wrapper classes".

Wrapper classes

* In java, Corresponding to 8 primitive data

types we have 8 predefined classes also Called "Wrapper classes".

* These classes are available in the package java.lang and their names similar to the name of data type.

for ex :- Integer, character, float, Boolean etc.

* Notice that the first letter in Wrapper class name is in uppercase while in case of datatype name it is in lowercase.

for ex :- byte and byte, long and long and so on.

Uses of Wrapper classes

* Wrapper classes are mainly used for two purposes:

- To represent Primitive data types as objects.
- To Convert string from of a primitive Value to it's original from, for ex: "10" to 10

Representing primitives as objects

* Consider the following statement:

```
int a = 10; // variable a
```

* Here "a" is a Variable initialized to 10
* But if we want we Can Convert it into an object by using the Wrapper class Integer, as shown below:

```
Integer obj = a; // variable a Converted to object
```

Converting string To primitive

* Another importance of wrapper classes is that they Contain Special methods which perform

Conversion from string to primitive datatype.

* These methods have their name as parseXXX where XXX is the name of primitive type.
* Also they are static in nature, so they can be directly called by their class name.

for example :- Integer.parseInt("....");

List of Wrapper classes

Data type	Wrapper class	Method.
int	Integer	parseInt()
Short	Short	Parseshort()
byte	Byte	ParseByte()
long	Long	ParseLong()
float	Float	Parse Float()
double	Double	Parse Double()
Char	Character	No available
boolean	Boolean	Parse Boolean()

Addition of Number Using Wrapper Classes:

```
Class AddNos
{
Public static Void main(string args[])
{
int a,b,c;
a = Integer. ParseInt(args [0]);
b = Integer. ParseInt(args [1]);
c = a+b;
Syetem.out.println ("First number is" +a);
Syetem.out.println (" Second number is" +b);
Syetem.out. PrintIn(" Their Sum is" +c);
}
}
```

Guess the Output?

```
Class AddNos
{
Public static void main(String args[])
{
int a,b,c;
a=Integer.ParseInt(args[0]);
b=Integer.ParseInt(args[1]);
c=a+b;
Syetem.out.println("First number is "+a);
Syetem.out.println("Second Number is "+b);
Syetem.out.Println("Their sum is "+c);
}
}
```

Running:

> bin> java AddNos 10 Bhopal.

Why did Exception occur?

* Because the method parseInt() Can only Work with strings Containing digits.
* If any string Contains non integer values then the method parseInt() will throw Exception.
* Even if we pass 20.5, then also it will throw.

"NumberFormatException".

How do accept decimal Values?

* So if we want to accept decimal values, then we must use the method parsefloat() or ParseDouble().
* They accept decimal/integer both kinds of values and throw Exception only if the given values is Non-numeric like "Bhopal", "10a" etc.

@ Python_World-In

Decision Control Statement

* Decision Making is the Most Crucial part of any program.

for Example :- Deciding whether a given number is Even or odd.

* In such Case Java Supports Various Decision Control Statements like other Programming languages they are :-

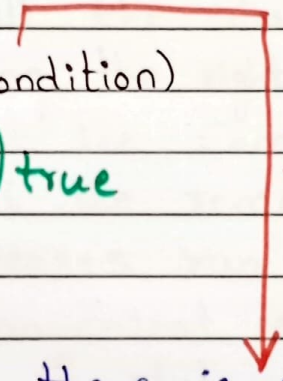
→ if, if else, nested if

→ Switch

→ Ternary operator

Syntax :-

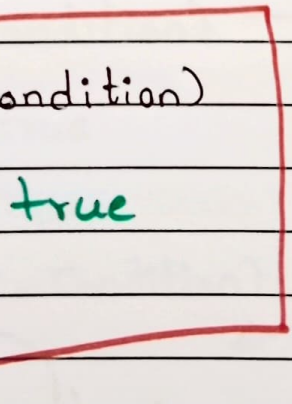
```
if(test_condition)
{
  ....
  ....
}
```



← if statement

* In Case there is only a single statement in the body of if-statement then curly braces Can be dropped.

```
if(test_condition)
{
  ....
  ....
}
else
{
  ---
  ---
}
```



← if else

- Every else statement should have one if statement.

if else if

*In Case of Checking multiple Conditions there are two options,

1. Use only if statement to check every Condition.
2. Use else if statement after the first if statement to check all the other remaining Conditions.

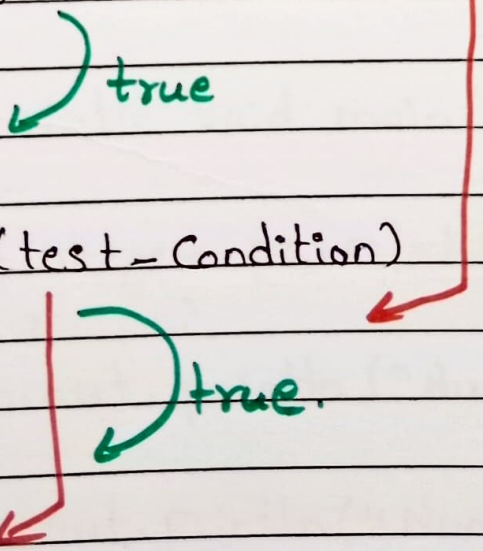
*The first method holds a drawback. Can you tell what???

➤ the drawback in using only if statement to check all the Conditions if, that even after getting the right statement and executing it the Compiler still Continues checking all the remaining statements, which increases run time of the program.

*So it Convenient and suggested to Use if else if statement to check multiple Conditions.

if else if

```
if(test_Condition)
{
----
----
}
else if(test_Condition)
{
---
---
}
else
{---
}
```



Nested if

Any Conditional statement (whin) with in the other Conditional Statement makes it nested in nature.

```
if(test Condition)
{
if(test Condition)
{
  ----
  ----
}
else
{
  ----
  ----
 }
}
```

Try this...

Accept an integer from user via Command line argument and check whether it is odd or even in nature.

Solution

```
class Even odd
{
Public static void main(String [] args)
{
int a=integer.parseInt(args[0]);
if(a%2==0)
System.out.println("Number is even");
else
System.out.println("Number is odd");
}
}
```

Try this...

* A Company provides insurance to it's employees according to the following Criteria:

- If the employee is married.
- If the employee is unmarried, Male and above 35 years of age.
- If the employee is unmarried, Male and above 30 years of age.

In all other cases insurance is not given. WAP to accept age, gender and marital status from the User Using Command line arguments and check whether the User is eligible for insurance or not.

The switch Statement

The switch statement is similar to if statement, as it is also a decision control statement.

It allows a Variable to be tested aganist a list of values where each Value is Called a Case.

Syntax :-

```
Switch(Variable_name or expression)
{ Case value : // statements
                break;
  Case value : // statements
                break;
    :
  default : // statements
}
```

- The switch statement can use different variables to check the conditions, which are byte, short, char, int.
- The use of strings and enumerated types are also supported.

Example :-

```
int month = 8;
Switch (month)
{ Case 1: System.out.Println ("January");
        break;
  Case 2: System.out.Println ("February");
        break;
  // and so on...
  default: System.out.Println("Invalid month");
}
```

- Case II - clubbing Cases :-

```
Switch (Variable name)
{
Case Value 1: Case Value 2: Case Value 3:
                ------
                break;
Case value 4: Case value 5: case value 6:
                ----
                break;
default:
}
```

* Any number of cases can be clubbed together as per condition.

— Exercise —

→WAP to accept a month number from the User via Command line argument and display the name of the season in which the month falls according to the table given below.

Month Numer	—	Season Name
11, 12, 1, 2	—	Winter
3, 4, 5, 6	—	Summer
7, 8, 9, 10	—	Rainy
Any other value	—	wrong input

→WAP which should accept 3 arguments via Command line of type operand, operator and operand and should display the result by performing appropriate Calculation.

Solution

```
java Calculator 10+4
  Sum is 14
java Calculator 3-8
  Difference is -5
```

— Ternary operator —

→The ternary operator Can be used as an alternative to the java's if-else and Switch statements.

But it goes beyond that, and Can even be used on the right hand side of java statements.

Syntax :-

```
<variable> =(test Condition)?<true Case>:
<fasle Case>;
```

Example :-

```
int a = 4;
String str;
str = (a % 2 == 0) ? "Even" : "Odd";
System.out.println(str);
```

Try this

→WAP to accept the integer via Command line argument and print its absolute value. (If user enters -1 then result should be 1).

Solution :-

```
class PrintAbsolute
{
Public static void main( string args[])
{
int a,b;
a = integer.parseInt(args[0]);
b = (a>=0?) ? a : -a;
System.out.println("Absolute value is "+b);
}
}
```

→WAP to accept an integer via Command line argument and check whether it is a leap year or not.

Note : Not every year divisible by 4 is a leap year. For example 1700 was not year is leap. But 1600 was a leap year. Similarly year 2000 is a leap year but 2100 will not be a leap year.

So the Condition for leap year is that:

1. Year must be divisible by 4 and not divisible by 100 (OR)
2. Year must be divisible by 400.

Loop Structure Types :-

* Loops in java also are control statements used for repeating a set of statements multiple times.

They are broadly categorized to be of "2" types.

➢ Entry Controlled - Condition is checked when the Control enters loop body. Example, while and for loop.

➢ Exit Controlled - Condition is checked after the flow enters loop body. Example, do-while loop.

While loop

Syntax :—

false

```
While (test Condition)
{
  ----
}
```

* The loop Continues will until the Condition is true, as Soon as the Condition goes flase flow exits the loop body.

Exercise-1

WAP to accept an integer from the User and print its factorial. Make Sure that your Program should print 1 if 0 is entered?

```
import java.util.*;
class Factorial
{
public static void main(String [] args)
```

```
{
Scanner kb=new
Scanner(System.in);
int n,f=1;
System.out.Println("Enter a no");
n=kb.nextInt();
while(n>=1)
{
f=f*n;
n--;
}
System.out.println("factorial is "+f);
}
}
```

Output:- Enter a no 5
Factorial is 120

Do-While loop

Syntax :-

```
do
{
----
----
}while(test Condition);
```

* Since, the Condition is tested at exit, so the loop body will execute at least once irrespective of the Condition.

Exercise - 2

- WAP to accept two integer from the User and display their Sum. Now ask the User whether he/she wants to Continue or not. If the answer is Yes then again repeat the process otherwise terminate the program.

displaying the message "Thank you".

Solution Ex 2

```
Class AddNos
{
Public static void main(String[] args)
{
java.util.Scanner kb = new
  java.util.Scanner(system.in);
int a,b;
String choice;
do
{
System.out.println("Enter two integers");
a = kb.nextInt();
b = kb.nextInt();
System.out.Println("Sum is" + (a+b));
System.out.Println("Try again? (Y/N)");
? choice = kb.next();
}while(choice.equalsIgnoreCase("Y"));
System.out.println("Thank you");
}
}
```

Can we replace next() with nextline()???

Buffer

* Buffer is a region of a physical memory storage used to temporarily store data while it is being moved from one place to another.
* So, in above case the keyboard's buffer is left with an "ENTER KEY" while we pressed after inputting the Second integer, which nextLine() accepts as an input.

* How Can we solve this???
* By Calling nextLine() before accepting actual input. Input since, this call would clean the buffer.

Solution Ex:2

```
Class AddNos
{
Public static void main(string [] args)
{
 java.util.Scanner kb = new
  java.util.Scanner(system.in);
int a,b;
String choice;
 do
{
System.out.println("Enter two integers");
a = kb.nextInt();
b = kb.nextInt();
System.out.println(" Sum is "+(a+b));
System.out.println("Try again? (Y/N)");
nextline();
choice = kb.nextline();
}while (choice.equals IgnoreCase ("y"));
System.out.println("Thank you");
}
}
```

For and labeled for loop

Syntax :-

```
for(initialization; test Condition; statement)
{
  ---
  ---
}
```

* The initialization and statement part Can be left blank.

break and Continue

* To terminate the loop and exit its body providing a Condition before it, in Such Cases we use the statement
* In situations where we want to skip further steps in a loop and move directly back to the test Condition, there we use the statement
* Let us Understand these through an Example
* WAP to accept an integer from User and check whether it is prime or not?

```
import java.util.*;
class CheckPrime
{
Public static Void main(string [Jargs)
{
Scanner Kb=new Scanner (System.in);
System.out.println("Enter a no");
int a=kb.nextInt();
int i;
for(i=2;i<=a-1;i++)
{
if(a%i==0)
break;
}
if(a==i)
System.out.Println("No. is prime");
else
System.out.Println("No. is not a prime");
}
}
```

Exercise

* Write a program to accept an integer from the User Calculate and print the Sum of it's digits. For example if input is 75 then Output Should be 12.
* Write a program to accept an integer from the User Calculate and print the Sum of it's first and last digit only. For example if input is 21 75 then output Should be 7.
* Write a program to accept an integer from the User and print it's reverse. For example if input is 451 the output should be 154.
* Write a program to accept an integer from the User and check whether it is prime or not.
* Write a program to accept an integers from the User and print it's table up to 10 terms.

Nested Loop

- Syntax :—

```
for (init; Condition ; Stmt)
{
for (init; Condition ; Stmt)
{
  ---
  ---
}
}
```

* After Completing the inner loop, the Control moves back to the statement part of Outer loop.

Labeled break and Continue

Since, the break Condition brings us out of the loop body but in case of nested loop if we want to Completely Come out of the loop, then we will use labeled break Statement.

Syntax :-

```
<label name> ;
for (init; Condition; stmt)
{
for (init; Condition; stmt)
 {
  ----
break <label name> ;
 }
}
```

Exercise

→ WAP to accept an int from user and print Sum of its digits?

→ WAP to accept an int from user and check whether it is armstrong or not?

→ WAP to accept an int from User and print its reverse?

→ WAP to accept an integer from User and check whether it is equal to its reverse or not?

→ Write a meno driven program in which you will provide 4 choices to the User :-
1. Factorial 2. Prime 3. Even/odd 4. Quit

Arrays

* An array is a Collection of data of Similar datatypes, be it Primitive or Non-Primitive. Example, an array of integers will will be consist a Collection of integer type data, an array of names will be a Collection of strings and so on.
* Array in Java is treated as an object. So, to Create an array we use the Keyword "new".

Syntax :— There are 2 steps involved in

1. Creating array reference —

< data type > [ ] < array reference name >; (or)

< data type > < array reference name > [ ];

Example - int [ ] arr;

Here 'arr' is a reference to an array of integers

Double tap to add title

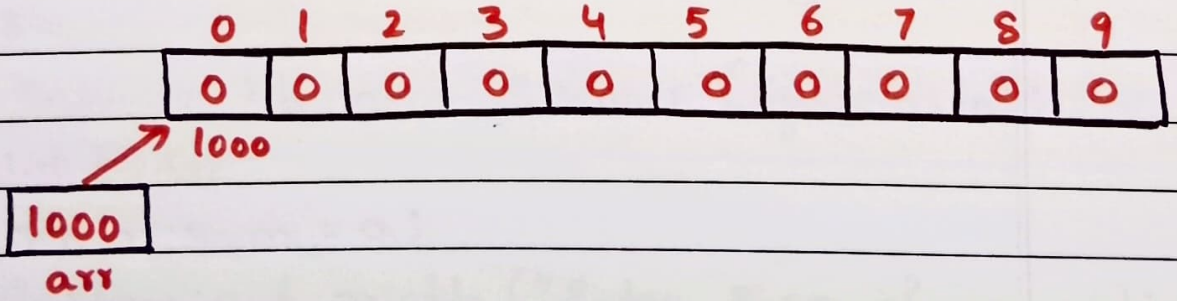
2. Creating actual array -

< array reference > = new < data type > [size];

Example :-

int [ ] arr;

arr = new int[10]; (or) int [ ] arr = new int [10];



* Whenever new is used in java, it gives default value 0 to the object formed.

→Intializing the array is same as that in C/C++ Example,

```
arr[0]=10;
arr[1]=20;
int[]arr={10,20,30,40,50}//can be initialized.
```

→Size of array can be set by variable which was not allowed either in C and C++ Example,

```
int n=10;
int[]arr=new int[n];
```

→If the size is given negative then Negative Array Size Exception occurs.

Exercise :- WAP to Create an array of 'n' integers where n is given by the user. Then ask the user to input values in that array and finally display the sum and average of all the programs of Numbers / Sum and average of all the numbers entered by the user.

Solution:-

```
import java.util.*;
class ArrayDemo
{
Public static void main(string args[])
{
Scanner kb=new Scanner(System.in);
int[]arr;
int n, Sum=0;
System.out.println("Enter size of array");
n=kb.nextInt();
arr=new int[n];
System.out.println("Enter numbers in array of size"+n);
for(int i=0;i<n;i++)
{
```

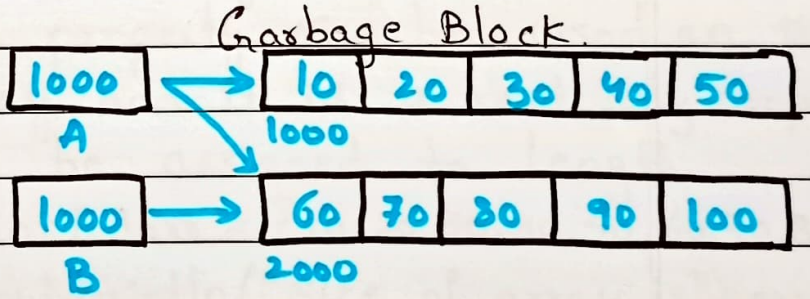
```
arr[i] = kb.nextInt();
Sum = Sum + arr[i];
}
System.out.println("Sum is" + Sum);
System.out.println("Average is" + (float)Sum/n);
}
}
```

De allocation of Dynamic Blocks :-

→ To Understand de allocation of dynamic blocks in java, we first have to understand the Concept of garbage blocks.

→ Garbage block in java are those dynamic blocks which are no more referred by any reference. For, example, let us Consider two arrays.

```
int[] A = {10, 20, 30, 40, 50};
int[] B = {60, 70, 80, 90, 100};
A = B;
```



→ Now, at this point the array with address 1000, is no more pointed by any reference. Hence, it is a garbage block.

Garbage Collector

* To Collect these garbage blocks from the main memory and deposit them back into free pool, Java internally uses a Software (build into JVM) named garbage blocks.

→ The garbage Collector periodically scans program's memory area identifies garbage blocks and Submits them back into free pool.

→ The programmer is Completely unware of this activation process of garbage Collector.

e So, in java dynamic blocks or objects have undetermined life time, as they are created on programmers request and de allocation is exclusively handled by JVM.

Using length property

→ Properties are Special methods which Can we Called without using parenthesis. Programmers Cannot develop properties in their class.

→ By Using this property we Can easily get the Size of an array.

→ The length property Can be used on any type of array and it is a read only property nothing Can be assigned to length.

→ Example :- int[] arr = {10, 20, 30, 40, 50, 60, 70, 80};
System.out.println("Size of array is" + arr.length);

— Exercise —

WAP which accepts Some integers Using Command line arguments and displays their Sum. In Case numbers passed are less then two, the program should display the message "please pass at least 2 numbers"

Solution :- Class LengthPropertyDemo.
{
Public static void main(String[] args)

```
{
int n, Sum =0;
n = args.length;
if (n<=1)
{
System.out.println("please enter atleast 2 numbers");
System.exit(0);
}
for(int i=0; i<n; i++)
Sum = Sum+integer.parseInt(args[i]);
System.out.println("Sum is" +Sum);
}
}
```

Enhanced for Loop

*Enhanced for Loop was introduced in Java in its version 5.0.

Syntax:-

```
for (<datatype><Variable name>: <array reference>)
 {
   ---
   ---
 }
```

*Variable's data type should be Same that of array.

*This loop is mainly used to perform read only operations. We Cannot change or manipulate array values using this loop.

Let's understand this through an example :-...

Example :-

```
Class EnhancedFor Demo
{
public static Void main(String[]args)
{
```

```
int[] arr = {10,20,30,40,50};
for(int x : arr)
{
 System.out.println(x);
}
}}
```

Drawbacks of Enhanced for loop

→ Array Cannot be (trans) traversed from the end, it will always start from the first element.

→ they array will be traversed Completely, it will not exit the loop at any intermediate point.

→ the variable has to be declared in the loop brace, it is a part of it's Syntax.

→ We can only perform read only operations i.e. traverse the array, no change or manipulations can be made in the array.

Two Dimensional Arrays

Rectangular 2D Array
Every row has same number of columns

Jagged 2D Array
Every row Can have different numbers of Column.

Rectangular 2D Array

- Creating array reference.

<datatype>[][] <array reference>; (OR)
<datatype> <array reference>[][];

Example - int[][] arr;

• Creating array object :-

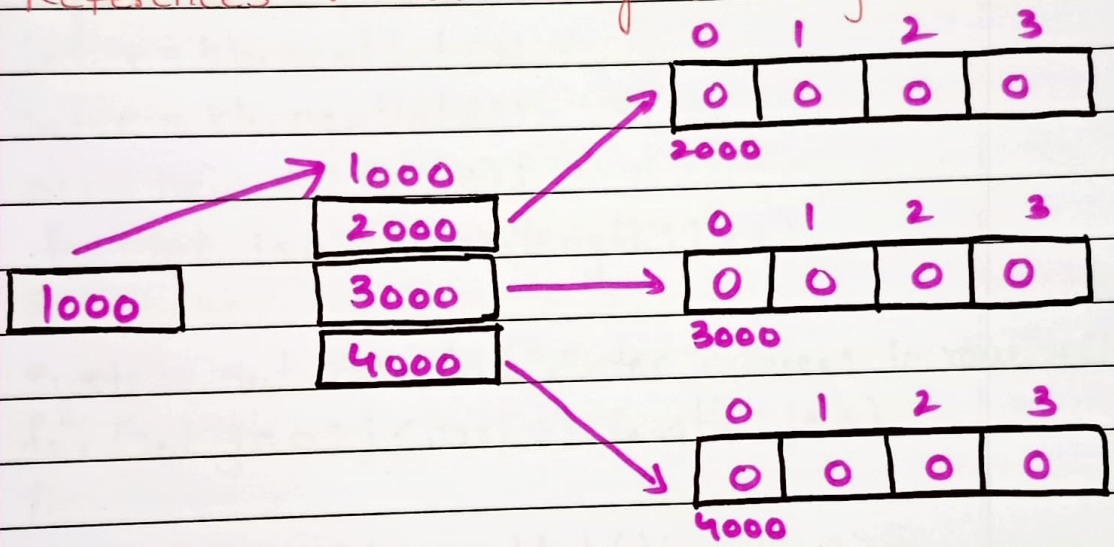
<array reference> = new <datatype>[size][size]

Example - arr = new int[3][4];

* "arr" is a reference to an array of References to an array of integer.

Let us understand this line through diagrammatic representation.

* "arr" is a reference to an array of References to an array of integer.



● What will be the output for this...

```
int [][] arr = new arr[3][4];
1.System.out.println(arr.length);          3
2.System.out.println(arr[0].length);       4
3.System.out.println(arr[0][0].length);    Error
```

— Exercise —

* WAP to Create a rectangular 2D array of the row and Column size mentioned by the User. Now, ask the user to input values in that array and finally display these values in matrix from as well as their Sum and average.

Solution:-

```
import java.util.Scanner;
class TwoDDemo
{
Public static void main(string[] args)
{
Scanner kb=new Scanner(System.in);
int [][] arr;
int Sum = 0;
System.out.println("Enter number of Rows & Columns");
int r = kb.nextInt();
int c = kb.nextInt();
arr = new int[r][c];
for(int i=0;i<arr.length;i++)
{
System.out.Println("Enter numbers in row"+(i+1));
for(int j=0;j<arr[0].length;j++)
{
arr[i][j] = kb.nextInt();
 Sum = Sum+arr[i][j];
 } }
for(int i=0;i<arr.length;i++)
{
 for(int j=0;j<arr[0].length;j++)
{
 System.out.Print(arr[i][j]+" ");
}
System.out.Println("\n");
}
System.out.Println("\n\n Sum of all numbers
is "+Sum+"\n Average is "+((float)sum/(r*c)));
}
}
```

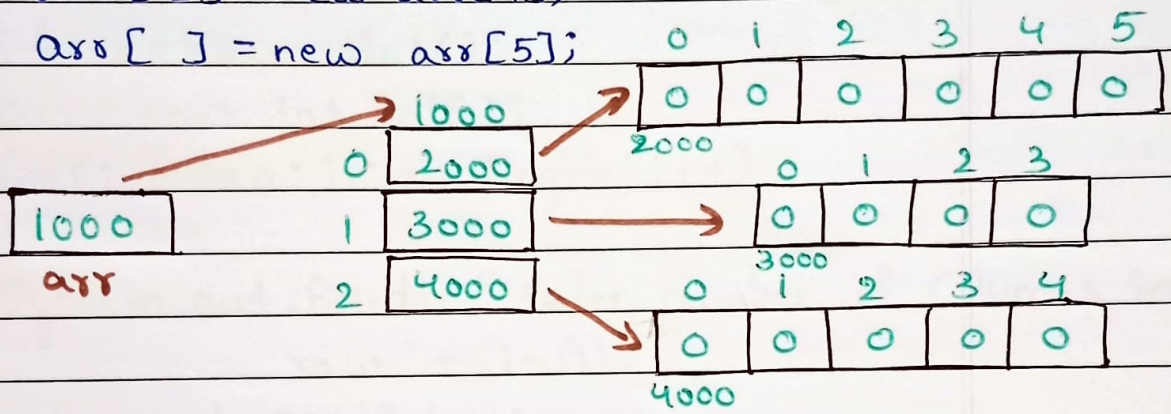
— Jagged 2D Arrays —

◎ Syntax :-

<data type> [][] <array reference>;
<array refrence> = new <data type> [size][];

Example :—

```
int[][]arr;
arr=new int[3][];
arr[0]=new arr[6];
arr[1]=new arr[4];
arr[2]=new arr[5];
```



Initializing a jagged array

1. int arr = new int[2][];
   arr[0] = new int[2];
   arr[1] = new int[3];
   arr[0][0] = 10;
   arr[0][1] = 20;
   arr[1][0] = 30; // and so on...
2. int[][] arr = new int[2][];
   arr[0] = new int[]{10,20,30,40};
   arr[1] = new int[]{50,60,70};
3. Int[][]arr = {{10,20,30},{50,60},{70,80,90}};

— Exercise —

* WAP to Create a jagged array where row and Column sizes are to be accepted from the user. Now, print all the values of jagged

array along with row wise Sum.

Solution:-

```
import java.util.Scanner;
class jaggedDemo
{
Public Static Void main(string[]args)
{
Scanner kb=new Scanner("System.in);
int[][]arr;
System.out.println("Enter number of Rows");
int r=kb.nextInt();
arr=new int[r][];
for(int i=0;i<arr.length;i++)
{
System.out.Println("Enter number of Columns in
                    row"+(i+1));
int c=kb.nextInt();
arr[i]=new int[c];
System.out.println("Enter values");
 for(int j=0;j<arr[i].length;j++)
 {
  arr[i][j]=kb.nextInt();
}
}
for(int i=0;i<arr.length;i++)
{
int Sum=0;
for(int j=0;j<arr[i].length;j++)
{
Sum=Sum+arr[i][j];
System.out.print(arr[i][j]+" ");
} system.out.println("Sum is "+Sum+"\n");
}
}
```

Object Oriented Programming

→A programming paradigm which is based on real world model of objects or entities

→It is organized around objects rather than "actions" and data rather than "logic".

→Object-Oriented programming takes the view that what we really care about are the objects we want to manipulate rather than the logic required to manipulate them.

→In programming any real world entity which has specific attributes or features can be represented as an object.

→Along with attributes each object can take some action also which are called it's "behaviors".

→In programming world, these attributes are called data members and behaviours/actions are called "functions" or "methods"

Are you an object?

→Yes, we humans are objects because:

- We have attributes as name, height, age etc..
- We also can show behaviors like walking, talking, running, eating etc..

Classes

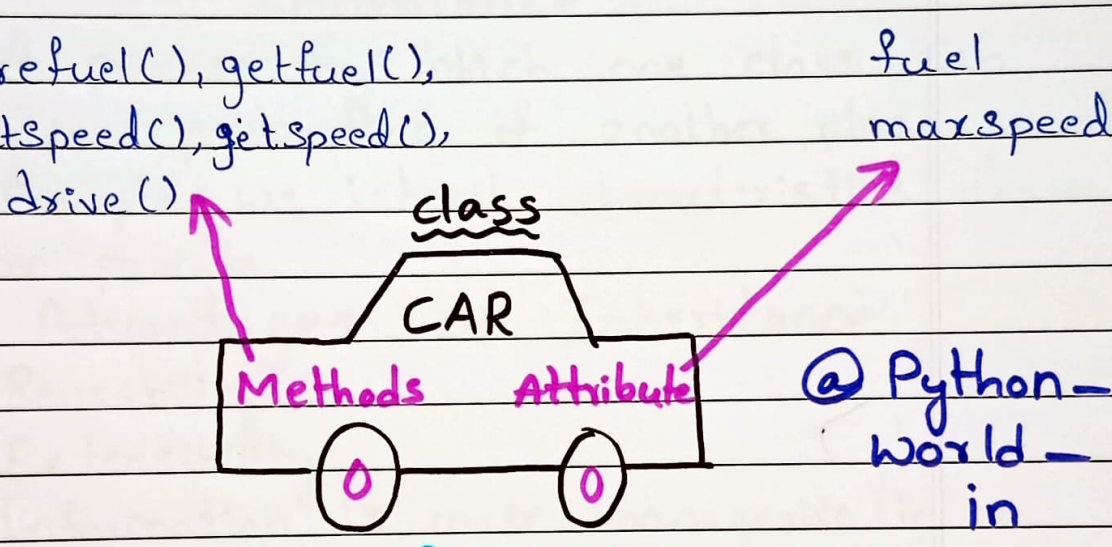
→ Now to create/represent objects we first have to write all their attributes under a single group.

→This group is called a class.

→A class is used to specify the basic structure of an object and it combines attributes and methods to be used by an object.

→ Thus. we Can say that a class represe-entation the data type and Object represents a kind of Variable of that data type.

→ for example :- Each person Collectively Come under a class called Human Being. So we belong to the class Human Being.



@ Python-World-in

Pillars of OOP

→ The object oriented Programming paradigm stands on 3 main pillars, which are :-

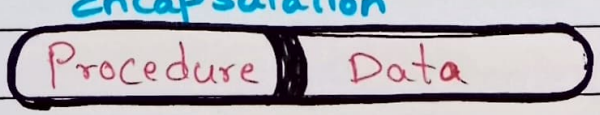
1. Abstraction and Encapsulation.
2. Polymorphism
3. Inheritance

Abstraction and Encapsulation

Abstraction : Focus on the meaning i.e. Suppress irrelevant "implementation" details.

→ Encapsulation is a process of binding or wrapping the data and the codes that operates on the data into a single entity.

Abstraction and Encapsulation go hand in hand.



Polymorphism

→ Poly - Many morph - forms.

→ polymorphism refers to a principle in biology in which an organism or species can have many different forms or stages. Have you seen how you perform poly-morphism in day to day life?

Inheritance

* A process by which one class can acquire properties of another class.
* Example :- we inherit characteristics from our parents.

Advantages of Inheritance

1. Reusability
2. Extensibility
3. Information is made manageable in a hierarchical order.

Creating a class and its object

Syntax :-

```
class <class name>
{
<access modifier> <datatype> <variable name> =
                                        value;
                    ≡
<access modifier> <return type> <method name>
                                    (arguments)
{
//method body
}
}
```

Example :-

```
class student
{
int roll;
char grade;
float per;
}
class usestudent
{
Public static void main(string [Jargs)
{
Students;
S=new student();
}
}
```

Access Modifiers

* There are 4 access modifiers provided by java which are,
  - public
  - Private
  - Protected
  - default.
* public and Private are of importance to us unless we learn "Packages" in Java.

Creating methods in a class

* The major principle of object oriented programming i.e, Encapsulation is implemented using methods in a class.
* In Java we just define a method in class, no declartion is required.

```
Class student
{
Private int roll;
Private char grade;
Private float per;
Public void SetData()
{
 roll=10;
 grade = 'A'
 Per =66.5f;
}
Public void ShowData()
{
S.o.P("Roll,grade and percentage is "+roll,grade,per);
}}
 class Usestudent
{
Public static void main(string [] args)
{
Student S=new student ();
S.setData();
S.ShowData();
}
}
```


Teacher's Signature.....

Initializing Data Members at RUNTIME

```
Public Void ShowData()
{
 S.O.P("Roll is" +roll);
 S.O.P("Grade is" + grade);
 S.O.P("Percentage is" + per);
}
class Usestudent
{
Public static void main(String[] args)
{
 Student S = new Student();
 S.setData();
 S.showData();
}
}
```

Intilalizing objects (or) Data member

* In Java to initialize an object we have 3 options Using methods is also one of them, which have Studied in the previous notes.

1. Explicit initialization
2. Using Constructors
3. Using initializer blocks.

Explicit Initialization

* Java permits us to initialize members of the class at the point of their declaration and such way of initialization is known as Explicit initialization.

* C++ Considers this way as error, but in Java this is Considered the fastest way of Initialization.
* This is preferred when all objects should have the Same value at initialization.

Constructors

* Constructors are Special methods of a class with following Characteristics.
→ They have the Same name as that of the class.
→ They don't have any return type.
→ They are Automatically called as Soon as an object is Created i.e. via new Keyword.
→ If the programmer does not provide any Constructor then Java has its own default Constructor in every class with an empty body.
→ There are no Copy Constructors provided by Java. They Can be made by programmers.

Exercise :-

WAP which Consists of a class named circle, with a data member radius. Accept radius as an input from the user and initialize the data member through parameterized Constructor. Include two more methods area() and Circumference() which Calculates area and Circumference and prints them.

Initializer Block

* In Java Instance initializer blocks are used to initalize instance data members or Say objects

* It runs each time when an object of a class is created.
* The whole block is Copied inside each constructor of the class.

Syntax :-

```
class <classname>
{
<data member>;
//initializer block
{
//initializer body
}
}
```

Benifits :- 1. Since, the whole block gets Copied in Constructor, So if there are multiple Constructors there is no need to write the same initialization again and again.

2. There are classes which do not have Constructors, they are known as Anonymous classes. Such classes Can use these blocks to initialize objects.

Drawback :-

1. Intilalizer blocks Cannot pass arguments and neither Can they return any value.

Method Overloading

⇒ Method overloading means having multiple versions of Same thing.

⇒ For example the Println() method in java Can be

Used to print integers, strings, boolean, double etc, just by calling the same method.

⇒ Some special rules to be remembered for over-loading are.

1. They differ in terms of their arguments.
2. The arguments can vary in number, data type and order.
3. Overloading cannot be done on the basis of return type of the method.

Selection of Overloaded method :-

- The Compiler Searches for the method of similar data type for which the argument was passed, if not found then,

1. It selects the overloaded method with next higher data type in terms of range.
2. If no higher range data type is available then Syntax error occcurs.

- Create a class name FigArea, with an overloaded method to Calculate area of Circle and Cylinder. These methods should return the value of area Calculated and printed in the main method. The program should ask the user about the choice of figure and dimensions.

Constructor Overloading

Same as method overloading we can overload a Constructor of a class.

The rules remain the same here also i.e each

Constructor must be unique with respect to argument.

⇒It's main benefit is for the person Creating objects of the class get a Variety of options to initialize the object of the class.

Argument Passing

* Java only allows pass by value, there is no pass by reference. But yes references are passed but Same as pass by Value.
* So we Can say that in Java we Can pass arguments as....

1. Passing Variable As Argument.
2. Passing references as argument.

Passing Variable as Argument

* In java also on passing Variables as argument, a Copy gets generated which is termed as formal argument.
* The method performs all operation on these formal arguments and hence no change Can be made on actual arguments.

Exercise

*WAP to swap two integer data members of a class. Later you Can modify it to accept input from the user.

Sample output:

x = 10
y = 20
x = 20
y = 10

Passing Array Reference

```
class Demo
{
Public Void doubler (int [ ]brr)
{
for(int i=0; i< brr.length ;i++)
  brr[i] = brr[i]* 2;
}
}
class Test
 {
  Public Static Void main(String [ ]args)
{
int [ ]arr = { 10,20,30,40,50};
Demo d=new Demo();
d.doubler (arr);
for (int i=0;i<arr.length;i++)
 S.O.P(arr [i]);
}
}
```

Returning Array Reference

```
Class Demo
{
Public int[ ] createArray (int n)
{
int [ ]brr = new int [n];
 return brr;
}
```

```
}
class Test
{
Public static void main(String [Jargs)
{
Demo d = new Demo();
int [Jarr = d. createArray(5);
System.out.println ("Length of array is" +arr.length);
}
}
```

Practice :- Design another method in the same class called Count() which should return total number of elements in the array which are greater than the number passed, Smaller than the number passed and equal to the number passed.

The "this" KeyWord

* The "this" keyword in Java is a Predefined object reference available inside every non static method of a class.
* On calling a method, the java Compiler transfers the address of the object to the called method.
* This address is Copied inside the "this" reference. In short "this" reference points to the object which is currently being used to Call a method.
* Two major benefits of Using "this" reference.

1. we Can Use the local Variables by Using the Same name as that of the data members of class.

2. We Can perform inter Constructor Call using "this".

Resolving the issue of local Variable and data member names.

- Example :-

```
class Box
{
Private int l,b,h;
Public Box(int l, int b, int h)
{
 this.l = l;
 this.b = b;
 this.h = h;
}
Public void Show()
{
S.O.P(" length = " + this.l);
S.O.P(" Breadth = " + this.b);
S.O.P(" Height = " + this.h);
}
}
```

Using "Static" Keyword

- The Keyword static Can be used at three situation i.e,

1. Static data members
2. Static methods
3. Static blocks
4. Static classes (Can be used only with nested class or inner class and not the outer class) @Python_World_in

"Static" Data members

- Using Usually, a non static data members is allocated in RAM only when an object is Created.
- Static members are Saved in RAM once, i.e. they are independent of the objects.
- A data member is made static when it should display same number change for all objects.
- For example,

```
class Data
{
 int a;
 int b;
 }
```

Both a and b will get space in memory when Object of class Data gets Created.
what if b is made static ???

```
class Data
{
int a;
Static int b;
}
 Class useData
{
Public static Void main(string [ ]args)
{
Data d1 =new Data();
Data d2 = new Data();
d1.a =10;
d2.a =20;
System.out.println (d1.a + "\n" + d2.a);
}
{
```

Static member
class member
Shared member

Features of "Static" Data member

→ Gets allocated in RAM as soon as Program is executed, irrespective of the object.

→ Only a single Copy is made.

→ Since, they are object independent they should be accessed using class name.

```
Data.b = 30;
Data.b = 40;
```

→ Static members are kept in Permanent Generation area of RAM.

→ Local Variables Cannot be made static i.e, not inside a method.

Garbage Collector

- We will understand this Concept through a practice program.
- WAP to Create a class Called Employee having the following data members.

1. An ID for storing unique id allocated to every Employee.
2. name of employee.
3. age of employee.

- Also provide following methods —

1. A parameterized Constructor to initialize name and age. ID should also be initialized in this Cons.
2. A method show() to display ID, name and age.
3. A Method ShowNextId() to display ID of next Employee.

The "object" class

* In Java every class by default inherits a class named. Object.
* this inheritance is done by Java and Cannot be avoided by any programmer.
* Object class is the Super / parent class of every class. It is Super daddy class.
* It is present in the package Java.lang.
* Object class has 8 methods in it. Hence, every class has at least 8 methods.
* Finalize() method is one of them and we Override this method.

Using "static" methods

A method should be made static in following three situations.

1. When it is only accessing static data of class.
2. When it is only accessing its arguments and not using any data members of the class.
3. When we have to Create a factory method.

```
Class Mymath
{
Public static int max (int a, int b)
{
 if (a>b)
  return a;
 else
  return b;
 }
}
```


Teacher's Signature.....

```
class Test
{
Public static void main(string [] args)
{
int max = MyMath.max (10,20);
System.out.Println ("Max is = " + max);   (OR)
System.out.Println ("Maximum number is " + MyMath.max(
  10,20));
}
}
```

Factory Methods

There are Situations where Creating an object might be dependent on Same Conditions.

For example, if Someone enters 0 or negative age. In such situations we have to check the Conditions before the object Can be made.

For Such Situations Factory methods Can be Used.

Constructors of Such classes are Private and the factory method is static in nature.

Factory methods Usually Create and return Object.

Let's See an example.....

```
Class Person
{
Private int age;
Private string name;
Private Person(int a, string s);
{
age = a;
name = s;
```


Teacher's Signature.....

```
}
Public void show()
{
S.O.P(age+","+name);
}
Static Person CreatePerson(int a, string s)
{
if(a<=0)
 return null;
else
 {
 Person P=new person(a,s);
 return P;
 }
 }
}
Class Test
{
Public static Void main(String[]args)
{
 Person  P1;P2;
P1=Person.CreatePerson(-25,"Amit");
P2=Person.CreatePerson(29,"Sumit");
P1.Show();
P2.Show();
}
}
```

Properties of "static" method.

* They are allowed to access only static data of the class implicity.
* They do not have "this reference" built in them.
* They Cannot use the keyword Super.
* They are Called directly using class name, without using objects or object reference.
* All methods of math class are static.

"Static" Blocks

* Consider the following programming situation.
* Suppose we have to Create a class Called Account for a Banking application, which shows accid, name, balance and rate-of-interest of a Customer. The challenge is that our Code should accept rate-of-interest at runtime and only once. How Can we achieve this??
* The Solution lies in using static blocks.
* Static blocks are independent blocks which are loaded in RAM and executed as Soon as the program executes.

Example

```
import java.util.Scanner;
class Account
{
Private int accid;
Private String name;
Private double balance;
Private static double rate-of-intrest;
```

```
static
{
Scanner sc = new Scanner(System.in);
System.out.println("Enter rate of interest");
rate_of_interest = sc.nextDouble();
}
Public Account()
{
Scanner Sc =newScanner(system.in);
System.out.Println("Enter account id,name and balance");
accid = kb.nextInt();
name = next();
balance = nextDouble();
}
Public Void show()
{
System.out.println(name+"\n"+accid+"\n"+balance);
}
Public static void showRate()
{
System.out.println("Rate of interest is "+rate_of_interest);
}
}
Class Test
{
Public static void main(string [Jargs)
{
Account A1 = new Account();
```

```
Account A2 = new Account();
Account A3 = new Account();
A1.Show();
A2.Show();
A3.Show();
Account.ShowRate();
}
}
```

Inheritance

→ Inheritance is one of the main pillars of object Oriented Programming.
→ Inheritance is acquiring properties of other class.
→ The class which inherits is the derived/sub.class. The class which is inherited or extended is the base/Super class.
→ Major benefit of Inheritance is Code Reusability.
→ Inheritance should be used where there is "is a" relationship. like manager is an Employee, Mango is a fruit etc.

Syntax for Inheritance

```
class <Base class name>
{
  ----
  -----
}
Class <Derived class name> extends <Base class name>
{
  ----
  ----
}
```

Types of Inheritance

* Single
* Multilevel
* Hierarchical

→ Java does not support Multiple Inheritance. WHY??

→ Because java does not support ambiguity. Assuming if there are 2 classes A and B having a method named show(). If both are inherited by a class C then there will be an ambiguity of which show() method is to be Called.

Using Keyword "Super"

→ The keyword Super is used by the derived class programmer to explicitly refer the members of its base class.

→ Using Super becomes Compulsory in 2 Programming Situations :-

1. calling base / Super class Constructor from derived class.
2. To resolve method overriding.

* From the previous example, if getIncome() method of Manager was also named getSal(), then it will struck in an infinite recursion.

Constructor Calling in Inheritance

* Constructors follow a very special rule in Case on inheritance and the rule is that, whenever the object of derived class will be created, the Constructor of base class executes first.

* Followed by the Constructor of derived class.
* This is true only in Case of non parameterized Constructor.
* If base class Constructor is parameterized, then Programmer has to explicity Call the base class Constructor from the Constructor from the Constructor of derived class.
* For doing this he has to use the Keyword Super.

Calling non parameterized Constructor

```
class A
{
Public A()
{
SoO.P("In Constructor of A");
}
}
class B extends A
{
Public B()
{
S.O.P("In Constructor of B");
}
}
Class Test
{
Public static Void Main(String [] args)
{
B obj = new B();
}}
```


Teacher's Signature.....

Calling Parameterized Constructor

→ In Case of Calling a parameterized Constructor, the derived class Constructor is invoked.

→ The programmer has to explicitly call the base class Constructor from the derived class Constructor using the keyword Super.

Exercise :-

Modify the employee Manager program in such a way that the method SetData() is replaced with a parameterized Constructor.

Method Overriding

→ A Mechanism used by derived class, to change functionalities of the same method present in base class.

→ Two necessities for overriding a method

1. Inheritance
2. Prototype (same visibility, return type and name)

→ It is generally done when the derived class wants to have a more specialized or specific version of the method inherited from the base class.

Overloading V/S Overriding

→ Although Overloading and overriding sound similar, but are completely different.

→ Overriding can only occur in case of inheritance whereas overloading is done within the same class as well as across inheritance.

→ Overriding occurs when Prototype (return type, name,

arguments) of a method is Same in both base and derived class while, Overloading occurs when name of method is Same but they differ in terms of arguments.

Relationship b/w Base class reference Derived class object

→ In Case of inheritance there is a special rule regarding Super class reference and derived class reference.

→ Rule is that the base class reference Can Point or hold the derived class object. Example, the Employee class reference Can point to Manager's object.

```
Employee e = new Manager();
```

→ But the reverse is not possible, the derived class reference Cannot point to base class object

Although reference of Super class Can point to an object of derived class but we Can only access those members which have been inherited from the Super class. Not those which are added by derived class.

Example, We Can access name and Salary but not bonus which is specific to Manager class and is not a part of Employee class.

```
Employee e=new Manager();
e.Sal = 12000.0;
e.name = "Amit";
e.bonus = 10000.0; X
```

Polymorphism Dynamic Method Dispatch.

* The word polymorphism means the ability to behave differently in different situations.
  poly — many Morphs — Forms.
* We have seen polymorphism in methods through methods Overloading and Overriding.
* Now, we shall see how we can achieve polymorphic behavior using a single base class reference and call different methods of derived class. which is called. Dynamic method dispatch.
* Before we can understand polymorphism, we need to understand a very term in programming i.e, Binding.

Binding

* the term binding means a Mechanism followed by the Compiler of a language to make function / method calls.
* Just like object oriented languages Java also follows two types of binding —
  1. Early / Compile time / Static Binding
  2. Late / Dynamic / Runtime Binding
* In Early Binding, if the method being called is a static method then Java determines the version of method to be called by looking at the object being pointed and not the reference.

A Program to achieve Dynamic Method Dispatch

```
class shape
{
private int dim1;
```

```
Private int dim2;
Public Shape(int dim1, int dim2)
{
this.dim1 = dim1;
this.dim2 = dim2;
}
Public double area()
{
return 0.0;
}
Public String name()
{
return "Unknown";
}
Public int getDim1()
{
return dim1;
}
Public int getDim2()
{
return dim2;
}
```

3.

```
class Rectangle extends Shape
{
Public Rectangle (int l, int b)
{
Super(l, b);
}
```

```
Public double area()
{
return (getDim1() * getDim2());
}
Public String name()
{
return "Rectangle";
} };
class Triangle extends Shape
{
Public Triangle (int b, int h)
{
Super(b, h);
}
Public double area()
{
return (0.5 * getDim1() * getDim2());
}
Public string name()
{
return "Triangle";
} };
Class UseShape
{ public static void main(string [] args)
{
Shape S;
S = new Rectangle (5,10);
S.O.P("Shape is" + S.name());
```

```
S.O.P(" Its area is " +S.area());
S = new Triangle (15, 20);
S.O.P(" Shape is " +S.name());
S.O.P(" Its area is ": +S.area());
} }
```

Abstract classes and Methods.

* Some methods of a class Cannot be defined as their functionalities depends on derived classes, like the Previous example methods area() and name().
* So to avoid writing their body and just declare them in base class so that they can be overridden in derived class as per their functionality, we use the keyword
* By declaring a method abstract we Can avoid defining the method, but with two Compulsions.
* The class should also be Prefixed with the keyword
* If the class is made abstract then its Object Cannot be Created but reference Can be Created.

from Previous Example :-

```
abstract class Shape
{
Private int dim1;
Private int dim2;
//--- Same as previous ---//
abstractpublic string name();
abstract Public double area();
   }
```

* Rest both derived classes and Driver class remains same. Program will show same output.

Methods which Cannot be made Abstract

* Static method :- Since, abstract is used when there is "no functionality defined yet" and static itself means "there is functionality even if you do not have object"; so, static and abstract are Completely opposite to each other.
* Constructors :- Since, Constructors are never inherited hence don't require to be overridden.
* Private methods :- These are not accessible in derived class.
* Classes which inherits abstract methods must Compulsorily override the abstract method or the derived class itself should be made abstract and thus, derived class objects Cannot be Created.

"final" Data members

* Any data member whose value must remain Unchanged throughout the program and Cannot be altered once initialized, then in this we Can Prefix Such data members with keyword final.

Example :-

```
class Circle
{
 Private int radius;
Private static final double pie = 3.14159;
```

* It's Value will remain Unchanged. Remember the data member Math.PI, it is declared the same way. they behave like Constants.
* The initialization of final data member can either be done explicity while declaring or through Constructors at run time only once. But in all the Constructors defined in a class.
* Once explicitly initialized at declaration then it Cannot be initialized again Using Constructors.
* Local Variables Can be made final as well.
* Java strongly recommends that when any data members is both final and static in nature. then it should be named in upper Case. Example, Private final Static double PI = 3.14159. It's not a rule but a Professional Coding Convention.

Example :—

```
Class Data
{
final int a;
Public Data(int x)
{
 a=x;
} }
Class Data
{
final static int a;
static
{
a=10;
```


Teacher's Signature

```
}
Public static void main( string [Jargs)
{
System.out.PrintIn(A.a);
}
```

"final" Methods

final is used with methods whose functionality remains Same throughout i.e in base as well as derived classes.

Methods which are declared finals Cannot be Over-ridden in derived classes. But they do get inherited.

```
class A
{
public final void display()
{ - - - - -
} }
class B extends A
{
Public void display() ✗
{ - - - - -
} }
```

```
class A
{
Public final void display()
{ - - -
} }
class B extends A
{
```

```
Public void print()
{
Super.display();
} }
class C
{
P S V main(String [] arg)
{
B obj=new B();
obj.display();
} }
```

* Abstract methods Cannot be made final and vise versa. Since, a method is made abstract because of its Unknown functionality, so they Can be accordingly modified in their derived classes. Whereas, a method is made final so that it's functionality should never change throughout the hierarchy.
* Abstract methods Can be called Using derived class object but Cannot be Overridden.
* Even method main() Can be made final.

"final" Classes

* Those classes which are Prefixed with Keyword final Cannot be inherited in Java.
* For example, class string is a final class and no other class Can inherit it.
* Classes are made final in Case where the data member or methods are Sensitive enough that they

should not be altered at any Cost.

* Is there any other way through which we Can prevent a class from being inherited??? (Just for knowledge)!!

By making Constructors Private.

```
final class A
{
  ---
  ---
}
class B extends A ✗
{
  ---
  ---
}
```

* Though, final classes Cannot be inherited but final classes Can inherit other classes.
* For example, every class does inherit the class object and hence a final class also does inherit the class object.

Interface

* An interface is an alternate to Pure abstract class i.e to Support run time Polymorphism
* An interface is also a Kind of Java class but with Some predefined restrictions which are as follows

1. An Interface Can Contain data members but they will by default be public, static and final by nature.
2. Even if we don't assign any visibility mode to the members of an interface, still their visibility

always remains public and a programmer is not allowed to alter it.

* An interface Can Contain methods but they will be Public and abstract in nature.

Syntax :—

```
interface <interface name>
{
 <data type> <variable> = value;
 <return type> <method name> (argument);
  = = = = =
}
Class A implements <interface name>
{
  - - -
  - - -
}
```

* A class Can inherit only a Single class at a time but a class Can implement multiple interfaces.

⇒ Example :-

```
interface Point
{
  - - - -
}
interface Shape
{
  - - - -
}
```

```
class Circle implements Point, Shape
{
  ---
  ---
}
```

What's new in interface ????

* Java 8 onwards a Special feature is added in interface which allows programmers to define methods in an interface.
* Such methods which are defined in an interface are Known as
* It helps in Situation where a class which implements an interface might not have any logic to override a method but has to forcefully override it with a blank body or just return any default value.
* In this Case any class which implements an interface is exempted from overriding default methods

Interface with Runtime Polymorphism.

* Just like we Can use an abstract class to achieve dynamic method dispatch, Similarly we Can use an interface also to achieve polymorphic behavior.
* The reference of an interface Can point to the objects of its implementation class and Can Call those methods which were declared in the interface and have been overridden by the class.
* Let us try the shape example with which we did using abstract classes.

Interface with Runtime Polymorphism

```
interface Shape
{
 double area();
 String getName();
}
class Rectangle implements Shape
{
 Private double l, b;
 Public Rectangle (double l, double b)
 {
 this.l = l;
 this.b = b;
 }
 Public double area()
 {
 return l*b;
 }
 Public string getName()
 {
 return "Rectangle";
 }
}
```

Inheriting One interface into another

* Just like we inherit a class into another Similarly, we Can inherit an interface into another.
* Through, a class Cannot extend more then one

class at a time but an interface can extend many interfaces at a time.

→ An interface Cannot implement any other interface.

→ A .class file is Created after Compilation for every interface.

Example :-

```
interface Shape
{
  double area();
}
interface Figure
{
  String name();
}
interface MyShape implements Shape, figure.
{
  ----
}
```


Packages

→ Packages are nothing but a fancy name for a floder i.e, a official or professional name for a folder by Java.

→ Packages are a Collection of related classes and interfaces i.e. classes and interfaces with Some Similar or related functionalities.

→ Java strongly advices us to group all our classes and interfaces inside a package due to following reasons

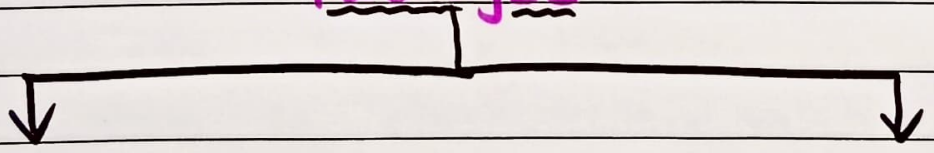
Benifits of Packages

* By making Use of Packages we Can easily resolve name Conflicts i.e. two classes Can have Same name if they are in different Packages.
* If and only if a class is a part of Package we Can import it in other programs, otherwise we Cannot import it.
* It becomes easier for us to manage our application if we keep them inside Packages. The Program in much Organised and Symmetric.

Structure of a Standard Java Program

```
Package <package_name>;
import ---
import ----
class <class_name>
{
  ----
}
```

* Java recommends to name packages in lower Case. For examples, java.lang, java.util, etc.. Package names are also Case (Senstitive) Sensitive.



Creating packages in "bin"

```
Package myjavacodes;
class Test
{
Public static Void main (string [] args)
{
System.out.Println ("This is a message from myjavacode");
}
}
```

Create a folder in bin with Same name as that of your package and then Save the file in the folder.

Compile :- c:\--\bin > javac myjavacodes\Test.java

Execution :- c:\--\bin > java myjavacodes.Test.

Creating two java files in the Same Package.

Num.java

```
Package myjavacodes;
class Num
{
Private int a;
Private int b;
Private int c;
Void set (int i, int j)
{
a = i;
b = j;
}
Void add()
{
c = a+b;
}
```

```
Void show()
{
S.o.P("Numbers are "+a","+b);
S.o.P(" Sum is "+c);
}
}
```

UseNum.java

```
Package myjavaCodes;
Class UseNum
{
Public static Void main(String []args)
{
Num obj=new Num();
obj.set(10,20);
obj.add();
obj.Show();
}
}
```

Compiling Program

* If we Compile UseNum.java, then automatically Num.java also gets Compiled and you will see two .class files will be Created i.e. Num.class and UseNum.class . this is because, we Use an object of class Num in class UseNum.
* But if you Compile Num.java then only a Single Num.class file will be formed.
* Java also Supports wild card Compilation, which is javac myjavaCodes*.java.

* It will Compile all the .java files present in the Package myjavacodes.

Execution :- Java myjavacodes.UseNum.

Creating Programs outside "bin"

```
class Greetings
{
Public static Void main(String[] args)
{
System.out.Println("Good evening User!");
}
}
```

* Save the above program in the main C: drive of your Computer with name "Greetings.java". Now, Compile the Program.

Setting PATH

→ Since, javac is an executable file(.exe) so, to execute it the DOS operating System looks for it in the Current drive Or folder. If not found, Dos looks for it in its PATH directory or library.

→ PATH is Called an environment Variable, which is an OS Variable within which we Can set locations of all those programs which we want to run from anywhere in our System.

→ In short we Can Say that, any program whose location is added to the PATH variable becomes globally accessible in out System.

To set PATH we have 2 options -

1. Temporary Setting is done via Dos window and remains until the window is open. General Syntax for setting PATH is

Set Path = <path to the desired location>;%path%

Ex: set path = c:\programfiles\Java\jdk 1.8.0-72\bin;%path%

2. Permanent Setting is done in following steps -

a) Right click on my Computer → Properties → Advanced System Settings → Environment Variables → new →

Variable name : Path

Variable Value : c:\programfiles\Java\jdk 1.8.0-72\bin : %path%

Click OK

Creating Package Outside "bin"

* let's create the Previous program in main C drive
* Now instead of Creating folders manually, Java's Compiler is powerful enough to Create packages. This Can be done through the following Command-

c: javac -d . <file name>.java

Switch on Current location

Create a directory

* The above Command does two things,

a) Builds a new package if package does not exist.

b) Create a byte Code file and adds it to the Package.

In place of "." if we specify any other location or drive then the Compiler Creates Package in that particular drive.

Accessing classes outside their Package

→In order to access a class outside its package we have to follow Certain steps —

1. We have to import it.
2. To import a class we must prefix that class with the keyword Public. In simple terms we Can Say that a class Cannot be imported Outside the package Unless it is declared with the keyword public.

→Only by making a class as public we get right to access it outside the Package and Create its object. But, to be able to Call its methods outside the Package they must also be declared Public.

→Only public members of the class Can be accessed outside the Package.

* If a class is public then there is another rule which programmers have to follow —
* The name of .java file should be Same as the name of Public class.
* If we have 10 classes and all of them should be accessible outside the Package then all these classes should be Saved in their own respective, .java files and each .java file should have Same name as public class.

Setting CLASSPATH

* CLASSPATH just like PATH is another environment Variable which is set to be able access third party packages i.e. those packages which are not available in Current location or java's Original library.
* If we recall our previous example, the class UseNum Creates an object of class Num in it. Since, Num is Created by us as a programmer, it won't be enough to just import it. we have to set its classpath and then we Can import it in other program.
* To set classpath we again have 2 choices.

1. Temporary Setting —

Set classpath = c:\; %classpath%;.

2. Permanent Setting —

Right click on my Computer → Properties → Advanced System settings → Environment Variables → new →

Variable name : Classpath

Variable Value : <location>; %classpath%;.

Click OK.

Access Modifiers or Visibility Modes

Acess Modifier	Private	Public	default	Protected
→ Same class	Yes	Yes	Yes	Yes
→ Non Sub class Same Package	No	Yes	Yes	yes
→ Sub class Same Package	No	yes	Yes	Yes
→ Non Sub class & different Package	No	yes	No	No
→ Sub class and Different Package	No	yes	No	Yes

Method Overriding and Visibility Modes :—

* When we override methods of base class in the derived class, then access modifiers play an important role and Java forces programers to follow a particular rule.
* Rule is that, while overriding a method of base class in its derived class either we Can keep the Same access modifier or we Can use less restrictive access modifier.
* For example, if a method is in default visibility in base class then while overriding it we Can make it protected or public but we Cannot make it Private.

Using Static import

* This feature was added from Java 5.
* This features facilitates Java programmer to use static members directly without using its class name.
* Its only advantage is less Coding is required to access the static members of the class.
* Let us see an example to Understand this....

```
import static java.lang.System.*;
class Test
{
Public Static Void main(String [Jargs)
{
out.println("Hello User!");
}
}
```

Using Static import

* Only available for static members of the class. We Cannot Use it for non static members of the class.
* If a local member is present with the same name as that of the static member, then the local member Overlaps the static member.
* If two two static members of different class have the Same name, then in this Case Using them directly will Create an Syntax error.

Exception Handling

* Exception in programming languages like java means runtime errors i.e errors which appear during execution of a Program.
* It might be due to User's wrong input or any logical fallacy of the program.
* Exception handling is the behavior of a Program after an exception occurs.
* But before handling Understanding how to exception, first let us Understand what java does when an exception occurs.
* By default java takes 2 actions whenever an exception occurs -

Exception Handling How java handles it????

* It immediately kills the program on the line where the exception occurs.
* It defines the reason for exception but is highly technical and is not friendly to an User.
* Both the above actions are not user friendly because,
* If exception occurs at least those lines should Continue to run which are not related with the exception.
* It would be much better if our program displays an easy to Understand message regarding the exception So that the User Can become aware

about his mistakes.

Exception Handling Keywords

* Java provides us keywords which can be used to write handle exceptions in programmers own way, which will be much more user friendly –

1. try
2. Catch
3. throw
4. throws
5. finally

Syntax :- **try and Catch**

```
try
{
 ---
}
Catch (<Exception class name> <object referenee>)
{
 ---
}
```

There cannot be any other line between a try and a Catch block, they should be Continuous.

A try block can have multiple Catch blocks.

All exceptions are pre defined classes in java.

if no Catch block matches the exception object then java show its default behaviour.

Exception Hierarchy

Throwable

Error

It represents those exceptions which are not meant to be handled by programmers.

- They are either handled by JVM or OS.

Exception

- This class represents those exception which can and should be handled by a programmer in his program.
- All exception classes are derived class of Exception class.

Exception

RuntimeException

→ Arithmetic Exception
→ No Such Element Exception
Input Mismatch Exception
→ Number Format Exception
→ Index Out of Bounds Exception.
→ Array Index Out of Bounds Exception.
→ String Index Out of Bounds Exception.

IOException

- File Not found Exception
- EOF Exception
- Malformed URL Exception
- Socket Exception
- Null pointer Exception

SQL Exceptio

Java's rule on multiple Catch.

* Java has a very strict rule while using multiple catch for a try block.
* They rule is that, a parent class of exception hierarchy Cannot Come before its child class.
* This is because a reference of parent class Can easily point to the child class object and hence, the child class Catch block will never run.

Example :——

```
try
{
 ---
}
Catch(IO Exception e)
{
 ---
}
catch(File Not found Exception f)
{
 ---
}
```

```
try
{
 ---
}
Catch (File Not found
Exception f)
{
 ---
}
Catch(IO Exception e)
{
 ---
 ---
}
```

Exercise :——

● WAP to accept 2 integers from the user and display the result of their division and Sum. Your program should behave in the following way—

* If both the inputs are integers and are valid then the program should display the result of their division and Sum.
* If denominator is then program should display relevant error message but should display the Sum.
* If input value is not an integer then the Program should display relevant message and neither division nor Sum should be displayed.

Obtaining description of an Exception.

* whenever an exception occurs in try block, java Creates an object of a specific Exception class and stores Some important details in the object.
* Now using that object we Can obtain all the possible details of the exception that occurred in the Catch block.
* To do this we Can use Several methods Using the object reference. Some important and Common methods are.

Methods

→ Public string getMessage():-
→ This method is present in the class Throwable.
→ The method returns "error message" generated by java regarding the exception.
→ Example :—

```
Catch (Exception e)
{
System.out.Println (e.getMessage());
}
```

- Public string tostring() :-

→ This method is present in the object class hence, in all classes.

→ It is invoked in two Situations.

1. In method println().
2. Concatenation of object and String.

Overriding the "tostring()" Method

- Example :—

```
Class Person
{
Private int age;
Private string name;
Public Person(int age, string name)
{
this.age = age;
this.name = name;
}
}
class UsePerson
{
Public static void main(string[] args)
{
Person p=new Person (21,"Amit");
System.out.println(p);
}
}
```

Person@15db9742 — This is Called Hashcode.

* A hashcode is an unique number generated from any object. This is what allows objects to be stored / retrieved quickly.
* But it is of no use to programmer or User.
* On passing object reference to Println() method, tostring() is invoked. Since, we have not overridden it so the base class version gets Called and hence we get hashcode in return.
* So, by overriding the tostring() method we Can get information related to a specific class and not hashcode.

Using "throw" keyword

* In java, exceptions occur on JVM's choice i.e. anything against java's flow. Example, if denominator is 0 then java itself generates Arthemetic Exception.
* But, in Some Situation a programmers may want to generate or define exceptions on their choice.
* To do this, java provides us the keyword throw. It's Syntax is :-

throw < Some exception class object reference >;

Example :-

- WAP to accept two integers from User. If the denominator entered is 0 then show java's exception message and if the numerator entered is 0 then generate your own exception displayed "Numberator must be positive".

Classification of Exceptions

Checked Exceptions

- Exceptions for which java forces the programmer to either handle it Using try-catch or the Programmer must inform or warn the Caller of the method that his code is not handling the checked exception.
- The warning given is through the keyword throws. It is Used in the method's prototype along with exception class name.

Unchecked Exceptions

- Exceptions for which java never Compels the Programmer to handle it.
- The programme would be Compiled and executed by Java.

Example the class Runtime Exception and all its derived classes fall in this Category.

Checked Exception

This in java is Called handle of declare rule.

The Caller also has 2 options, either Use try-Catch or the keyword throws.

If every method uses throws, then ultimately the responsibility goes to the JVM which will follow the standard mechanism of handling the exception.

All exceptions which are not derived classes of Runtime Exception fall in this Category. Like SQL Exception, IO Exception, Interrupted Exception etc...

Programmer defined / Customized Exceptions

→ Many times, in some situations a programmer might not find any Predefined exception classes to be used with throw.

→ for example, in case of a banking application the method withdraw has some minimum limit like 500 else an exception will generate. In this case there is no predefined java's exception class.

→ So, java advises us to design our own exception classes. Such classes are called Customized exception class.

→ To Create an exception class in our own exception class.

1. Provide a parameterized Constructor so that exception message can be set and passed on to parent class's Constructor.
2. Inherit or extend any of the predefined exception class in our own exception class.

→ The Customized exceptions become checked exception if we inherit the base class Exception.

→ Since, only Runtime Exception and its child classes are unchecked in nature, so programmer has to specifically inherit anyone of those to Create and Unchecked exception class.

Using the keyword "finally"

→ There are certain statements in our program whose execution is so crucial that before our program gets terminated, these statements must be executed.

→ Example, if we have opened a file or any database Connection and before the Program Completes its execution the file or database Connection should be closed.

→ In such case java suggests us to write such statement in a block whose execution is guaranteed by java and such blocks are created using the keyword finally.

→ If no exception occurs finally block is executed.

→ If an exception occurs inside the try block and its catch has been defined, in such case also finally is executed.

→ Even if no catch block is used then also the finally block is executed.

→ Moreover, a try block is required for an finally block. If an exception occurs outside try block in that case finally block is not executed and also when the method System.exit() is used.

Syntax :-

```
try
{
---
}
```

```
catch(---)
{
----
}
```

```
finally
{
====
}
```

Multi Catch feature

→ We Can Use a Single Catch to handle multiple exceptions.

→ This feature was introduced in java from 7th Version.

Syntax :-

```
Catch(Arithmetic Exception | Invalid Numerator Exception ex)
{
System.out.Println(ex.get Message);
}
```

String Handling

→ Java provides 3 classes to handle Strings as per situation, these are.

1. String
2. StringBuffer
3. StringBuilder

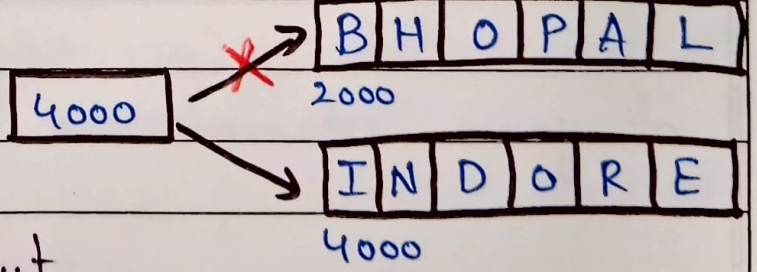
* StringBuilder will be Covered in the Multi-threading chapter.

String class

→ String objects in java are immutable i.e. Content once stored Cannot be changed.

→ For Example,

```
String city = "Bhopal";
System.out.Println(city);
city = "Indore";
System.out.Println(city);
```

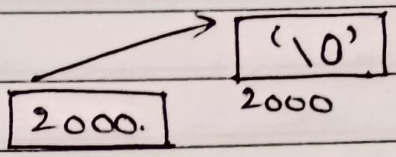


Though the output will change but the objects won't.

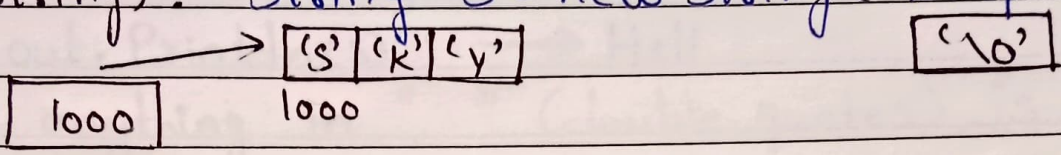
@Python_World_in

Constructors of String

→ String ( ) :- String S = new string ( );



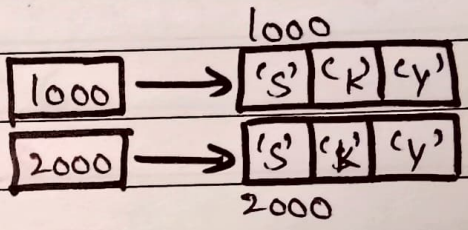
→ String (string) :- String S = new string ("Bhopal");



→ Difference in initialization :—

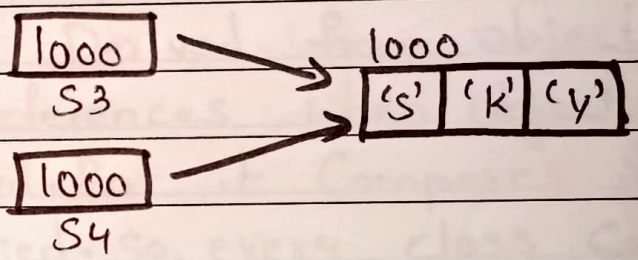
String S1 = new String ("Sky"); S_1

String S2 = new String ("Sky"); S_2



String S3 = "Sky";

String S4 = "Sky";



Constructors of string

→ To check the memory diagram we can compare the object references,

```
String S1 = new String ("Sky");
String S2 = new String ("Sky");
String S3 = "Sky";
String S4 = "Sky";
System.out.Println (S1==S2);
System.out.Println (S3==S4);
```

→ String (char [ ]) :- Converts a character array to string object.

→ String (char [ ], int 1, int 2) :-

int 1 - starting index.

int 2 - Number of characters to be Converted into String.

Char arr[] = {'H', 'e', 'l', 'l', 'o'};

→ String s = new String (arr, 0, 4);

System.out.Println(s); → Hell

→ In java anything in " " (double quotes) is Considered to be a String to be Precise a String object.

→ Example :- "Bhopal".Length(); → 6

Methods of String class

⇒ Public boolean equals (object) :-

Derived from object class. It Compares object references when object of any other class is passed. But it Compares the string when a string is passed. So, every class can override equal in its own way.

⇒ Public boolean equalsIgnoreCase (String) :-

Method belongs to string class and ignores Case Sensitivity.

⇒ Public int CompareTo (string) :- Method belongs to string class and Compares String and returns 0 if true else difference of their ASCIIs

⇒ Public int CompareToIgnoreCase (string) :-

Similar to above method but ignores Case Sensitivity.

⇒ Public int indexof (int) :-

Returns index of the character present in the string which is passed in the argument. If not found return

-1. It is a Case Sensitive method.

➡ Public int indexOf(String) :- Accepts a substring as argument and returns the beginning index where the substring occurs.

➡ Public int length() :- Gives length of string.

➡ Public char charAt(int) :- Takes index number and gives character at that index.

➡ Public void getchars(int, int, char[], int) :- Takes multiple characters and pastes their copy to an array of characters.

➡ Public boolean startsWith(String) :- Test if this string starts with the specified Prefix.

➡ Public boolean startsWith(String, int) :- Test if this string starts with the Specified Prefix beginning a Specified index.

➡ Public boolean endwith(String) :- Tests if this String ends with the Specified Suffix.

➡ Public int lastIndexof(int) :- Returns the index within this string of the last occurrence of the Specified character.

➡ Public int lastIndexof(String) :- Returns the index within this string of the rightmost occurrence of the specified substring.

➡ Public String substring(int, int) :- Returns a new string that is a substring of this string. The first argument is starting index for substring and Second argument is end index - 1 of the substring.

⇒ Public string Substring(int):- Returns the string representation of the passed data type argument.

⇒ Public static string ValueOf(any primitive datatype): Returns the string representation of the passed data type argument.

⇒ Public string Substring(int):- Returns the substring from index passed as argument till the last index of the string.

⇒ Public string toUpperCase():- Converts all the characters of the string to upper Case.

⇒ Public string toLowerCase():- Converts all the characters of the string to lower Case.

CLASS StringBuffer

* The objects of class StringBuffer in java are mutable i.e. Content of an object can be changed without Creating a new object.
* String-Buffer is Used when data of a class may change in future. Example, Salary of an Employee.
* String Buffer also has Same methods as that of the class string except Some of them
* StringBuffer is also present in the package java.lang.

Constructors of StringBuffer.

⇒ Public StringBuffer():- Creates an object with size 16 characters initalized with '\0'.

⇒ Public StringBuffer(int):- Creates a string buffer

with Specified Capacity in the argument and initalized with null character S.

⇒ Public StringBuffer(string) :- The object is Created and initialized with the String passed in the argument and is appended with 16 null characters ('\0').

⇒ Public int Capacity( ) :- This method returns the Current Capacity. Using this method we can Confirm the extra 16 Characters reserved by java.

⇒ Public Void ensureCapacity(int) :- Increases Capacity to argument passed.

⇒ Public StringBuffer append(string) :- An overloaded function and append any data type.

```
StringBuffer S = new StringBuffer("India");
S.append(" is my Country");
System.out.Println(S);
```

⇒ Public StringBuffer reverse( ) :- As the name Suggests it reverses the Original String

⇒ Public StringBuffer replace(int, int, String) :- This method replaces the characters in a Substring of this Sequence with characters in the specified string.

```
StringBuffer S = new StringBuffer("Hello World");
S.replace(6, 11, "India");
System.out.Println(S);  ——— Hello India.
```

What is A Collection ???

As the name indicates Collection is a group of objects known as its elements. Examples of Collections are :

→ List of email ids
→ List of Names
→ List of Phone numbers
→ Records of student
→ Records of books etc..

How java Supports Collections????

To handle such collection of objects, java offers us a huge set of predefined classes and interfaces called as "The Collections Framework" which is available in the package "java.util".

Why do we need to learn Collection???

The first question which comes in mind of every programmer is

*Why should I use Collection classes when I have an array?

Answer :-

Although arrays are very useful data storage structures but they suffer from several important drawbacks which are :—

1. size needs to be declared at the time of declaration, so can only be used if we know beforehand how many elements we would be storing.

2. Remains of fixed size
3. No ready made methods for performing operations like inserting, removing, Searching or Sorting.
4. Arrays are not based on any popular Data structure.
5. Can only hold homogeneous data elements

Advantages of Collections

1. Can dynamically grow or shrink
2. Reduces programming effort
3. Increases program Speed and quality.
4. Provide Predefined methods to perform all C R U D operations.

Arrays V/S Collections

Array	Collections
Arrays are fixed in size, So once we have Created the array we can not increase or decrease it's size	Collections are growable by nature so after Creation we can increase or decrease their size.
Array can hold Primitives as well as objects.	Collections don't work with primitives, they only can hold objects.
Arrays can hold only homogeneous data	Collections can hold both homogeneous as well as heterogeneous data.

Good Performance but Poor memory Utilization. Coding is Complex	Poor Performance but good memory Utilization Coding is Easy.

Types of Collections

There are 3 main types of Collections:

- Lists :- always ordered, may Contain duplicates and Can be handled the same way as usual arrays.
- Sets :- Cannot Contain duplicates and Provide random access to their element.
- Maps :- Connect Unique keys with values, Provide random access to its keys.

Important Methods of Collections

The Collection interface is one of the root interfaces of the java Collection classes. The general methods list of the Collection interface is:

1. boolean add (object obj)
2. Void clear()
3. boolean Contains (object obj)
4. boolean equals (object obj)
5. int hash Code()
6. boolean isEmpty()
7. iterator iterator()
8. boolean remove (object obj)
9. int size()

Collection V/S Collections

* For beginners there is a point of Confusion regarding the terms Collection and Collections
* Collection in java is an interface available int the Package java.util and it acts as the Super interface for all Collection classes like ArrayList, LinkedList, Hashset etc..
* Collections is a Class in the Package java.util which Contains Various static methods for performing utility operations on Collection Classes.
* Some of it's popular methods are Sort(), Copy(), binary Search() etc.

The List interface

* The java.util.List interface is a Subtype of the java.util.Collection interface and represents an ordered Collection (Sometimes Called a Sequence).
* It means we Can access the elements of a List in a Specific order and by an index too
* It allows duplicate objects.
* Each element is inserted and accessed in the list Using it's index

Implementation classes of "List"

* we Can choose between the following List implementations in the java Collections API:

→ Java.util.ArrayList

→ java.util.LinkedList
→ java.util.Vector
→ java.util.stack

The "ArrayList" class

⟹ Array List implements the List interface.
⟹ Array List is Created with an initial size of 10.
⟹ Array List Capacity grows automatically
⟹ It allows duplicate elements.
⟹ Insertion order is preserved in the ArrayList.

Creating The "ArrayList" Object

⟹ ArrayList. Can be Created in 2 ways:
Type Unsafe
AND.
Type Safe

Type Unsafe ArrayList

⟹ Type Unsafe ArrayList Can be Created as shown below.

- ArrayList obj = new ArrayList()

. Although they are easier to Create but we Cannot check what kind of data we are adding in the ArrayList.

for ex:—

```
obj.add("Amit");
obj.add(25);
obj.add(true);
```

All the above lines will successfully Compile and run.

Type Safe ArrayList

⟹ Type Safe ArrayList Can be Created as shown below.

```
ArrayList <String> obj = new ArrayList <String> ();
```

⟹ The < > is Called diamond operator in java and was introduced from java 7 onwards.

⟹ It tells the Compiler to only allow programmer to add String values in the ArrayList.

⟹ Any other type of value Cannot to added in the ArrayList and if we try to do so, the Compiler will generate Syntax error.

⟹ for ex:

```
obj.add ("Amit"); // Correct
obj.add (25);     // wrong
obj.add (true);   // wrong
```

Inserting Elements in ArrayList

⟹ To insert an element in the ArrayList, we have to Call the method add()

⟹ This method has 2 versions:

⟹ Prototype :-

- Public boolean add (object)
- Public void add (int, object)

⟹ The first method accepts an Object as argument and adds that object at the end of the ArrayList.

⟹ The Second method accepts an index number as well as an object as argument and adds the object at the Specified index.

⇒ If index is out of range then it throws the exception IndexOutOfBoundsException.

⇒ for Ex :-

```
ArrayList<String> cities = new ArrayList<>();
cities.add("Bhopal");
cities.add(0, "Indore");
```

Retrieving Elements of ArrayList

* To retrieve an element from the ArrayList we have to Call the method get().
* Prototype :-

```
public object get(int index)
```

* This method accepts an index number as argument and returns the element at that position.
* If index is out of range then it throws the exception Index Out of Bounds Exception.

for ex :-

```
String S=cities.get(0);
String p=cities.get(1);
System.out.Println(s); // will show indore
System.out.Println(p); // will show Bhopal
```

Checking Size of ArrayList

* Size of an Arraylist means total number of elements Currently present in it.
* To retrieve size of an ArrayList, we have a method Called Size() whose prototype is:
* public int size()

for ex : int n=cities.size();

Exercise 1 :- WAP to store names of first four months in the ArrayList and them print them back.

Retrieving item From ArrayList Using Enhanced for :—

→ We can traverse an ArrayList also using enhanced for Loop.

→ Using Enhanced for loop :-

```
for (string item: cities) {
System.out.Println ("retrieved element:" + item);
}
```

Searching An Element in ArrayList

Sometimes we need to check whether an element exists in ArrayList or not.

→ For this purpose we can use Contains() method of java. Contains() method takes type of object defined in the ArrayList Creation and returns true if this list Contains the Specified element.

for ex :—

```
boolean found = cities. Contains ("Bhopal");
```

Removing an item from ArrayList

There are two ways to remove any element from ArrayList in Java.

The method to be Called is remove()

This method has 2 versions:

Prototype :-

→ Public boolean remove (object)

→ Public Object remove (int)

* we Can either remove an element based on its index or by providing object itself.

- Cities.remove(0);
- Cities.remove("Indore");

Introduction To Custom ArrayList

What is a Custom ArrayList?

* A Custom ArrayList is an ArrayList which Can hold objects of programmer defined classes.
* For example, Suppose we have a class called Emp and we want to store Emp objects in the ArrayList.
* Then Such an ArrayList will be Called Custom ArrayList.

Creating A Custom ArrayList

How do we Create a Custom ArrayList?

→ To Create a Custom ArrayList, we use the following.

Syntax :-

ArrayList <name of our class> refName = new ArrayList<>();

for ex :-

ArrayList <Emp> empList = new ArrayList <>();

Creating A Custom ArrayList

How do we add objects in a Custom ArrayList?

To add objects of our class in a Custom ArrayList, we use the same Syntax as before, i.e. by Calling the method add()

for ex :-

```
ArrayList<Emp> emplist = new ArrayList<>();
Emp e = new Emp(21, "Ravi", 50000.0);
Emp f = new Emp(25, "Sumit", 40000.0);
empList.add(e);
empList.add(f);
```

Points To Remember :-

If we are adding objects of our own class in ArrayList then we must always override the (equal) equals() method inherited from the Super class Object.

⟹ This is because whenever we will Call the method remove() on the ArrayList object, it internally Calls the equals() method of our class.

⟹ This also happens when we Call the methods indexof() or Contains()

⟹ Now if we do not override this method in our class then the equals() method of object class will get Called and as we know the equals() method of object class Compares memory addresses of 2 objects.

⟹ So even if objects are having same data member values then also equals() method of object class will return. false.

⟹ Thus the methods remove(), indexof() and Contains() will fail to find our Object in the list

How To sort The ArrayList???

* The Java language provides us predefined Sorting functions/methods to sort the elements of Collections like ArrayList.
* Thus we do not have to write our own logic for sorting these Collections.
* This is done using a class called "Collections" in the package java.util which Contains a static method Called Sort() which Can sort an ArrayList
* The prototype of the method is:

Public static Void Sort(List L)

* This method accepts a List as argument and Sorts it's elements in Natural order.

What is Natural Order?

* Natural order means the default Sorting order which is as follows:
* If the List Consists of string elements it will be sorted into alphabetical order.
* If it Contains integers it will be Sorted in numeric order.

* If it Consists of Data elements it will be Sorted into Chronological Order.

How To Sort Custom ArrayList ??

* But when we Call the Sort() method of Collections class and pass it our Emp list then it will generate an error.
- Can you guess why ?
* This is because we have not defined any Sorting order for our Emp objects !!!

Solution :-

- To Solve this problem we will have to Supply the information to Collections Class about how to Sort the Emp list.
- This is done by implementing an interface Called Comparable and overriding it's method Called Compare To() which has the following prototype:
- Public int Compare To (Object)

Summary of Benefits

* Maintains the insertion order of elements.
* Can grow dynamically.
* Elements Can be added or removed from a Particular location.
* Provides methods to manipulate stored objects.

The TreeSet class :-

* This class implements the Set interface.
* The TreeSet class is useful when we need to extract elements from a Collection in a Sorted manner.
* It stores its elements in a tree and they are automatically arranged in a Sorted order.

Program :—

```
* import java.util.*;
  public class TreeSetDemo {
        Public static void main(string args[]) {
        TreeSet<String> ts = new Treeset<>();
         ts.add("C");
         ts.add("A");
         ts.add("B");
         ts.add("E");
         ts.add("F");
         ts.add("D");
         System.out.println(ts);
      }
    }
```

output :-

[A, B, C, D, E, F]

(Trans) Traversing a Treeset

```
* import java.util.*;
  Public class SetDemo {
     Public static void main(string[]args) {
```

```
Treeset < String > st = new Treeset < > ();
St.add ("Gyan");
St.add ("Rohit");
St.add ("Anand");
Iterator itr = st.iterator();
While (itr.hasNext()) {
    String str = (string) itr.next();
    System.out.println(str);
  }
 }
}
```

Output :-

Anand
Gyan
Rohit

Adding Custom objects To Treeset

* Suppose we want to Create a Treeset of Book object and we want to get the output in ascending order of Price.
* Now, if we write :

```
Treeset < Book > ts = new Treeset < Book > ();
Book b1 = new Book ("Let us C", "Kanetkar", 350);
Book b2 = new Book ("Java Certification", "Kathy", 650);
Book b3 = new Book ("Mastering C++", "Venugopal", 500);
ts.add(b1);
ts.add(b2);
ts.add(b3);
```


Then java will throw a ClassCast Exception.

Why Did This Happen ?

* This is because for any object which we add to the Treeset . Created Using default Constructor then 2 Conditions must be Compulsorily Satisfied :
* The objects added must be homogeneous.
* The objects must be Comparable i.e. the class must implement the java.lang.Comparable interface.
* In our Case first Condition is satisfied but since Book has not implemented the Comparable interface, a class Cast Exception arised.
* So to solve the above exception, we must implement the Comparable interface in Our Book class.

Exercise - 8

* Modify the Library management System Code by adding One more feature which is to print Books in ascending order of Price. Also display the Books one by one Using iterator.

SoMe Extra methods of Treeset

Since Treeset implements Navigable Set and Sortedset interfaces, it has Some more methods as Compared to Hashset. Some of it's very important methods are :-

* Public object last() :- Returns the last (highest) element Currently in this set.
* Public object first() :- Returns the first (lowest)

element Currently in this set.

* Public object lower(object): Returns the greatest element in this set strictly less than the given element, or null if there is no such element.
* Public object higher(object): Returns the least element in this set strictly greater than the given element, or null if there is no Such element.

Hashset V/s Treeset

* HashSet is much faster than Treeset as it has a Constant-time for most operations like add, remove and Contains but offeres no ordering guarantees like Treeset.
* Treeset offers a few handy methods to deal with the ordered set like First(), last(), etc which are not present in Hashset.