Functions:-

i) open funct^n:- Which are defined outside the class. (Not part of class).

ii) member funct^n:- Which are defined inside the class (are part of class).

- In C :- open funct^n
  C++ :- open & member functions.
  python :- " "
  Java :- member funct^n, even main() has to be a M.F & Hence everything has to be part of class.

Generally what happens:-

Library
↓
Packages --> Nested packages could be the
↓
classes or interfaces
↓
Functions

* There are ~~7~~ Seven Librarylies in Java.
→ Java, Javax, Sun, Sunw, com, org, Launcher.

- So, it is speculated that the real name of this Language is "Oak" and Java is the name of its Library.

Eg:- final int a = 10;

- It is always recommended to use this Notations.
- The name given to any variable, constant, object, classes, packages, libraries and functions are called as identifiers.

* Java Notation or Naming Notations.

i) Lowercase notations:- All the letters of an identifier are in lowercase.
applied to:- variable, objects, packages, libraries.
Ex:- out, in, io, err, java, javax, util, lang, etc.

ii) Uppercase notations:- All the letters of an identifier are in uppercase.
applied to:- constants.
Ex:- PI, MAX_PRIORITY, MIN_PRIORITY, LEFT, RIGHT, CENTER, etc.

iii) Mixed case Notation (Pascal case):- First letter of every internal word is capital including first word.
applied to:- classes & interfaces.
Ex:- DataInputStream, BufferedReader, ArrayIndexOutOfBoundsException, LayoutManager, System, Scanner, Runnable; etc.

iv) Camelcase Notation:- First letter of every internal word is in uppercase, excluding first word.
applied to:- functions.
Ex:- getSelectedItem(), nextInt(), nextFloat(), readLine(), toString(), println(), next(), etc.

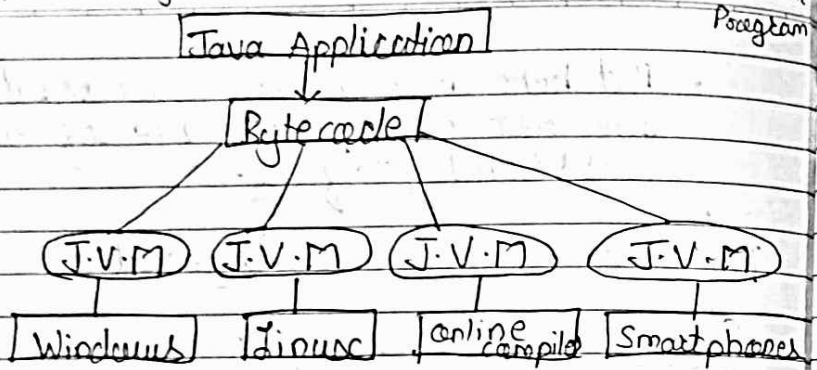
** • "Java is a single rooted hierarchy"
OR
basmic super-class (Base class) object.

- So, basically Java has a class "Java.lang.Object" which is default base class of every in-built classes or user-defined classes.
- This was done to prevent overwriting i.e java had to provide some basic functionality to all its classes and Hence they created one sepearated class containing all that methods and they made it the base class/super-class of all the classes.
- We'll Notice it much often that in Java every class has it's own small hierarchy to get the functionality carry forwarded from respective base class.

* • Hence we say that Java has became powerful becoz of this hierarchical inheritance.

- There are some classes which are not following any hierarchy except "Object class" being there.
Ex:- System.out.println(), So, System class is just a single class/Single level hierarchy.

- If you see '$' sign in bytecode that indicates that class is nested in any class side
- Inner class => "outerclassname $ innerclassname.class"
- Nested class => "outerclassname $ innerclassname.class"
  Here 'I' represents only one 'I' class is nested of that name in whole Program



- From above diagram we can say bcoz of bytecode Java follows "Create anywhere run anywhere" princple.
- We can make/declare class inside a class

i) Inner class.

```
class A
{ public void method1()
  { = }

  class B
  { = }

  class C
  { = }
}
```

In this innerclasses will have bytecode name as "outerclass $ innerclass .class"

Ex:- A$B.class
A$C.class

★ • Class can be created anywhere or we can say class can be created at every possible place where we are allowed to declare a variable
• This means in function parameter list also we can declare it (class)

ii) Nested classes.

```
class A
{ public void method1()
  { =
    class B
    { = }

    class C
    { = }
  }

  public void method2()
  { =
    class D
    { = }
    =
  }
}
```

Bytecode name
A$1B.class
A$1C.class
A$1D.class

★ • Inner classes can access member of outer class without creating an object but outer class needs to create object of inner class to use their Member functions.

* • Method overloaded methods can only call it's M.F & M.F's of its Base class.
• Any "object" can only call it's M.F & M.F's of its Base class.

hashcode

1) finalize():- protected void finalize() - throws Throwable.

2) toString():- public String toString()

3) hashCode():- public int hashCode().

4) clone():- public Object clone()

5) wait():- public void wait() throws InterruptedException
:- public void wait(int time_in_ms) throws InterruptedException

6) notify():- public void notify()

7) notifyAll():- public void notifyAll()

7/2/23

* Java bytecode OR "Java is platform independant language"

• Previously, if we consider C/C++, then after successful compilation, it creates a translated machine level language file of our program. And it has ".obj" extension with same name as file name.

one.c / one.cpp → Prog (source file) → after successful compilation → one.obj (Translated file) (Machine lang. file) → after execution → one.exe (executable file)

*/

• But here in Java, after successful compilation, we get a seperate file of every user defined class and interface.

one.java

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

→ after successful compilation →

A.class, B.class, C.class, D.class → Bytecode are interpreted by J.V.M (Java Virtual Machine) → output

• From above diagram, note that after successful compilation we get an individual file of every user defined class & interface of that program. these files will have same name as class name or interface name and with extension ".class" these files are called as bytecode.

• Strictly remember that these bytecodes are not translated files, but they are encrypted files also remember that these bytecodes are interpreted by "J.V.M". These bytecodes are universally same and executes equally on every platform this is why bcoz of bytecode Java is platform independant language.

- In Java every constant is data member.
- Java Provides No's of Library constants. Every constant is public static Data member of specific class.

- After successful compilation java make a bytecode which is not translated code or Machine code
- It is just an encrypted code which is not readable by humans.
- This bytecode makes Java platform independent Language.
- ⋆ '.class' is the extension of this bytecode and its name is same as that of class this means every user-defined class in your program will generate separate bytecode files.
- ⋆ Interfaces will also generate bytecode with the same extension '.class'. So, it is not recommended to use the same name.

34/2/2023:

Java is Single rooted heirarchy OR Cosmic super class java.lang.Object

- Java provides a library class java.lang.Object, this class is the overall Base class of every Java class, whether it is a library or user defined class.
- This means, if we define a class, then automatically this class java.lang.Object gets inherited in it during compilation.
- Java provides this fexility to achieve a free run time polymorphism of any class and there are some important Member funct^n in class java.lang.Object which are available into every Java class.

- If we call a funct^n, and that funct^n is throwing any exception, then the calling funct^n must throw same exception.
- ⋆ "readLine() throws IOException" this is from java.io Package this funct^n is used to take input from user, scanner and other uses the same funct^n in their class.

4/2/23

Note:
- Every exception name is a class Name.
- Previously in C++ we applied access specifiers as tag but Here in Java we must apply individual Access specifiers to every Data member and member function separately.
- ⋆ While defining a function, we can use access specifiers and supporting Keywords in any order bat return type must be exactly before function name.
- ⋆ While declaring a variable, we can use access specifiers and supporting Keywords in any order but datatype must be exactly before variable name.
- In C/C++ default return type of a function is "int" whereas no default returntype in Java i.e every function must have returntype only constructor don't have returntype.

6/2/23:

Note:
- The main difference btw class and interface is
- ⋆ We can create objects of "class" only we can't create objects of "interfaces"
- ⋆ Ultimately any class or functions doesn't "throws" that particulare excepth^n class it throws Object of that class.

Some M.F which are in ⋆ java.lang.Object. class.

A-Z	65-90
a-z	97-122
0-9	48-57

{ ≡ } => { ≡ } : C-T error.

{ ≡ } => { ≡ } : C-T error.

3/2/23

* Java is a strongly typed language

- Previously, If we consider C/C++, then we can assign a value of one datatype to the variable of another datatype. this is beoz C & C++ are loosily datatype language

Eg:- `int a = 9.2;` → `a = 9` `float c = 11.5;` → `c = 11.5`

`Boolean b = 1;` → `b => true` `char d = 65;` → `d = 'A'`

- Every above statement is valid in C/C++, but it will show compile time error in Java beoz, Java is strongly typed language. this means in Java, we cannot assign value of one datatype to the variable of another datatype. Every above statement will show compile time error.
- Also remember that, In C/C++ 1 can represent as true & 0 can represent false but Not in JAVA.
- Also remember that, Java performs strict typed check for function parameter also i.e the datatype of actual parameter and formal parameters must match.

* • All relational operators check the relation on the bases of hashcode that means if two variables have same value then Java won't have two blocks in Hashtable for two values instead would create one block now that value & will point both the variables to the same block.

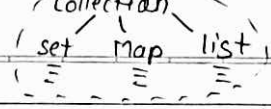
Eg:- variable → 5 ← variable

* Hash Code :- Hashtable is a data structure which is similar to array and in array we have index value starting from '0' similarly Hashtable have index values technically called as Hashcode.
- This is a 8-12 digit No. which is Randomised.
- In Array we can access values by their indices but Here we are not allowed to get values from Hashcode in Java for security reasons.
- Java creats a runtime environment which never lands on the actual machine architecture and hence it is difficult to hack.
- This J.R.E have a list of tools which includes J.V.M, hashtable etc.

* Default datatype for values having decimal point is "double" datatype and Therefore in whole Java wherever it was required they have used "double" datatype irrespective of their places such as parameter list, returntype or it may be a simple Data member.
* • So, it is highly recommended to use "double" datatype instead of "float" in our user defined classes.
* • In Java "java.awt.Color" class we have '7' constructors and among them '3' constructors have 'float' datatype in their parameter list.
- Rest whole Java won't have 'float' datatype used, its very rare.

datatype	wrapper class	parameterized constructor (To wrap)	static parse methods (To unwrap)	Non-static value Method (To unwrap)
byte	Byte	Byte(byte Val_to_wrap)	public static byte parseByte (String val)	public byte byteValue ()
short	Short	Short(short Val_to_wrap)	-11- -11- short parseShort (String val)	-11- short shortValue ()
int	Integer	Integer(int -11-)	-11- -11- int parseInt (-11-)	-11- int intValue ()
long	Long	Long(long -11-)	-11- -11- long parseLong (-11-)	-11- long longValue ()
float	Float	Float(float -11-)	-11- -11- float parseFloat (-11-)	-11- float floatValue ()
double	Double	Double(double -11-)	-11- -11- double parseDouble (-11-)	-11- double doubleValue ()
char X(Not available except)	Character	Character(char -11-)	-11- -11- char parseChar (-11-)	-11- char charValue ()
boolean	Boolean	Boolean(boolean -11-)	-11- -11- boolean parseBoolean (-11-)	-11- boolean booleanValue ()

* All the classes which are for data structure are called as Collections framework. There are total 19 classes & interface in this group.



Note • Previously in C/C++ we had Keywords "Signed" & "unsigned" to use with datatypes to specify range in +ve & -ve values but Here in Java no such Keyword. By default every value & datatype is "signed".

** • Previously in C/C++, the range of datatype was a cycle but not in Java.

* • Exception is a situation that occurs during runtime

11/2/23

* Wrapper classes:- These are to wrap values of primitive datatypes into an object.
• We have individuals wrapper class for every primitive datatype.
• All wrapper classes are in java.lang pkg.

Primitive D.T	wrapper class
byte	class java.lang.Byte
short	java.lang.Short
int	java.lang.Integer
long	—11—
float	—11—
double	—11—
char	java.lang.Character
boolean	—11—

• The initial versions of Java were not purely object oriented. In initial versions, every value was a primitive datatyped value and there were classes and interfaces of collections framework that always work on objects, therefore while working any collections framework, we cannot use any primitive datatype to store in collection framework. Therefore, we had to use respective wrapper class to wrap value of primitive datatype as an object and then store it into collections framework.

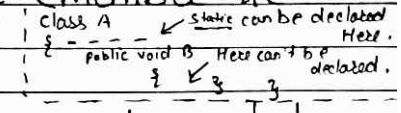
• Similarly, we have member functions in wrapper classes that unwraps an object into primitive datatype. Refer following table that contains some of important funct^n's & constructor of wrapper classes.

* • Now, it is not mandatory to use wrapper classes to use collections framework as now every value is itself an object.

12/2/23

Note * • Java provides many library constants (in Uppercase notation) every library constant is public static datamember of respective class.

• In Java local variables cannot be static only datamember can be static (member fn member can't be static).



• In Java we declare object as shown below:-

```
class_name obj_name = new class_name();
```

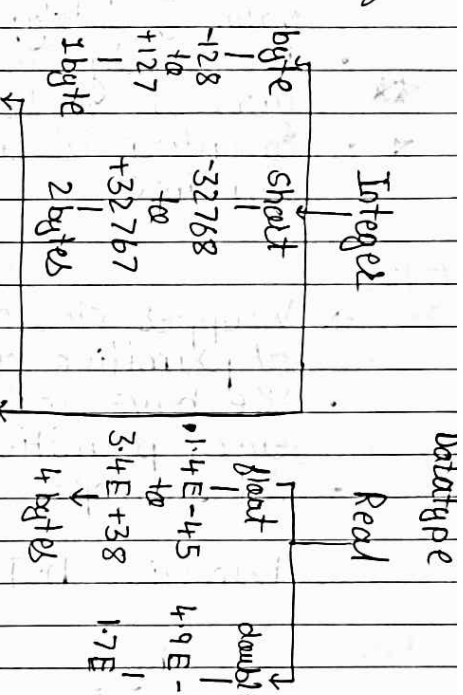
Reference to that object.. object

* • The datatype of any object is its classname or interface name. and therefore we say class & interface name are secondary Datatype.

• UTF = Unicode Transformation Format.
* • All Native methods in Java would create their own "headerfile" as C/C++ are machine dependant languages.

3/2/23
• Any consale value with decimal point is by default of type "double" to make it consider of type float we apply 'F' or 'f'.
• Any consale value without decimal point is by default of type "int" to make it consider as type long apply L or l
** • Java has 3 literal values → true, false & null

10/2/23
• Integer :- numeric value without decimal point
• Real :- numeric —11— with —11— —11—
• character :- stores unicode character
• Boolean :- stores boolean true and boolean false



* Keywords *
• are the words whose meaning are already known to compiler.
• Keywords are also called as "reserved words"
• Java → 49 Keywords but 51 Reserved words and goto, const, true, false & null are R.W.

* Primitive Datatypes :- This datatypes are inbuilt datatypes of any language.

* goto & const are reserve words in Java and not be used as identifiers.
• Java has a Keyword called "native" which allows us to write code in language other than Java and therefore "goto" & "const" are reserved to Not have any confusion btw those language and Java.
• Main objective of "native" Keyword is :-
i) To improve the performance of the system. as C/C++ are much faster.
ii) To overcome the memory management in Java
• Another reason for making them as reserved words could be bcoz some Tools of Java are prepared usig C/C++.

- String value is itself an Object
- variables → also called mutable
  Constant → -11- -11- immutable.

Amazing · Whenever you'll print an object By default toString() method gets executed, its return type is String so, you must return a String otherwise compile time error. This method is a M·F in cosmic Superclass Object.
· You can override this toString() method in your userdefined class as per your needs.

17/2/23

* toString() Method *

· Java provides a simple facility to print an object remember that when we print an object or we use an object in expression, then toString() Method executes and that object behaves as invoking object.
· To avail this facility for our userdefined class, we have to override toString() method in our class. We must override it as

"public String toString()"

· This method must return a string and that will get considered wherever Object is used.

18/2/23 In Java Strings are immutable. This means every string manipulation related function returns a new String object. It will never override/or update existing string object.

- String Buffer class is mutable.

Library → Package → classes & Interfaces & Functions

* Java A·P·I (Application Programming Interface) or Library packages.

· The entire Java has total 7 libraries. They are
⇒ java, javax, sun, sunw, com, org, launcher.
among these libraries, "java" is the core library that contains different packages related to standard client side development.

Note · Based on the purpose of classes and interfaces, Java is divided into three platforms of Java
i) Java S·E → Standard Edition
ii) Java EE → Enterprise Edition
iii) Java ME → Micro Edition.

— x —— x END OF NOTE x —— x ——

· The java library provides various packages & nested packages following are some of commonly used packages of java library
· java.io, java.util, java.lang, java.awt, java.sql java.applet, java.net

1) java.io :- These package contains set classes & interfaces related to input output operations following are some of commonly used class and interfaces of this package.
· class BufferedReader
· class DataInputStream
· class FileReader
· class InputStreamReader

20/2/23

2) java.util :- These package contains classes & interfaces related to additional utility support following are some of commonly used classes & interfaces of this package.

- True, false & null are literals.
- reference doesn't have any memory.
- Formal parameter can never have an object
- Java doesn't support functions with default arguments.

Pojo
JavaBeans

- It is always a good practice to write code in separate class & 'PSVM' in Separate class.
- Anything concatinated with String is ultimately String
- To convert into String read last point on the Pg.

- In Java, every object is anonymous (no name) and the reference is the name given to object.
- In many books, they say "reference stores hashcode of object".
- other people say it is a 3 way hashing system (3 way referal System).
- Any Highlevel language by default passes reference i.e pass by reference works in any function. All primitive D.T would pass by value, call by value but Secondary would class call by reference by default..
- You can never pass an object, it just looks like that you are passing it looking at the code but it never leaves it place.
- We can put this like the hashcode of that object is getting passed rather than object itself.
- Whenever there would be 'new' Keyword used then another object would be created for sure.
- In Java you can initialize D.M but not in C++

15/2/23

- There are two types of classes in Java
i) Pojo (Plain old Java object):- Normal classes that we create in our program.
ii) JavaBeans:- Classes containing setter() & getter() methods are called JavaBeans class and in this type of classes also include constructors.
- As Java strongly supports XML i.e Extensible MarkUp Language, and we are allowed to call constructor, setter & getter Methods from any XML tag and Hence we make seperate classes combining all of this Methods only called as JavaBeans.
- As we have seen wrapper classes, 'int' & 'Integer' both are different. 'int' is primitive datatype and also a Keyword, on the other hand 'Integer' is class name which is a secondary datatype.
- As we Know, In Java by default every value is an object. So,
int a = 10; & Integer a = 10;
variable reference
- Both are valid and would print the value of 'a' as '10' but only difference is that primitive datatypes don't as are not able to invoke an method. it is just a simple variable. You can call methods by ① statement.

Imp
- To convert primitive datatypes into String just concatinate that with empty quotations.
Eg:- int a = 10; String b = "" + a; // valid.
- For Secondary datatypes. You have the references/object you can invoke "toString()" method from it..
Eg: Integer a = 10; String b = a.toString;
- Whenever you want to pass/call by value use primitive and for call by reference you can use secondary datatypes.

- B.C ref = D.C object

Date / Page

* - Right hand side is majority times an object
- The major difference btw Java & python is that Java has Datatype to a reference & python has references without datatypes.
* - Always when we see class name in parameter list then immediate next to it is a reference to that class & not an object.

Eg:- public void addElement(Object ref); → this is ref to Object class

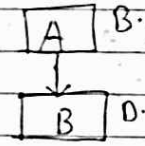
- We can declare an object in 2 steps also

i) class_name obj_name;
≡
ii) obj_name = new class_name();

- The most important line throughout Java is

**** Base class reference = Derived class object *

Eg:-



A a1 = new A(); ✓
B b1 = new B(); ✓
A a1 = new B(); * ✓
∴ we can say a1 = b1 *

- We Know, Java has a Cosmic superclass Object which is overall Base class of every class in Java.
- We can pass our object of any class that has class Object reference in their parameter list and Hence, it enables us the fecility of runtime polymorphism.
- If Not have Object class reference but reference of another class is given then make that class your B.C by extending it into your user defined class. It doesn't matter just B.C ref = D.C obj this relatn should satisfy.

14/2/23

Note :- Constructor of wrapper class is to wrap the primitive datatyped value as object.

- Parse methods are to convert/Parse parameter string value into primitive datatype value.
- Value methods are to unwrap the object & return primitive datatyped value of it.

* - In Java every static & Non-static datamembers are by default gets initialized with their default zero's. Refer full^n table.

Datatype	default zero
byte, short, int, long	0
float, double	0.0
char	'\0'
boolean	false.
Secondary datatypes	null

* - Local variables, formal parameters and block variables does not get initialized with default values. We must assign a value before using them otherwise compile time error.

- In Java Every library constant is public static D.M of respective class.
- Primitive to obj → use constructor
- object to primitive → use value methods.

Date / Page

• readUTF()

25/2/23

Note • We know, that Java has many heirarchies of classes and interfaces. The root class of every heirarchy is either an abstract class or interface.

* • If a function returns an object, then we can use its base class name as returntype. but while receiving, we must typecast into derived class.

• Previously in C/C++, a function cannot return an array but here in Java a function can return an array

Eg:-
```
int[] checkR()
{   int a[] = new int[5];
    =
    =
    return a;
}
```

** • While passing an array as actual parameter and while returning an array, do not use indexing operator ([])

•
```
System.out.println()
```
library class in java.lang → System.out
non-static m.f of class PrintWriter

a library reference of PrintStream declared as static d.m of class System

* • a class cannot be private & protected same for interface. Hence, public & default (friendly) only both are allowed.

27/2/23

7) Secure and Robust:- Java is secure bcoz concep of pointers is removed if we consider pointer, then there are many problems in pointers. Point causes data lose, System crash, virus attack. Java has removed pointers therefore, Java applicat^n are now secure and no chances o system crash.

• Java is Robust which indicates layered security & accessibility to entities We can assign many access specifiers & supporting Keywords to classes and members of classes & interface. In other words we can say Java has solution for almost every accessibility related issues

8) Reliable:- From every above features & previou discussions, we say Java has numbers of background tools to support various technolog , Java is secure & robust, Support XML & database, etc.

• In case of applicat^n development, Java allows GUI development, console application, network based application development, etc.

• Java is portable, dynamic and loaded with many dynamic concepts to match any current technologies and framework Hence we say Java is reliable and one of best programming language to development of almost every typ of application.

In C++, "cin" & "cout" were streams and we also know that the classes which produces the streams are called as stream classes.

- Previously in C++, there were three types of stream classes -

i). character stream classes
ii). template —11—
iii). Wild character —11—

- Here in Java, we also use stream for input output operat^n. Here in Java there are two types of stream classes

i) character Stream classes
ii) Byte stream classes.

Imp ★

- The character stream classes performs input output operations based on characters any library class that ends with word "Reader" & "Writer" are among character stream classes.
- Every Reader class performs character Based input.
- Every writer class performs character Based output.
- There are two Heirarchy classes java.io.Reader & java.io.Writer, these two are Base classes of every Reader, Writer class

class Reader	class Writer
↓	↓
class BufferedReader	class BufferedWriter
class FileReader	class FileWriter
class InputStreamReader	class PrintWriter
class PipedReader	class PipedWriter

The byte stream classes performs I/O operat^n Based on bytes. Generally networked based communication is performed through byte stream classes.

- Any library class that end's with word InputStream & OutputStream are among byte stream class.
- Every inputstream class performs byte based Input.
- Every outputstream class performs byte based output.
- There are two library classes java.io.InputStream & java.io.OutputStream. These two classes are base classes of every input and output stream class.

class InputStream	class OutputStream
↓	
class DataInputStream	class DataOutputStream
class FileInputStream	class FileOutputStream
class PipedInputStream	

* Dependancy injection.

- class Vector
- class Scanner
- class Calender
- class DateTime
- All collections framework related classes & interfaces
- interface Map
- interface Set

3) java.lang :- These package contains the most commonly used classes & interfaces, for general purpose development. Following are some of commonly used classes & interfaces of these package.

- class Object
- class System
- class Thread
- class Throwable
- class Exception
- ~~class~~ All wrapper classes.
- interface Runnable
- interface cloneable
- interface Numeric

Note * "java.lang" package is by default imported in every Java program.

4) java.awt (Abstract window toolkit) :- classes related to G.U.I :- These package contains package classes & interfaces related to G.U.I development mostly client side development.

- class Button
- class Checkbox
- class Font
- class Color
- class TextField
- interface LayoutManager

- Streams are objects that are used to perform I/O operations.
- Public final static PW out = new PW(---);

Character Template
whilld - Charact
3 type of strea in C+

5) java.sql :- These package contains classes & Interfac related to database connectivity & firing S.q.l queries. Following are some of commonly used classes & interfaces of these package.

- class DriverManager
- class SQLException
- interface PreparedStatement
- interface Connection
- interface Statement
- interface ResultSet

6) java.applet :- These package contains classes & Interfaces related to developing simple client side webpages. Following are some of common used classes & interfaces.

- class Applet
- interface AppletContext
- interface AppletStub
- interface AudioClip.

7) java.net :- These package contains classes & interfac related to developing networking based application (client - server application). Following are some of the commonly used classes & interfaces

- class Socket
- class ServerSocket
- class UnKnownHostException
- Class InetAddress.

* Streams in Java :- Previously in C++, we had already used streams for input output operat
- We know, that the objects that can perform input output operations are called as stream

a more powerful framework Hibernate that performs O.R.M (Object Relational Mapping).

7/3/23

* Input/output statements *

We know that, input/output statements are to communicate with end users (also with input/output resources).

- Java allows input/output operations in form of bytes & characters.

1) Output statement: To perform general purpose output operation with console we have 3 + 1 library functions:-
- println()
- print()
- printf()

All are non-static

- The println() function is by default print with "\n" it automatically appends your message with "\n".
- The print() function print's specified message but in same line (like printf).
- We can manually add "\n" in messages string.
- The printf() function is totally similar to printf() function of 'C' which allows format specifiers in message.
- All this methods are in two different classes class java.io.PrintWriter, class java.io.PrintStream.

This means to use above functions we have to use objects of respective class but as advantage Java provides a preconfigured (built in object) of class java.io.PrintStream named as "out", which is public static Data Member of class java.lang.System. Therefore, instead of declaring our user-defined object and using printing function, we prefere to use preconfigured object "out" as follows.

```
System.out.println("—");
System.out.print("—");
System.out.printf("—");
```

System → library class in java.lang

out → library object of class java.io.PrintStream which is static d.m. of class java.lang.System

printf → Non-static m.f of class java.io.PrintStream

- Actually print() & println() are overloaded functions which has following overloaded signatures

```
public void println() ← To feed do new-line only
public void println(byte val)
public void println(short val)
 —11—   —11—  (int val)
              (long val)
              (float val)
              (double val)
              (char val)
              (boolean val)
```

9) Portable :- The portability of Java can be justified by simple concept of sharing program source code & bytecode.
- If multiple developers are developing a distributed application, and all are having different system configuration & development tools, then also their source code can be merged over another platform without single change in it.
- Even if bytecode are shared then also it can be used on any other platform with any technology / framework.

28/2/23

10) Dynamic :- Java is dynamic lang. becoz by using Java, we can prepare almost every type of application starting from basic console applicat^n upto n-tier enterprise web applicat^n
- We can also prepare client side GUI applications, network applications (using socket programming), basic web applications (by using applet plus awt).
- We can also prepare mobile applications / android applicat^n (by using Java M.E libraries) etc.
- Many features & facilities of Java also makes it suitable to create new frameworks & technologies and dynamic libraries / modules

Java.net.InetAddress

* • System.out is Preconfigured objects towards console o/p.
• JSP & J.S.F (Java Server Faces)

11) Distributed :- Java is a distributed language becoz of it's ability to create distributed applications by using network libraries.
- We can create distributed applications to establish communication btw multiple clients and also to share files data over the network.

12) XML Support :- Java strongly supports XML & that's why we are able to create any web related application by using Java. We can create applets, servelets, JSP (Java Server Pages) by using Java & XML combination.
- Even we can also use Java classes from an XML file by using J.S.P action tags (J.S.P:---)
There are such 17 action tags
We can also declare objects of Java class & call to setter() getter() methods from an XML file.

13) JDBC (Java Data Base Connectivity) :-
Java provides pakage Java.Sql & Javax.Sql These pakages contains classes & interfaces related to database connectivity and firing Sql queries.
- java.sql pakage contains classes & interfaces related to general purpose sql operations for DDL & DML queries, whereas javax.sql pakage contains classes & interfaces related to meta data of databases.

Note • JDBC is a technology, that connects Java Programs with database but currently we have

public void println(char arr[])
—11— —11— (String val)
public —11— (Object obj)

2/3/23

- Similarly the print() function is also an overloaded function & has following signatures:
  public void
  System.out.print(byte var);
  System public void print(short var);
  public void print(int var);
  —11— —11— (long var);
  —11— (float var);
  —11— (double var);
  —11— (char var);
  —11— (boolean var);
  —11— (char arr[]);
  —11— (String val);
  —11— (Object obj);

- The main diff" btw print() & println() function is "new line"(\n)
- To print any string message with variable values, we have to concatenate string message by using String concatination operator (+) remember that if we concatinate string with any variable, then resulting D.T is String.
- ∴ In printing also we have to use string concatenation "+" to display values of variable

- If we use any short-cut calling statements, then return type of inner function & parameter of outer function must be same.
  "\"
  S.o.p(c-x); (Parameter, return type — Both same.)

for example

- S.o.p("—11—"+a);
- S.o.p(a+"—"+b+"—11—");
- S.o.p("Addition is"+(a+b)); → Must do this.
- From JDK 1.5 onwards we are also allowed to use "printf()" similar to the printf() we used in 'C'. We can use format specifiers & formatting characters in the same way as we used in 'C', for example.

⇒ public PrintWriter printf(Locale l, String arg, Object ... obj);

⇒ public PrintWriter printf(String arg, Object...o

* Input Statement:-

- Originally we have readLine() method as input statement. This method is also present in byteStream & character stream classes for various purposeful input. This method is originally defined in class "java.io.Reader" java.io.InputStream. Both this class are abstract classes therefore, we cannot declare their object. Therefore we declare objects of their derived classes & call to readLine() method by using objects of derived class.

- If any M.F is not working on its D.M then make that M.F as "static". If only working on it Parameter.

Scanner sc = new Scanner(System.in);
all" are the member functions are M.F of class scanner to receive directly passed input.

- public byte nextByte()
=> reads input from console & returns as 'byte' type value.

- public int nextInt()
=> reads input from console & returns as 'int' type value.

- public short nextShort()
=> reads input from console & returns as 'short' type value.

- public long nextLong()
=> reads input from console & returns as 'long' type value.

- public float nextFloat()
=> —"— —"— —"—
'float' type value.

- public String next()
=> Reads one-word string and returns as "String" type value.

- public String nextLine()

- import static java.lang.math.*;
- The package "java.lang" is by default imported but their nested packages are not default imported.

=> Reads multiple-words string and returns as "string" type value, as gets() in C lang.

Note that above listed classes Scanner's M.F does not throw IOException But if any invalid input is provided then these functions will throw java.util.InputMismatch Exception during runtime.

for ex:-

```
Scanner sc = new Scanner(System.in);

int age;                    char gen;
age = sc.nextInt();         gen = sc.nextChar();
                            gen = sc.next.charAt(
float age;
avg = sc.nextFloat();

String sname;
sname = sc.next(); // for single word
sname = sc.nextLine(); // multiple words.
```

6/3/23

* Creating & Saving Java Program:-
- For compiling we have tools provided in JDK.
i) For compiling:- javac file_name.java ↵
ii) To run:- java Main bytecode_name.java ↵
Here, to run the code ".java" is optional but for compiling it is must.
- In this, we need to remember 2 names

- Blank "dis.readLine();" to flush() as we do in 'C'.

Boolean → %b, Integer → %d, Real → %f,

* Importing Packages:- Previously in c/c++ we include header files by using preprocessor directive #include<> similar to that, now we use Keyword "import" to import/include any bytecode.
- Actually we do not include packages but we include/import bytecode.
- To import any bytecode general Syntax is

```
import bytecode name;
        or
import class_name;
        or
import lib_name.pkg_name.bytecode;
```

- Actually by importing bytecodes we are actually importing files.

for ex:-

```
import java.util.Vector;
import java.awt.Button;
import java.awt.event.ActionListener;
import javax.swing.ImageIcon;
```

Note. In case if we need multiple bytecodes from single package then we can also use pattern matching character '*' to replace every files / bytecode

for ex:-

```
import java.io.*;
(will import every bytecodes from java.io package)
```

```
import java.awt.*;
(will import every class in 'awt' but not in nested package)

import java.awt.event.*;
(now will import all classes of event package only)
```

4/3/23 - Remember that this '*' will only replace bytecode/files. It will not replace packages & libraries therefore following statements are not valid

```
import java.*.*;
import *.*.*;
```

** Note. Actually even it is optional to import any bytecode. but now every time, we have to use a bytecode with complete path.

* Class "java.util.Scanner":- initial JDK version uses readLine() method & read() to receive input from console. But JDK 1.5 onwards we have another class "java.util.Scanner". This class can also receive input from console & similarly works with preconfigured obj "System.in" therefore constructor of this class also requires preconfigured object "System.in" as parameter. for example.

DIS OR I·S·R + B·R

- We can put it like "System.in" is what we are looking for. So, if this is the Parameter then we can use it for taking input.

class

Mostly we prefer java.io.DataInputStream & class java.io.InputStreamReader plus java.io.BufferedReader.

- This Re readLine() method is defined as

public String readLine() throws IOException*

From above signature, Note that the readLine() method reads input & returns the input as "String". Therefore, we need to parse (convert) the string input into required primitive datatype by using parsing methods of Wrapper classes.

Exc:-
```
DataInputStream dis = new DataInputStream(System.in);
int age;
String s = dis.readLine();
age = Integer.parseInt(s);
```
OR
```
age = Integer.parseInt(dis.readLine());
```

=>
```
InputStreamReader isr = new InputStreamReader();
BufferedReader br = new BufferedReader(isR);
float avg;
String s = br.readLine();
avg = Float.parseFloat(s);
```
OR
```
avg = Float.parseFloat(br.readLine());
```

* To read character we have read() originally.
- gen = dis.read().charAt(0);
-> public char charAt(int index_val); — method in String class

3/3/23

* Accepting a character input:- Previously readLine() reads a string from console & returns this function can input single character but that character will be as string.
- To accept single character input, we have another special method read(). This method is defined as

public int read() throws IOException

* This function reads character & returns its ASCII value. Therefore, while receiving, we must typecast into character.

for exc:-
```
char gen;
DIS dis = new DIS(System.in);
gen = (char) dis.read(); // Typecast is Must.
```
- The read method can also be called by using combination of I·S·R + B·R

System.in

* Accepting String Input:- The readLine() mthd originally reads & returns string value. Therefore to accept any string value we can directly use readLine() without any typecast or Parsing.

Eg:-
```
String sname;
sname = dis.readLine();
```
OR
```
sname = br.readLine();
```

** • Reference can never have size & therefore in Java, int ar[5]; is invalid & will give compile time error.

```
for(i=0; i<arr.length; i++)
{
  Sop1n("At"+i+"argument received"+arr[i]);
}
}
}
```

- Arrays in Java are objects that are dynamically allocated on the heap memory.
- Hence, in Java we have a different way of declaring it

I) → `int ar[] = new int[5];`

Reference — actual object/array in this case.

2) → `int[] ar = new int[5];`

- Above only 2 ways are there of creating an array & both are valid & can be done in 2 steps rather than doing it in Single Line.
- We use "int[]" when we want to create multiple arrays & to write less for exc:-
  `int[] ar1, ar2, ar3;`
- Now when ever you need you can assign them size as per your needs.

* • Every array in Java is by default an object of built-in "Array" class. and thereby we can use array name to invoke/call Data Member & Member function of "Array" class

* • In Java Access specifiers are not given ~~with~~ to use in same program but they are specified or need to be applied keeping package into consideration.

** • "length" is a public data Member of Array class and hence we are able to use it with any array name & dot operator as array name is itself an object.
- When we declare array then the size which we have mentioned gets assigned to that "length" data Member

Eg:- `int ar[] = new int[7];` → this 7 gets assigned to length.

- It is believed that when we do "java main-byte-code-name" we are actually passing command Line arguments because in C/C++ also we do have this facility of C.L.Arg.

* • A good programmer always create's a single class in one program & then bundles all these single class from different programes under same package.

10/3/23

⇒
```
class Test
{
  p.s.v.m(String args[])
  {
    int n1, n2, n3;
    n1 = Integer.parseInt(args[0]);
    n2 = ——— 11 ——— args[1]);
    n3 = ——— 11 ——— [2]);
    int S = n1+n2+n3;
```

to java program i.e we can give any name to java program

- To compile a java program we use jdk tool "javac" & to run Java program we use "java" tool.

* Therefore,
  To compile → javac File_name.java ↵
  To run → java main_bytecode_name ↵

9/3/23

Note
- Previously in C/C++ we cannot overload main(). but here in Java, we can overload main() the main() which has String[] as parameter, will only begin execution. other main() methods will be like concrete methods & can be manually called if required.
- In Java we can define numbers of main() mthds in individual classes but while executing the bytecode name that we specify will begin the execution.

* Command-Line arguments:- It is the facility to provide input to java programs even before program execution starts we do this to provide external inputs & that does not use & any offcial streams. We provide comnd line argu. while executing java programs through command lines./command prompt. We do it as.

java main_bytecode_name arg1 arg2 --- ↵

- Every array in Java is object of "Array" class.
- Length is a public D.M of Array class
- ∴ we do arr.length; ↵
- int ar[] = new int[5]; this '5' goes to length

- These command line arguments must be seperated by space & every provided command Line argument will be initialized to String[] present as parameter of main()

```
public static void main(String ar[])
{
   ≡                          ↑ To receive command line argument.
}
```

- Every provided command line arguments will get initialized as String values, for exc:-

```
class Test
{
  public static void main(String ar[])
  {
    ≡
  }
}
```

	ar[]
ar[0]	25
ar[1]	67.93
ar[2]	XYZ
ar[3]	Hello there

To compile:- javac Test.java
To run:- java Test 25 67.93 XYZ "Hello there"

⇒
```
class Test
{
  public static void main(String ar[])
  { int i;
    S.o.pln("Provided arguments are\n");
```

one of filename & other of the bytecode containing "main" method.

- So, people started to give same name at both the places for ex:- file name is "A.java" & the class containing main funct" is also have class name A. Therefore it becomes easier.

** - We Know that "static" entities don't need object to invoke them. They can be called by using classname & dot operator (.).

*** - But if you have to use many static entities at different places then we can have a shortcut to do so. Simply import that package as "static import".

for ex:-

x Static import java.lang.Math.*; // Not valid

✓ import static java.lang.Math.*; // Only valid way

- "*" is also important to do static import.
- With this you can use static D.M / M.F without classname & Dot operator.

** - If original method of base class is not throwing any exception then the overriding method of derived class must not throw any exception.

** - If original method of base class is throwing any exception then it is optional for overriding method of derived class to throw the same (or another) exception.

8/3/23

- We Know that, Java is a structured programming lang. & therefore we have to define main() to begin the execution of program.
- And we also Know that, Java does not allow open functions i.e every function must be a member function & therefore main() must also be a member function of a class.
- Therefore, a general & Relaton to define main() method will be always

```
class Name
{
    public static void main(String ar[])
    {
        =
    }
}
```

- String args[]

- An important point is, above signature of main() must not change if we do any change in above signature of main method, then it will considered as a concrete normal main() & it will not begin execution.

Note - In above signature, we can apply 3 additional Keywords:-

- synchronized
- final
- strictfp.

- A Java program must be saved with extension "java" & their is no rule in giving file name

For ex:-

```
class Test
{
    static
    {
        =
        = }
    +
    other M.F
}
```

- Note that static block is not a function therefore no parenthesis () & no access specifiers to it. Therefore no overloading possibilities of static block.
- If we define static block in a class, then it executes automatically even before execution of ~~on~~ first function of any class.
- Unlike constructor, a static block will not execute for every object, but it will execute only once in entire application/ program.

Refer foll^n program:

```
class Test
{
    static
    {
        S.o.pln("static block executes");
    }
    Test()
    {
        S.o.pln("constructor executes");
    }
}

Class A
{
    P.S.V.m(String args[])
    {
        S.o.pln("main starts");
        Test t1 = new Test();
        Test t2 = new Test();
        S.o.pln("main ends");
    }
}
```

O/P

```
main start
~~const~~ Static
const
const.
```

- From above o/p note that even before the execution of constructor the static block is been executed.
- If logically required then we can also declare static block in main class & then this static block will execute even before the main function.
- The main purpose of static block is specially to initialize static data member

- We can apply access specifiers to classes & interfaces but a class or interface cannot be private & protected i.e class & interface can be only public or friendly.
- Only one class or interface of a program can be public & that program must be saved with that public name".

⇒
```
class Test
{
is public → public void method1()
            { --- }

is protected → private void method2()
               { --- }

is
Protected → protected void method3()
            { --- }

is
default → void method4()
          { --- }
}
```

13/3/23

* Constructor: We Know that, constructor is a member function with 3 characteristics
  i) Must have same name as class name.
  ii) Cannot have return type therefore cannot return
  iii) Cannot be called manually, automatically executes when objects of class is declared.

** • We should always use static block to initialize static data members to prevent overwriting of it. as they execute once in whole program.

- Previously in C++ we had 3 types of constructor
  1) default constructor (Non-parameterized constructor)
  2) Parameterized Constructor
  3) Copy Constructor.
- Here, in Java we are allowed to define '2' types of constructor
  i) default constructor (No arguments)
  ii) parameterized constructor.

Note:- In Java, their is no copy constructor but logically we can override "clone()" to implement it as copy constructor.

- We Know that, In case of inheritance when object of derived class is declared first of all constructor of Base class executes & then constructor of derived class executes.

Imp

* Static Block *

- Java provides an additional facility to specially initialize static data Members of a class & that is by defining static Block.
- A static block is defined inside a class, but it is not any Kind of member of a class.

```
        S.o.pln("Addition is "+s);
    }
}
```

To compile: javac Test.java ↵
To run: java Test 10 20 30 ↵

- To make this dynamic we can have a for `for(int i=0; i<args.length; i++){}`

10/3/23

- In Java there are 3 access specifiers which are themselves Keywords & one access specifiers named friendly is not a Keyword but it is the default access specifiers i.e if not specified then it is friendly / default access specified, In total we count 4 access specifiers.

* Access specifiers :- We know that, we can apply access specifiers to members of classes & interfaces. Here in Java we have 4 access specifiers they are:-

- private
- public
- protected
- friendly

- private members are not accessible outside the class. (or accessible within that class)
- protected members
- public members are accessible everywhere in current prog/pakage, and also into another prog/pakage when imported.
- friendly (default) :- These are accessible everywhere in current prog/pakage, but not into another prog/pakage when imported.

Note
- Java provides '3' Keywords - "private", "public" & "protected" but "friendly" is not available as Keyword. Therefore to make any member as friendly, simply donot apply any access specifier.
- We have Keyword "default" but it is valid only in Switch, case & in interfaces.
- In Java, we have to apply individual access specifiers to every datamember & Member function.

* - To have best experience about above access specifiers, use them along them pakage declaration.

- Java also allows to apply access specifiers to classes & interfaces. But two have best experience use it with pakage declaration.

- Any indexOf() returns '-1' if that value is not present.
- In file handling also EOF returns '-1'.

16/3/23

- This class provides foll^{n} overloaded constructors

  - Vector()
  - Vector(int capacity)
  - Vector(int capacity, int increment)

* - First constructor creates Vector obj, with default settings, by default capacity of vector is 10. for ex

  Vector vc = new Vector();

- Second constructor creates vector obj with specified capacity for example

  Vector v2 = new Vector(20);

- Third constructor creates Vector obj with specified initial capacity and increments with increment value provided

  Vector v3 = new Vector(20, 2);

Note - Vector is also one of collections framework class. Therefore, there is no advantage to declare such collections framework as local variable of any function. Therefore we prefer it to declare as public static data member of a class so that it can be used as a global variable of program

- We know that vector is a dynamic memory that is it expands but whenever vector expands it will not expand by one block but it will expand according to its current capacity & directly becomes double.

- This class provides foll^{n} commonly used Member functions.

  - addElement()
  - addElementAt()
  - add()
  - removeElement()
  - removeElementAt()
  - removeAllElements()
  - isEmpty()
  - capacity()
  - size()
  - contains()
  - firstElement()
  - lastElement()
  - indexOf()
  - elementAt()

1) addElement():- This method adds specified element into invoking vector object this method is defined as

  public void addElement(Object obj);

15/3/23

```
=> class Student
   {
      public static final int sfees = 3500;
      public static String btime;

      Static
      {
        DIS * dis = new DIS(system.in)
        System.out.println("Enter batch time");
        btime = dis.readline();
      }
        + other m.f
   }
```

- From above example note that we can also write normal executable statements and input statements in Static block.
- In above static block, we have intentionally used readLine() to specify that we cannot manually throw any exception from static block. Therefore, we are forced to use Try{}, catch{} block only in static block.

* java.util.Vector *

- Java provides a library class java.util.Vector which is one of collections framework

** • Even in file handling when we reach at EOF it will return '-1'

** • Size means currently present no. of elements

* • Capacity means can store how many elements i.e Max size

class which we can use as a dynamic / expandable array

- The Collections framework mainly used as global variables because you can consider them as your pipeline which is common to entire program. If you feel some entities will be needing in another function then simply add that entity in the Vector and whenever required take it from Vector and use it.
- If any value is not present in Vector and you do indexOf() then this function will return '-1' which means that value is not present at any index or even in whole vector.
- Vector can have size 'zero' while we declare it's object and while adding the elements it would for sure increase its size/capacity by 1 if not mentioned but from next time it would increase by the double value of previous.

for ex:- current capacity = 0 then 1 then 2 then 4

so that it never gets reset because of constructor.

For ex:-

```
class Student
{
    private int sage;
    private String sname;
Static private int sees;
Static private String btime;

    Static
    {
       sees = 3500;
       btime = 8:00 AM;
    }

    public Student (int a, String n)
    {
       Sage = a;
       Sname = n;
    }

    + other M.F
}
```

From above class note that the static block is specially used to initialise static data members.

- anonymous block will always execute before constructor. If constructor will not execute then anonymous block will also not execute.
- anonymous block is for dynamic class.

14/3/23

	Static Block	Constructor
•	It is Block	• Const^n is a M.F of a class
•	Cannot be overloaded	• Const^n can be overloaded
•	We can access only Static members from S.B	• Const^n can access static/ Non-Static
•	Static block executes only once before execution of 1st program in whole function program.	• Const^n executes for every object.

- Static Block :- Executes first, even before execution of any static or non-static method of that class.
- Constructor :- Executes for every declared object.
- anonymous block :- behaves as constructor of anonymous class.
- Execution order :
  Static block
  anonymous block
  Constructor

** -> String s2 = "Junit".toUpperCase(); // is valid

18/3/23

* "java.lang.String" or class String

- First thing is, in Java, String is a library class name. Therefore it is a secondary datatype (Non-primitive datatype).
- A String is set of characters, with dynamic length. In Java string is an "immutable entity" i.e it does not allow any change/overwriting in itself we always get new resulting string object.
- In Java String is not char[] and char[] is not String.
- In Java string and character array does not end with '\0'.
- In Java every string value is an object of class string. Therefore we can directly use a string value as "invoking object"
- For different logical operations we have following overloaded constructors of this class They are.
- This class is final class i.e it cannot be derived or inherited.
- If any invalid index no. is provided to array, then it throws ArrayIndexOutOfBoundsException exception. If any invalid index no. is provided to string, then it throws StringIndexOutOfBoundsException exception.

AIOOBE at SIOOBE.

-> String ()
String (String val/ref/obj)
String (String val/ref/obj, int start-index, int end)
String (char array [])
String (byte array [])
String (char array [], int st-index, int end-index)
String (byte array [], —11— —11—)
String (StringBuffer obj)

20/3/23

1) String (): - Creates string object with default string value null.

- String s1 = new String();

2) 2nd constructor creates String object with specify and it initialise it with specified string.

for eg:- String s1 = "There was a King";
String s2 = new String (s1);

3) 3rd constructor creates a String object and initialize it with the sub-string specified from "start index" to "end index - 1" for eg:-

String s1 = "There was a King".
String s3 = new String (s1, 3, 12);
s3 => "re was a K"
3 12

This method is defined as:-

```
public int capacity()*
```

9) ~~Si~~ size():- This method will return the count of currently present elements in invoking vector object. This method is defined as

```
public int size()*
```

10) contains():- This method will search the specified object in invoking vector object. This method is defined as

```
public boolean contains(Object obj_to_search)
```

11) firstElement():- This method will return current first element in invoking vector object. This method is defined as

```
public Object firstElement()
```

- Return Type of this function cosmic super class Object, therefore while receiving any object through this method, we must typecast into required class.


```
A a = (A) v1.firstElement();
```

12) lastElement():- This method will return current last element from invoking vector object. This method is defined as

```
public Object lastElement();
```

13) indexOf():- This method obtains and returns index no. of specified object in invoking vector object. This method is defined as

```
public int indexOf(Object obj_to_check)
```

*# If specified object is not present in invoking vector object then returns '-1'

14) elementAt():- This method obtains & returns object present at specified index. This method is defined as

```
public Object elementAt(int index)
```

- If specified index is not present/valid then this method will throw ArrayIndexOutOfBoundsException

- If invoking vector object is already full and we use this method to add new element, then this functions expands invoking vector object.

17/3/23

2) addElementAt():- These method adds specified object at specified index value

- Signature:-

```
public void addElementAt(Object obj_to_add
                         , int index);
```

3) add():- These method will add specified object into invoking vector object. this mthd is defined as

```
public void add(Object obj_to_add);
```
*

- If vector is already full and we use add() method, then this functions will not expand invoking vector object

4) removeElement():- These method removes specified object from invoking vector object this method is defined as:-

```
public void removeElement(Object obj_to_remov
                          );
```

- If specified object is not present then operati fails & these method will not have any effect.

5) removeElementAt():- This method removes the object at specified index in invoking vector object. This method is defined as

```
public void removeElementAt(int index);
```
*

6) removeAllElements():- This method will remove all elements ~~in~~ present in invoking vector object. Now, size is zero but capacity will not be zero. This method is defined as

```
public void removeAllElements();
```
*

7) isEmpty():- This method checks & return whether invoking vector object is empty or not i.e checks whether size of vector is zero or not.

```
public boolean isEmpty()
```

8) capacity():- This method return the maximum size of invoking vector objec

public boolean equals (Object obj)

- public String toLowerCase()

Eg:- String s1 = "HELLO";
-11- s2 = s1.toLowerCase();
s2 => "hello"

9) equals(): - This funct^n compares invoking string object with specified parameter string. This method considers case sensitivity.

parameter type is Object

- public boolean equals (String to_check);

Eg:- String s1 = "Hello ALL";
s2 = "James";
s3 = "JAMES";

boolean b1 = s2.equal("James"); // false
b2 = s3.equal("James"); // false
b3 = s3.equal("JAMES"); // True

10) equalsIgnoreCase(): - This funct^n compares invoking string object with specified parameter string without considering case sensitivity.

- public boolean equalsIgnoreCase(String s)
- Now above all ex. are correct.

11) endsWith(): - This funct^n ~~ends~~ checks wether invoking string object ends with specified string value or not. Case sensitive

- public boolean endsWith(String to_check)

Eg:- String s1 = "There was a Ring".
boolean b1 = s1.endsWith("ing"); // true.
b2 = s2.endsWith("."); // false.

12) startsWith(): - This function checks whether invoking string object starts with specified string value or not. It is also case sensitive.

- public boolean startsWith(String to_check)

Eg:- string s1 = "There was a Ring"
boolean b1 = s1.startsWith("there"); // false
b2 = s1.startsWith("Th"); // True

13) contains(): - This funct^n checks & returns whether invoking object contains specified parameter string or not.

- public boolean contains(String to_check)

Eg:- String s1 = "Hello All";
boolean b1 = s1.contains("All"); // true.

3) trim() :- This function removes the wide space additionally present at the beginning & at the end of invoking String object.

- public String trim();

Ex:- String s1 = "--- There was a King ---";
s2 = s1.trim();
s2 => "There was a King";

21/3/23

4) charAt() :- This method obtains & returns and returns single character present at specified index in invoking String object.

- public char charAt(int index)

Eg:- String s1 = "Hello All";
char var = s1.charAt(4);
var = 'o'

- If invalid index no is provided, then this function throws StringIOOBE Exception.

5) toCharArray() :- This function converts & returns invoking String object into a char[].

- public char[] toCharArray()

Eg:- String s1 = "Hello All";
char a[] = s1.toCharArray();
a = [H|E|L|L|O| |A|L|L]

6) getBytes() :- This function converts & returns invoking string object into byte[] containing ASCII values of each character.

- public byte[] getBytes()

Eg:- String s1 = "AB CDE";
byte a[] = s1.getBytes();

a → [65|66|32|67|68|69]
0 1 2 3 4 5

7) toUpperCase() :- This function converts & returns invoking string object into All upperCase characters. This function creates new resultant String whereas the old string or invoking string object remains intact.

- public String toUpperCase()

Eg:- String s1 = "abcd";
-"- s2 = s1.toUpperCase()
s2 => "ABCD"

8) toLowerCase() :- This function converts & returns invoking string object into all Lower case characters. This function creates new resultant String whereas the old String or invoking String object remains intact.

4) 4th constructor creates String object & initialize it with characters of specified char[].

Eg:-
```
char ar[] = {'J','a','m','e','s'};

String s4 = new String(ar);
    s4 => "James";
```

5) 5th constructor creates a string object & initialize it with ASCII characters of specified byte[].

Eg:-
```
byte ar[] = {65,66,67};  ar
String s5 = new String(s5);
    s5 => "ABC";
```

6) 6th const^n creates a string object & initialize it with the characters from "start index" to "end index - 1" of specified char[].

Eg:-
```
char ar[] = {'The','J','h','e','-','a','a','s'}
String s6 = new String(ar, 1, 4);
    s6 =
```

7) 7th const^n creates a string object & initialize it with ASCII characters of specified byte[] from "start index" to "end index - 1"

Eg:-
```
byte ar[] = {65,66,67,68};
String s7 = new String(ar, 1, 3);
    s7 => "BC";
```


8) 8th constructor creates string object & initialize it with the value of string buffer object.

Eg:-
```
StringBuffer s1 = new StringBuffer("ABC DEF");
String s8 = new String(s1);
    s8 = "ABC DEF";
```

Following are some of commonly used functions of class String.

1) length(): - This function returns length of invoking string object. This function is defined as

public int length();*

Eg:-
```
String s1 = "ABC DEF";
int s2 = s1.length();
    s2 => 7
```

2) substring(): - This function obtains & returns a substring from specified startindex to endindex - 1. This method is defined as

public String substring(int st-index, int end-index

Eg:-
```
String s1 = "There was a King";
    s2 = s1.substring(3,13);
    s2 = "re was a K";
```

- We don't write anything before "this" & "super" but after that, it is allowed.

Eg: a.this; } Not allowed this.a; } allowed
a.super; } allowed super.a; }

We know that when a member function is called by using an object, then hashcode of invoking object get's passed as first parameter which is internally received as reference of "this". automatically present as first parameter.

- This generation of "this" is only in non-static member functions. Therefore a static member function is called by using class name.

** Java provides another special use of Keyword "this" and that is to manually call another constructor (of same class) for same object i.e we can execute multiple constructors for single object.

NOTES

- In C++, we know that we have "*this" pointer and by default it is always passed as first parameter ~~by~~ ~~JVM~~ and it holds/point to the address of invoking object.
- Similarly, In Java we have a Keyword "this" which also gets passed implicitly or by default as first parameter containing the "hashcode" of invoking object.

Imp: Use of keyword "this" is not allowed in body of static member function & in static block.

* In the bytecode, the "this" Keyword is typically represented as a reference to the current object's memory address, which is stored in a register called the "this" pointer. When a method or constructor is invoked, the JVM passes the "this" pointer as the first parameter, allowing the method to access the fields and methods of the current object.
- And therefore, we use class name with static entities as no "this" parameter is going to pass.

25/3/23

Refer foll° program:-

=>
```
class Student
{
    string sname;
    int sage;
    int sfees;

    public Student ()
    {
        Sname = "Noname";
        Sage = 20
        Sfees = 3500;
    }

    public Student (String nm, int a, int f)
    {
        Sname = nm;
        Sage = a;
        Sfees = f; }
```

* If specified string not present then returns -1.

. public int lastIndexOf(String to_check)

Eg:- String s1 = "There was a theater near the pond";
int a = s1.lastIndexOf("he");

a => 26

19) concat():- This function concatenates invoking string with parameter string and returns resulting string.

. public string concat(String ref/val)

Eg:- String s1 = "James";
s2 = "Gosling";
String s3 = s1.concat(s2);
s3 => "JamesGosling";

20) replace():- This function replaces specified string with replacement string and results a new string.

. public String replace(String curr_st, String rep_

Eg:- String s1 = "He was there";
s2 = s1.replace("was", "is");
s2 => "He is there";

by default, priority is given to local variables, formal parameters are considered as local variables.

* Keyword "this":- We are already familiar with the actual use of keyword "this", to differentiate between local variables and data members having same name.

. As a major diff" b/w keyword "this" in C++ & java is

☆ In C++, "this" is a pointer and therefore we use it with arrow operator "->"

☆ In Java, "this" is a reference & therefore we use it with "." operator for example

. class Test
{
int x, y;
public void assign(int x, int y)
{
this.x = x;
this.y = y;
}
}

· lexicographically ": The process of comparing ASCII values

14) compareTo():- This function compares invoking string object with parameter string object by considering & comparing ASCII values.

· public int compareTo(String toCheck)

#	Situation	Returns
·	Both strings are totally equal	0
·	invoking string is lexicographically smaller then parameter string	-ve value.
·	invoking string is lexicographically larger than parameter string	+ve value.

15) compareToIgnoreCase():- This function compares invoking string object & parameter string object without considering case sensitivity. This function performs lexicographical comparision.

· public int compareToIgnoreCase(String toCheck)

16) split():- This funct^n splits/divide invoking string object by specified string value. This funct^n is defined as

· public String[] split(String splitBy)

Eg:- String s1 = "There is a theater near the pond";
String s2[] = s1.split("_");
s2 = {"There", "is", "a", "theater", "near", "the", "pond"}
String s3[] = s1.split("th");
s3 = {"ere" is a eater

17) indexOf():- This function returns the index no. of first occurance of specified string in invoking string object. This funct^n is an overloaded function which are as followed

· public int indexOf(String toCheck)
· public int indexOf(String toCheck, int startInd
Eg:- String s1 = "There is a theater near the pond
int a = s1.indexOf("he");
int b = s1.indexOf("he", 15);
· after "15th index" when occured.

24/3/23

18) lastIndexOf():- This function obtains and returns the index no. of last occurance of specified string in invoking string object.

27/3/23

Eg:-
```
class A
{
    P.S.V.M(String args[])
    {
        S.O.pln("mains starts");
        Test t1 = new Test();  ← const" executes
        t1.method();
        S.O.pln("main ends");
        // Runtime.runFinalizersOnExit(true);
    } // scope of t1 expires.
}
```

- From above main method, we can experience the exact execution of finalize method which is by default false/off this is why we need to call finalize() manually.
  In above example simply uncomment the last statement & experience automatic execution of finalize() when scope of object t1 expires.
- Also remember that, similar to destructors, execution of finalizers happens in reverse order of declaration of objects.

Note:- In multithreaded environment if other parallel threads demands same resource, then we can manually release the resource by manually calling "finalize()"

different no's of columns.

* Arrays In Java *

- We know that an array is set of values of similar datatype. like other prog. lang. Java also supports array.
- We have one-D, two-D & dynamic size array

1) One Dimension Array :- A one dimensional array has only one column with specified no's of row. To declare 1-D array general syntax is

datatype array-name[] = new datatype[length];

OR

datatype[] arr-name = new datatype[length];

Ex:- int a[] = new int[10];
float b[] = new float[15];

** - Never mention size on L.H.S bcoz it's a reference & it'll show compile time error
- In Java we can declare array of variable length taken from user.

Eg:-
```
int n;
S.O.pln("Enter the size of array:");
n = sc.nextInt();
int ar[] = new int[n];
```

- In similar fashion, Java provides finalize() to release resources in our user defined class, we have to override finalize() as

```
public void finalize()
{
    ==
}
```

- The finalize() method is originally defined in "root" super class "Object" (java.lang.Object) as

```
protected void finalize() throws Throwable
```

- We override finalize() in our class with above specified signature if any change is made in above signature of finalize() then that finalize() becomes a normal concrete method which will not behave as expected.

Refer following class:-

```
class Test
{
    public Test()
    {
        == // is constructor
    }
    public void method1()
    {
        == // is normal m.f
    }
    public void finalize()
    {
        == //is finalizer method.
    }
}
```

- Now, when object of above class Test is declared, then constructor executes automatically but finalize method will not execute automatically when scope of object of class Test expires bcoz the automatic execution of "finalize()" method is OFF/false", Now we have two options for this, one is call finalize() manually (like normal member functions) whenever required.
- Enable its automatic execution (like destructor) by using "runFinalizersOnExit()". This mthd is static member func of class "java.lang.Runtime". This method is defined as.

```
public static void runFinalizersOnExit(boolean val)
```

```
public Student (int a)
{
    this();  ← calls to default/non-parameterized const" for same invoking obj.
    Sage = a;
}

public Student (String snm)
{
    this (snm, 21, 3100);  ← calls to 3-parameter const" for same invoking object.
}

public void showValues()
{
    s.o.pln("Name:-" + sname);
    s.o.pln("age:-" + sage);
    s.o.pln("fees:-" + sfees);
}
```

⇒
```
class A
{
    p.s.v.m(-)
    {
        Student s1 = new Student();
        --11-- s2 = ------11-- (25);
        --11-- s3 = ------11-- ("Junit");
        =
        =
    }
}
```

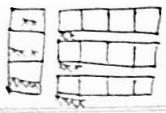
- Note that in above example two constructor executes for object s2 & s3
- for object s2 ⇒ default/non-parameterized constructor + (int) parameter constructor.
- for object s3 ⇒ all parameters constructor + (String) parameter constructor
- ** Remember that for above demonstrated use of Keyword "this" it must be first statement of any constructor.
- ** Remember that above demonstrated use of Keyword "this" is valid only from constructor, not from any normal member function.
- ** Remember that, above demonstrated use of "this" is valid to access constructor of own class only. (Not for Base class constructor)

* finalize() method or finalizers in Java.

26/3/23

- We Know that, In C++ we use distructor to release the resources releasing the resources is very important step so that the same resource becomes available to other functions. We also Know that in C++, distructor executes automatically when scope of an object expires.

[3] [4]
ref No of objects



Refer foll" prog:-

```
=> for(i=0; i<a.length; i++)
   {
      for(j=0; j<a[i].length; j++)
      {
         a[i][j] = sc.nextInt();
      }
   }

=> for(i=0; i<a.length; i++)
   {
      for(j=0; j<a[i].length; j++)
      {
         S.o.print(a[i][j] + "\t");
      }
      S.o.println();
   }
```

- We can create same dynamic block as dynamic two D. array directly by initializing elements

Eg:-

```
int b[][] = { {1,2,3},
              {4,5},
              {6,7,8,9,10},
              {12},
              {13,14,15}
            };
```

1	2	3		
4	5			
6	7	8	9	10
12				
13	14	15		

- For each loop:- specially to iterate on an array Java provides an updated version of for loop called as "for each loop". The general syntax to use for each loop is

```
for(datatype ref : array-nm)
{
   statements;
}
```

3/3/23

for ex:-

```
class Test
{
   p.s.v.m(String a[])
   {
      int a[] = {10, 20, 30, 40, 50};
      S.o.pln("Array elements are");
      for(int val : a)
      {
         S.o.pln(val);
      }
   }
}
```

- In above example, Note that reference "val" will refer each element of array "a" in each iteration, also remember that, for-each loop is like "read only" loop which can only fetch the value, we cannot overwrite/ update any value.

- While working on array, we can use field "length" to obtain length/size of invoking array.
- In 1D array "length" indicates no. of rows.
- In 2D array "length" -11- no. of rows & length of each row indicate no. of column.

- Even while initializing a two Dimensional array we must not specify length.

29/3/23

* Declaring array of Objects

- If logically required, then we can declare array of objects of any class to do this we have two steps:-

1 Step:- Declare array of references of required length.

2 Step:- Apply an object to each reference by using a loop.

for eg:-
```
Student s[] = new Student[10];
for(int i=0; i<s.length; i++)
{
    s[i] = new Student();
}
```

- Note that, while using step 2, we are using only default constructor (Non-parameterized) for every object. If logically required then we can also individually initialize each reference through required object & required constructor.

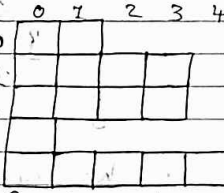
- Ultimately, all array level stores any value / Any ref stores hashcode/ values in reference of that.

Eg:-
```
Student s[] = { new Student(), new Student(
new Student(12, "ABCD", "Pune") };
```

- If we do not apply step 2, and directly call member functions using array of references then it will throw NullPointerException exception.

* Declaring dynamic sized array

- A dynamic sized array indicates a 2D array in which each row can have different no's of column for example:-



- To declare a dynamic array we have to follow following two steps:-

Step 1 Declare two dimension array (specify only no. of rows)

Step 2 Individually assign no. of columns to each row.

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

- By default, Array in Java is initialized with default zeros.
- While passing an array as actual parameter, do not use square bracket ([ ]) but while declaring an array as formal parameter we must specify [ ].

Note. Whenever, an object is required as data member, then always declare its reference as data member, & initialize it through it's constructor.

Eg:-
```
class Aa
{
    int a[];
    A a1;
    public Aa()
    {
        a = new int[5];
        a1 = new A();
    }
    +
    other m.f.
}
```

28/3/23

- If logically required, then we can also initialize one dimensional array by specifying list of elements in {-,-,-}.

Syn:- datatype ar-nm[] = {lists of elements};

Eg:-
```
int a[] = {10,20,30,40,50};
float b[] = {1.0f, 2.3f, 3.1f};
String c[] = {"one", "two"};
char d[] = {'A', 'B', 'C'};
```

Note:- While initializing array we must not specify length otherwise Compile time error.

In computer it is always like

y=column

* two Dimensional Array:- We know that In 2D array, we can specify No's of rows & No.s of column in individual [][]. To declare two dimensional array :-

row column

datatype ar-nm[][] = new datatype [len1][len2];

Ex:-
```
int a[][] = new int[4][3];
float b[][] = new float[5][8];
String c[][] = new String[8][15];
char d[][] = new char[15][45];
```

- While returning & an array from funct^n do not use [ ] but while receiving returned array use or specify [ ] & return type must be same datatype array.

- Similar to 1D array, we can also initialize 2D array but while initializing a 2D array, we must initialize it we must initialize it with row wise in a seperate pairs of {-}.

Eg:-
```
int a[][] = { {1,2,3,4},
              {5,6,7,8},
              {9,10,11,12},
              {13,14,15,16},
              {17,18,19,20}
            };
```

1	2	3	4
5			
9			
13			
17			

Eg:

```
class A                                   (old method (overridden method))
{
  public void method1()
  { == }
  public void method2()                   (virtual concrete method)
  { == }
}
class B extends A                         (new method (overriding method))
{
  public void method1()
  { == }
  public void method3()                   (virtual concrete method)
  { == }
}
```

- Note that, In above example method1() is getting override and therefore new overriding method1() will hide the accessibility of old overridden method1() and therefore when ever we call to method1() by using object of derived class B, it will access new overriding method1() of derived class B

```
class C
{
  P.S.V.m(String arg[])
  {
    B b1 = new B();
    b1.method1();   <- calls to new overriding method1() of class B
    == 
  }
}
```

- In Java, Whenever a method get's override into derived class, then that original overridden method of Base class is never accessible externally by using object of derived class

-> Refer foll^n example:-

```
class A
{
  public void mth1()
  { == }
}
class B extends A
{
  p.v. mth1()
  { == }
}
```

★★★

Public	itself and above are allowed	• In inheritance we can use this in order to specify the Access specifier in the Base derived class.
Protected		
Friendly		• Downward are weaker
Private		• Upward are stronger

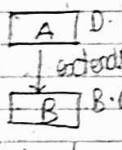
1/4/23

- Members of base class directly / locally access members of Derived class.
- When object of derived class is declared, first of all constructor of base executes, then constructor of derived class executes.
- When object of base class is declared, it executes constructor of base class only.

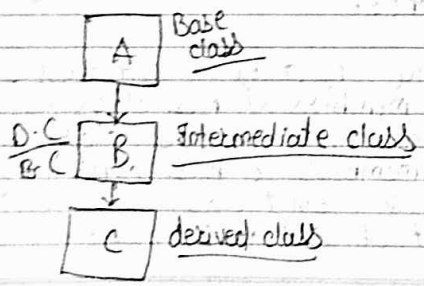
4/4/23

* Types of Inheritance :-

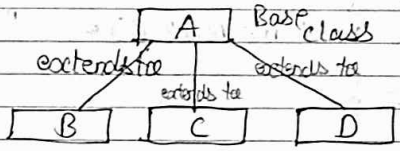
1) Single Inheritance :- In this the derived class derives properties of one base class, then it is single Inheritance.



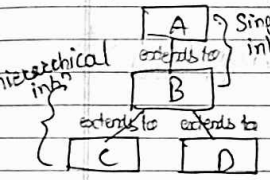
2) Multilevel inheritance :- When a D.C becomes Base class of another class then it is multilevel inheritance.



3) Hierarchical Inheritance :- When multiple derived classes derives properties of common base class then it is called hierarchical Inheritance.



4) Hybride Inheritance :- It is combination of more than one types of inheritances.



Method Overriding :- We know that, if method of a base class is with faulty logic or incomplete or incompatable logic, then such method of Base class should get overridden into derived class.

- Method Overriding means redefining a method of a base class into derived class with same signature.

- If we apply for-each loop on a two Dime. array, then in each iteration, for-each loop refers to each row & internally we have to apply a nested for loop that refers to each element of referred row.

Eg:-

```
for(int b[] : a)
{
    for(int val : b)
    {
        S.o.p(val + "\t");
    }
    S.o.println();
}
```

	0	1	2	3	4
0	1	2	3		
1	4	5			
2	6	7	8	9	10
3	11	12	13	14	

* Inheritance:- We Know that, inheritance means deriving properties of one class into another class.

- The class whose properties are being derived is called as base class/super/parent class.
- The class in which properties are being derived is called as derived/sub/child class.
- Remember that Java does not allow multiple inheritance therefore here we have four types of Inheritances:-

1) Single inheritance
   Multilevel inheritance
   Hierarchical inheritance
   Hybrid Inheritance.

- Here in Java we have Keyword "extends" to inherite properties. for example:-

Base class

```
class A          |  class B extends A
{                |  {
  ==             |    ==
}                |  }
```

- As compared to C++, here we can see two differences
  1) we use Keyword "extends" instead ":".
  2) No access modifier while inheriting.
- Remember that, every member of base class get's inherited into derived class with complete signature. Therefore a member of base class will remain with same access specifier as of base class, after inheritance into derived class.

Rules of inheritance or advantages of inheritance.

- By using object of derived class we can access members of base class also.
- By using object of Base class we can access members of base class only.
- Members of derived class can directly/locally (without object) access properties of base class.

- In above example, every derived class overrides method 1. therefore reference of base class A can call to method 1 but according to feature RuntimePoly. this ref. of B.C can receive object of derived class. & whenever method 1 is called by using ref. of class A, it will access method 1 of the class who's object been refered by reference of Baseclass A.
- The same mechanism is applied for Cosmic super class object Vs other Java classes

* Keyword "final".

- Java provides a multipurpose Keyword "final" that we can apply :-
  - with variable declaration.
  - with function defination (body)
  - with ~~function~~ class defination (Body)

a) with variable declaration :- When Keyword final is applied with variable declaration, then that variable becomes constant. alternately we can say a final variable cannot change its referal for example

-> final int a = 10;
final float b = 9.2f;
final String S = "dIEMS";
final A a = new A();

**** If a D.F. is declared as final, then prefer it to declare static.

- We Know that const^l final entities must be initialize.
- We can declare data members and local variables as final entities.

b) With function defination (body) :- When Keyword "final" is applied with function defination, then that function cannot be override into derived class. for example.

7/12/23

Eg :-

```
class A
{
    public final void method1()
    {
        == }  // is final method

    public void method2()
    {
        == }
}

class B extends A
{ public void method1()
  {     // C.T error.
        == } // final method1() cannot be overridden
    public void method2()
    {     // valid
        == } // method2() is not final in B.C

    public void method2(int p)
} { == } // valid // final method1() can be overloaded
```

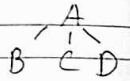
6/4/23

* Dynamic dispatch of method or acheiving runtime polymorphism.

• We Know that
B.C ref = D.C obj *
this mechanism is also applicable when reference of Base class is present as formal parameter of a function.

** • The same mechanism is also applicable in case of hierarchical & multilevel inheritance

-> class A
{ public void method1 ()
{ so.pln(-);
= }
}
class B extends A
{ public void method1 ()
{
so.pln("m1 of B");
}
}
class C extends A
{ public void method1 ()
{
so.pln("m1 of C");
}
}



class D extends A
{ public void method1 ()
{
so.pln("m1 of D");
}
}
class Test1
{
public static void calling (A ref)
{ =
ref.method1();
}
}
class Test2
{
p.s.v.m(String args[]) obj of A ∴ m1 of A
{
Test1.calling (new A()); obj of B ∴ m1 of B
Test1.calling (new B()); obj of C ∴ m1 of C
Test1.calling (new C()); obj of D ∴ m1 of D
Test1.calling (new D());
}
}

O/P
=> m1 of A
m1 of B
m1 of C
m1 of D

```
class Test
{
    p.s.v.m(-)
    {
        B b1 = new B();
        b1.method1(); // calls to new overriding  meth1()
        A a1 = new B();
        a1.mth1(); // calls to  —11—  —11—
    }
}
```

5/4/23

Rules for reference of Base class.

- When their is situation,
  [B.C ref = derived class object]*
  their are two rules for reference of B.C

i) The reference of Base class can access only members of B.C from object of Derived class.

ii) In case of overridden methods reference of Base class will access new overriding method of derived class from assigned object of derived class.

Eg:-

```
class A
{
    p.v.method1()
    {
        == }
    p.v.method2()
    {
        == }
}
```

⇒

```
class B extends A
{
    p.v.method1()
    {
        == }
    p.v.method3()
    {
        == }
}
```

⇒

```
class Test
{
    p.s.v.m(String args[])
    {
        A a1 = new B();
        a1.method1(); // valid - calls to mth1() of Derived class
        a1.method2(); // valid
        a1.method3(); // C.T error
    }
}
```

*** · The statement "super.super.x;" or super.this.x; or "this.super.x" has no meaning.

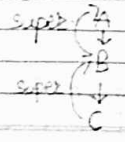
```
{ super(n, m);
  ac_num=acn;
  ac.type=act;
}
other m.
}

class Test
{
  p.s.v.m(--)
  {
    BankAcc ac1 = new BankACC
    (1234567259, 'S', "Ashwin", 95234567S);

    BankAcc ac2 = new BankAcc
    ("9876543211, 'C', "Junit", "2347659211");
  }
}
```

Note · Remember that use of Keyword "super" for such purpose will pass parameters to only immediate / exact B.C.

· Incase, in multilevel inheritance, we have to use Keyword "super" in constructor of each class to pass parameters to heirarchy class or exact class.



· As a precaution, always prefer to define a non-parameterized constructor in every class. If logically not required then, also make it empty body constructor.

7/4/23

2) To call overridden methods of Base class

· We know that, "Method overriding the new method of Derived class hides accessibility of original method of Base class this means, the original method of Base class never accessible externally" by using Object of Derived class.

· But if logically required, then the original method of B.C is accessible from exact d.C or immediate B.C by using Keyword "super".

foreg:-

```
class A
{   int x;
  public void method1()
  {
    ==}

  public void method2()
  {
    ==}
}
```

->

```
class B ext. A .
{   int x;
  public void method1()
  {
    ==
    method2(); // calls to method2() of
                  Base class A.
```

Eg:-
```
class A
{
  int x, y;
  public A(int p, int q)   <- Parameterized constructor.
  {
     x = p; y = q;
  }
  +
  other member functions.
}
class B extends A
{
  public B()
  {
    super(10, 20); // passing parameters to
    ≡              // parameterized constructor
  }                // of Base class A
  + other member fun^n.
}
class Test
{
  p.s.v.m(String arg[])
  {
    B b1 = new B(); // invoking default
    ≡               // constructor of Derived
  }                 // class B
}
```

- In above example note that we declared default object of derived class B which executes default constructor of derived class B and from there parameters are passed to parameterized constructor of Base class A by using Keyword "super"
- Remember that, for such purpose of Keyword "super", it must be first statement of constructor of derived class.
- It is also possible to provide external parameters to parameterized constructor of Base class. for this we have to define parameterized constructor in derived class also.

Eg:-
```
class Person
{
  String nm;
  long mb;
  public Person(String n, long m)
  { nm = n;
    mb = n;
  }
  + other m f^
}

class BankAcc extends Person
{
  long ac-nm;
  char ac-type;
  public BankAcc(long acn, char act, String n, long m)
```

- static only keyword that is valid to
- "x's gets constant"

c) With class defination :- When Keyword "final" is applied with class defination then that class cannot be derived further (or that class never be base class)

Eg:-
```
final class A
{
≡
}
class B extends (A) ← shows C.T error.
{ ≡ }
```

Note:
- A "final" D.M should be prefered to declare as "static" also.
- Constructor can never be "final" or "static" or "abstract".
- A final M.F can be static but a final M.F can be "abstract".
- A final class can never be "abstract" class.

Imp
- final + static → valid combination
- final + abstract → invalid combination
- static + abstract → invalid combination.

Note :-
- final D.M can also be initialized from constructor, static block or anonymous block.
- final local variables must be initialized at the time of declaration only.

be written anything as they are invoking keywords.

* Keyword "super" :- Java provides keyword "super" that is always to indicate / refer members of base class.

** Use of keyword "super" is valid only into derived class, to refer members of exact base class

- In Java, keyword "super" has two application
1) To pass parameters to parameterized const^{n} of base class.
2) To differentiate btw overridden methods of base class (To call overridden method of B.C).

8/4/23

3) To pass parameters to parameterized constructor of base class :- We know that when object of derived class is declared, first of all it executes constructor of Base class and then constructor of derived class executes.
- This situation is valid only in case of default constructor (Non parameterized / No argument)
- But a common situation appears when constructor of Base class is parameterized constructor. In other words, if Base class has only parameterized constructor in such case, it is responsibility of constructor of derived class to pass parameters to parameterized constructor of Base class by using keyword "super".

- Java provides a facility to directly apply Keyword "abstract" with class definition, even if that class does not have any abstract function in it. just for the purpose of not allowing instanciation of the class. This makes following two points:-

1) abstract functions must be inside an abstract class.
2) An abstract class may or may not have an abstract function in it.

* Interface *

- An Interface is set of inter-related Data members & member function. Where,
  - Data Members are by default "static" & "final".
  - Member functions are by default "abstract".
- To declare an interface, general syntax is

```
interface Interface-Name
{
    d.M & M.F
}
```

for ex:-

```
interface A
{   p.v set();
    p int x = 10;
    p v get() throws IOException;
}
```

- Interfaces & abstract classes both can have their own references
- class can only be used in two ways:- i)
- Interface can't have static & anonymous block.

13/4/23

- Here, "interface is a Keyword.
- Similar to classes, bytecodes of interfaces also gets created. It will be same name as interface name and extension ".class"
- ∴ In one program interface name & class name cannot be same.
- From above discussion, we can say member functions of interface are by default abstract. Therefore an interface is ultimately like an abstract class. Therefore, we cannot declare objects of interface. but reference will be always created.
- After defining interface, we can inherit it into a class by using Keyword "implements"

eg:-

```
interface A
{
    =
}
```

```
class B implements A
{                  (B: implementing class; A: implemented interface)
    =
}
```

- Remember that implementing an interface is similar to inheritance i.e every member of implemented interface inherits into implementing class and according to inheritance, the implementing class must override every abstract methods of implemented interface. Otherwise the implementing class must be declared as "abstract class".

Eg:-
```
abstract class A
{
    public void m1()
    { == }
    public void m2()
    { == }
    public abstract void m3();
}
```

Note:- An abstract class can have virtual functions but it is optional to have ~~it~~ abstract member function in abstract class.
- We do this to not allow creation of objects of that specific class.

- above class A is now an abstract class and an abstract class cannot be instantiated (Object wala). But an abstract class can be derived into derived class.
- But the rule is if an abstract class is inherited into a class then that D.C must override every abstract method of B.C. otherwise that D.C must be itself abstract class.

Eg:-
```
class B extends A
{
    public void method2()
    { == } // overridden abstract method of B.C
    public void method3(int p, float q)
    { == } // overridden abstract method of B.C
    public void method4()
    { == } // overridden abstract method of B.C
    public void method5()
    { == } // is concrete virtual method of D.C
}
```

that now any of the methods can be "final" as they do have body.

- Note that while overriding abstract methods of B.C 'A', into derived class 'B', we must not use Keyword "abstract".

12/4/23
#
abstract class:- From above discussion, we can say, a class which has atleast one abstract function, is an abstract class. and we have to manually apply Keyword "abstract" with that class definition.

As a characteristic, an abstract class cannot be instantiated i.e we cannot declare its object.

```
    =
    method1();//calls to itself makes it
                recursive.
    =
    super.method1(); // calls to method1() of
                //Base-class A
    x = 20; //assigns 20 to own d.m of X
    super.x = 30; //assigns 30 to D.m of class
                //A, X.
    }
    public void method3()
    {
      method2();
      method1();
      =
      super.method1();
      =
      x = 100;
      super.x = 200;
      =
    }
}
```

Note: Every member of BC can be accessed from derived class by using Keyword "super" even if it is not overridden.

10/4/23

* Keyword "abstract."

- To strongly support Run time Poly^m & life cycle skeleton Java provides Keyword "abstract". We can apply Keyword abstract to with
  - Member function
  - Class defination.

Note1 =>
- Data members can never be abstract
- Abstract M.F of class can never be "private".
- Abstract M.F of "Interface can never be "private" and "protected".
- An abstract M.F can never be static and final
- An abstract class can never be "final".
- A constructor can never be "abstract" & "final" & "static".

abstract member function:- a member function declared with Keyword "abstract" is abstract member function of that class.

* abstract funct^n don't have body.

for eg:-

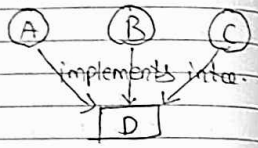
```
abstract class A
{
    public void method1() <- are virtual m.f
    {
      =
    }
    public abstract void method2();
    public abstract void method3(int x, float y);   } are abstract method
    public abstract void method3() throws Exception;
}
```

- above class conteins three abstract mem.funct^n In Java, if a class contains atleast one abstract method then that class also must be defined with Keyword "abstract". therefore above class A must be defined with Keyword abstract.

- Marker interface : interfaces having no data & methods.
- Always prefer empty interface rather than Empty class.
- Eg:- java.rmi.Remote → an empty interface.

15/4/2023

3) Implementing multiple interfaces into a class: It is also possible to implement multiple interfaces into a class. While implementing interfaces (Multiple) the list of implemented interfaces is seperated by ",".



Eg:-
```
interface A
{
  concrete abstract methods
  of interface A
  ==
}
```
→
```
interface B
{
  concrete abstract methods of interface B
  ==
}
```
→
```
interface C
{
  Concrete abstract methods of interface C
  ==
}
```
→
```
class D implements A, B, C
{
  overridden menthods of interface A, B, C
  +
  concrete virtual methods of class D
}
```
(D: implementing class; A, B, C: implemented interfaces.)

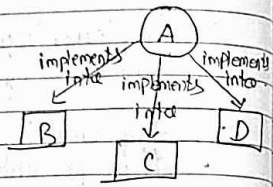
Note: In Java, we have many library interfaces which are empty body interfaces (NO, DMS, MS). Such interfaces are called as marker interfaces.

XXX Always prefer to define empty body interface over empty body class.

4) An Interface extending another interface:
Similar to classes, inheritence in interface is also possible, where an interface can inherit another interface by using Keyword "extends".
- Similar to inheritence of classes, properties of Base interface gets derived into derived interface.

Ex:-
```
interface A
{
  concrete abstract methods of
  interface A
  ==
}
```
(A: i/f; extends into; B: i/f)
→
```
interface B extends A
{
  concrete abstract methods of interface B
  +
  derived abstract methods of interface A
}
```

→ interface A
{
concrete abstract methods of interface A
=
}



implementing interface A
implemented interface A

→ class (B) implements (A)
{
concrete virtual methods of class B
+
overridden methods (abstract) of interface A
}

→ class C implements A
{
overridden abstract methods of interface A
+
concrete virtual methods of class C
}

→ class D implements A
{
overridden abstract methods of interface A
+
concrete virtual methods of class D
}

- In above skeleton we declare object of implementing classes B, C, D and call the required member functions.

3) Implementing multiple interfaces into a class

Notes

- It is never possible to inherit i.e "extend" a class into an interface as we know interface can't have methods with body.
- Even if we try to make a "abstract class" having every method pure virtual i.e without body and then we try to extend it into an interface then also it is not allowed.
- We know that By default "Cosmic super class Object" is inherited into every class of Java and that is every method of class object is also available in that abstract class and those methods have body.
- We also know that every class is having a default constructor if we don't write it manually and that constructor is having body and if we extend that class then interface will also have that body of constructor which is not allowed in interface.
- ☆ Interface is an entity which can never have Body inside it.

=> interface A
{
concrete abstract methods
≡
}

class B implements A
{
overridden meth. of i/f A
+
concrete virtual methods of class B
}

14/3/23

. We know that an interface is ultimately an abstract entity. Therefore we cannot declare objects of interface, but we can declare references interfaces as well as abstract classes.

. After implementing interface, we declare object of implementing class and call the required member functions in it. (either overridden or concrete).

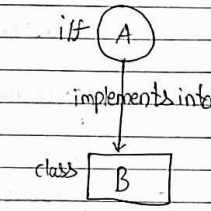
* Logical possibilites of using interface :-

1) Implementing an interface into a class.
2) Implementing an interface into multiple classes.
3) Implementing multiple interfaces into a class.
4) An interface extending another interface.
5) An interface extending multiple interfaces.
6) Implementing an interface into derived class.
7) Implementing multiple interfaces into derived class.

1) Implementing an interface into a class :-
. It is possible to implement an interface into a class.

-> interface A
{
concrete abstract methods of interface A
≡
}
-> class (B) implements (A)
{
overridden abstract methods of interface A.
+
concrete virtual methods of class B.
}

(B) ← implementing class
(A) ← implemented interface A



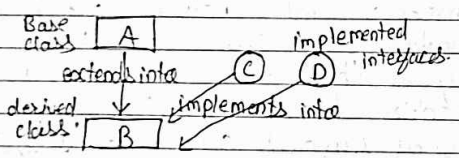
. In above skeleton we declare object of implementing class B and call the required member functions of class B

2) Implementing an interface into multiple classes
. It is also possible to implement an interface into multiple classes simply
1) Every implementing class must override every abstract methods of implemented interface

→ interface B
{
concrete abstract methods of interface B
=
}

→ class (C) extends (A) implements (B)
(C) derived class. (A) Base class (B) implemented interface.
{
inherited virtual methods of base class A
+
overridden abstract methods of interface B
+
concrete virtual methods of class C
}

7) Implementing multiple interfaces into a derived class:- It is also possible to implement multiple interfaces into a derived class



Eg:- class A
{
concrete virtual methods of class A
=
}

→ interface C
{
concrete abstract methods of interface C
=
}

→ interface D
{
concrete abstract methods of interface D
=
}

→ class (B) extends (A) implements (C, D)
(B) derived class (A) derived Base class (C, D) Implemented interface
{
inherited virtual methods of class A
+
overridden abstract methods of interface C
+
overridden abstract methods of interface D
+
concrete virtual methods of class B
}

* Dynamic dispatch of methods using Interface

In case of inheritance of classes, we Know that
B·C ref = D·C object

The same mechanism is valid in case of implementing class and implemented interface.

if ref = class object * (A) implements into → A ref = new BC
B

→ interface A
{
concrete abstract methods of interface A
≡
}

→ interface B
{
concrete abstract methods of interface B
≡
}

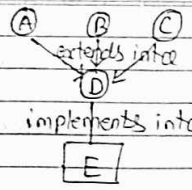
→ interface C
{
concrete abstract methods of interface C
≡
}

→ interface (D) extends (A, B, C).
(D: derived interface; A, B, C: Base interfaces)
{
concrete abstract methods of interface D
+
inherited abstract methods of base interfaces
A, B, C
}

→ class (E) implements (D)
(E: implementing class; D: implemented interface)
{
overridden methods of base interfaces
A, B, C, D
+

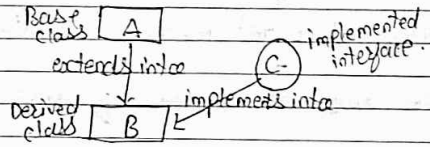
• In adapter classes they have created empty body of abstract methods of interfaces.

concrete virtual methods
of class E
≡
}



18/4/23

ii) Implementing an interface into derived class:-
It is also possible that a derived class can implement an interface along with extending a base class.



• In above case the derived class must first extend Base class then implement interface.

Note:
• In above case, if the base class contains one or more virtual functions with same signature as of abstract methods of implemented interface, then overriding such methods into derived class is optional.
• A very famous concept of "adapter classes" in java.awt is based on above mechanism.

Ex:-
class A
{
concrete virtual methods of class A
≡
}

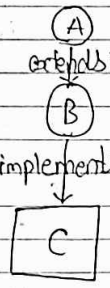
- If any class implements derived interface B then that implementing class must override every abstract method of Base interface A and derived interface B along with its concrete methods.

16/4/23

- As we know class can have another class in it known as inner class.
- Similarly it is allowed to have another class inside an interface.
- We know that every data member of an interface is by default "static" & "final" & every member function is by default "abstract".
- So, similarly any class inside an interface is by default a "static" class.

17/4/2023

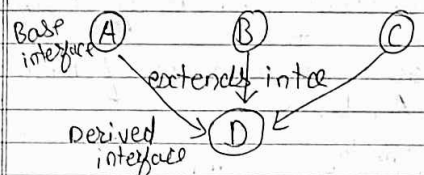
Eg:- interface A
{
concrete abstract methods of interface A
}
≡
{
⇒ interface (B) extends (A) ← Base interface; derived interface.
{
concrete abstract methods of interface B
+
inherited abstract methods of interface A
}



→ class C implements B
{
overridden abstract methods of base interface A
+
overridden abstract methods of derived int. B.
+
concrete virtual methods of class C.
}

- ultimately we declare object of implementing class C and ~~etc~~ call to required member functions defined in it.

5) An interface extending multiple interfaces
Similar to multiple inheritance of class in C++, Here in Java multiple inheritance of classes in not valid but multiple inheritance of interfaces is allowed i.e an interface can extend multiple interfaces where list of Base interfaces is seperated by comma(,).



- Here also abstract members of base interfaces gets derived into derived interface and if any class implements this derived interface D, then it has to override every abstract methods of base interfaces and derived interface.

overriding.

4) Even after overriding, static method of interface & static mthd of implementing class, both are available externally.

difference between static mthd of class & interface

*** 1) static method of class gets derived/inherited into derived class. Therefore, can be called by using base class name or derived class name with dot (.) operator.

*** 2) static method of interface cannot gets derived/ inherited into implementing class. ∴ static mthd of interface can be only called by using interface name with dot (.) operator (Not by using implementing class name.

CERTIFICATE

This is to certify that

Master / Miss _____

studying in _____ *School*

Standard _____ *Division* _____

has completed all the assignment satisfactorily and has been graded _____ *for the year* _____

in _____ *Subject*

School Stamp

Subject Teacher
Date

Principal

- We know that an interface is replacement of runtime Poly situation equivalent to multiple inheritance. We also know that a class can implement multiple interfaces and ultimately all implemented interfaces becomes base classes of implementing class. Therefore now object of implementing class is eligible everywhere where references of implemented interfaces are available as formal parameter.
- above sRelation is a good example, u

Eg:- class X implements A, B, C
{
    ≡
}

-> class Test1
{ p.v. method1(A ref)
  {
    ≡
  }
}

-> class Test2
{ static p.v. method2(B ref)
  {
    ≡
  }
}

-> class Test3
{ static p.v. method3(C ref)
  {
    ≡
  }
}

-> class Test
{ p.S.v.m(String args[])
  {
    Test1.method1(new X());
    Test2.method2(new X());
    Test3.method3(new X());
    ≡
  }
}

22/4/23

default & static methods in interface.

- From JDK 1.7 onwards we can define static member function & default member functions in interface with body.

a) "default" method of interface:-

1) It is defined with Keyword "default".
2) It is a non-static member function.
3) It can be override in implementing class as non static only.
4) After overriding into implementing class, the "default" mthd. of interface gets hidden and always new overriding method of implementing class is alwa available, externally.

b) "static" method of interface:-
1) It is defined with Keyword "static".
2) It can be overridden into implementing class as static only.
3) because of "static" it cannot get's hidden after

- In above scenario also the reference of interface can call the only common methods between them & overridden methods defined in implementing class.

19/4/23

Eg:-
```
interface A
{
    public void method1();
}
-> class B implements A
{
    public void method1()
    {
        ==  // is overridden
    }
    public void method2()
    {
        ==  // is concrete of class B
    }
-> class Test
{
    p.s.v.m(String args[])
    {
        A ref = new B();
        A.method1(); // is valid: overridden of A
        A.method2(); // concrete method of B.
                     // :. invalid.
    }
}
```

- Same mechanisms of R.T.P is applied when reference of interface is used as formal parameter of a function.

Eg:-
```
interface A
{ public void method1(); }

-> class B implements A
{
    p.void method1()
    { == }
}
-> class C implements A
{ p.void method1()
    { == }
}
-> class D implements A
{ p.void method1()
    { == }
}
-> class Test2
{
    p.s.void calling(A ref)
    { ref.method1(); }
}

-> class Test
{ p.s.v.m(String arg[])
    {
        Test2.calling(new D()); // mI of class D
        Test2.calling(new B()); // mI of class B
        Test2.calling(new C()); // mI of class C
    }
}
```

Saathi

Date __/__/____

Note! · While updating a package, if we give a public entity name same as any existing bytecode name, then this new public entity (same name) will override that existing bytecode without any notification or asking for permission.

25/4/23

* Importing a Package *

· Similar to library pkgs, we can also import user defined pakags by using Keyword "import".

Eg:- - import pk1.*;
// will import all public bytecodes of pk1.
- import pk1.A;
// will import only A.class from pk1.

· After importing bytecodes from required pkg, we can use it in any program for any required purpose:-
- declaring object
- inheriting
- using as parameter, etc

Eg:- If we consider above package pk1 and its public bytecodes

```
import pk1.*;
class X
{ p s v.m(---)
  {
opt^n 1 -> A a1 = new A();
```


Saathi

Date __/__/____

OR

```
opt^n 2 -> pk1.A a1 = new pk1.A();
           =
  }
}
```

· above example demonstrates that, we are importing all bytecodes from package pk1 & declaring object of class A of it.

· If we are importing any user-defined byt package and willing to use any bytecode the prefer to use that bytecode along with package name as shown in option 2

Refer following

```
import pk1.*;
· class Test extends pk1.B
{
  =
}

· class Test2 implements pk1.C, Runnable
{
  =
}

· public void calling(pk1.C ref)
{
  =
}
```

Saathi

Date

24/4/23 File: A.java

Eg:-
```
package pR1;
import java.io.IOException;
import java.io.DataInputStream;

public class A
{
  ==
}
```

- The main purpose of declaring package is to indicate interpreter about current bytecode that these bytecodes belongs to specific project/ application and Hence effect of access specifiers must be according to it.
- Also remember that, in one program only one entity can be public either class or interface is allowed to define and the program must be saved with that public entity name.

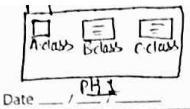
Eg:-

File: A.java
```
package pR1;
public class A
{
  ==
}
```

File: B.java
```
package pR1;
public class B
{
  ==
}
```

File: C.java
```
package pR1;
public interface C
{
  ==
}
```

- After declaring package a java prog. can be compiled normally by using JDK tool 'javac' but there is a special command to compile a prog. with packages and it is



Saathi

Date

```
javac -d . filename.java ↵
```

for example above 3 prog. should be compiled as

-> javac -d . A.java ↵
-> javac -d . B.java ↵
-> javac -d . C.java ↵

- This will automatically create a folder/directory with "package name pR1" and will place all 3 bytecodes A.class, B.class & C.class in it.

Updating a package:- Even after numbers of days if we want to add more bytecodes in existing package, then also we can do it.

- Simply prepare a java file with required public entity class or interface and declare same package name (which one to update).

Eg:-

File: D.java
```
package pR1;
public class D
{
  ==
}
```

File: Dd.java
```
package pR1;
public interface Dd
{
  ==
}
```

- When above programs gets compiled by using "javac -d . filename.java", then above four byte codes D.class & Dd.class will be added in package folder pR1.

S. No.	Date	Title	Page No.	Teacher's Sign/Remarks

Date __/__/____

Saathi

* Packages *

- A package is set of classes & interfaces or it is a set of bytecode.
- Actually a package is virtual connection between multiple bytecodes that we generate through individual program.
- When we declare packages, each package creates an individual folder/directory in which bytecode are placed.
- A good programmer always prefers to define one class or interface in one program & connect/relate them to a package & giving it same package name.
- To declare a package, general syntax is

```
package package-name;
```

*

- Here, "package" is a keyword.
- Note that package is only a declaration to indicate JVM that this prog. belongs to same project.
- Also note that a package is only a declarat° it does not have any body and also remember that package declarat° must be first statement of any program. even required packages are imported after package declarat°.

Page No.

Saathi

Date __/__/____

Eg:-
```
class A extends Thread
{
    public void run()
    {
        ≡
    }
}
```
→
```
class B extends Thread
{
    public void run()
    {
        ≡
    }
}
```
→
```
class C extends Thread
{
    public void run()
    {
        ≡
    }
}
```
→
```
class Test
{
    public static void main(---)
    {
        S.o.pln("Main Starts.");
        A aJ = new A();
        B bJ = new B();
        C cJ = new C();
        aJ.start();
        bJ.start();      } Starting threads.
        cJ.start();
        S.o.pln("Main ends.");
    }
}
```

Saathi

Date __/__/____

- By this, all three threads executes in parallel, but the order of execution is not fixed.
- ☆. The execution of threads depends on complexity of statements, availability of resources and other background processes.

Note. From above discussion threads looks more powerful then normal functions but it is not correct. At any stage, normal functions are more powerful & assured and accurate then a thread.

30/4/23

- We can't run ANY Thread more than once for the same object. Because it is believed that a thread is not the run() method inside a Thread enabled class but its the "object" of that Thread enabled class and hence it is only possible to execute run() method only once for a single object throughout the program.
- It is possible to create another object of that class and start the method by that object.
- In reality there is nothing like parallel execution. It's just the context switch which happens between all the running threads. In a normal senario only one Kernel level thread is allocated to all the thread of the program and hence that Kernel level thread executes every thread of that program.

Saathi

Date __/__/____

Creating a thread:-

- There are two options/ways to create a thread
1) By extending class "java.lang.Thread".
2) By implementing interface "java.lang.Runnable".

1) Extending class java.lang.Thread

Eg:- class A extends Thread → becomes thread-enabled class.
{
≡
}

- Above class A becomes a thread-enabled class in which we have to override "run()" we override "run()" as

- public void run()
{
≡
}.

Eg:- class A extends Thread
{
public void run()
{
≡
}
}

Saathi

Date __/__/____

- If we do any change in above signature of "run()" then it becomes normal concrete method, and it will not execute in parallel.
- Also remember that "run()" is not abstract method of class Thread also remember that originally this method is of interface java.lang.Runnable.

2) Implementing interface "java.lang.Runnable"

Eg:- class B implements Runnable
{ → class B becomes thread-enabled class.
≡
}

29/4/23

- Above class B becomes thread-enabled class in which we override "run()". Remember that "run()" is abstract method in "interface Runnable" therefore, we must override "run()" if we implement "interface Runnable"

Eg:- class B implements Runnable
{
public void run()
{
≡
}
}

*- In any of above case to execute "run()" parallely as threads, we execute them by using "start()" of thread life-cycle.

Jar :- Java Archives

Saathi

Date __/__/____

Note! · A class cannot be private or protected. It can be only public or default friendly.

· Only public bytecodes are imported outside the package. This means default friendly bytecodes indirectly gets hidden/unavailable outside the package.

· We can also create ".jar" files from packages by using JDK tool "jar" when a jar file is created, it only contains bytecodes. simply add any jar file in a project and start using bytecodes from it as reuseable bytecodes.

* Multi-Threading *

· We know that when a function calls to another function, the calling function transfers its own controls to the body of called funct^n and calling funct^n gets paused.

· After execution of called funct^n the controls returns back to the body of calling function & execution of calling function resumes. This type of execution is called as linear execution or sequentiational execution where only one funct^n executes at a time.

· This linear execution is suitable in case of procedural programming where order of execution of functions is important. but in any multitasking environment, such linear execution is time consuming.

· The linear/sequential execution

- execution time increases

Page No.

· Unhandled Exception :- Without try-catch block.

Saathi

Date __/__/____

- resource utilization is minimum.
- no parallel processing happens.

28/4/23

· Java provides multithreaded environment in which we can execute multiple threads, parallely threads are special functions which executes in parallel with each other, without disturbing each other's execution & without acquiring controls from calling function.

Note! Threads are totally independant of each other, even an unhandled exception occurs in one thread, then that single thread will only terminate. Other threads executes uninteruppetedly. Even main function also.

· If we apply multiple threads in a program, then

- Execution time reduces.
- Resource utilization increases.
- Application becomes multi-tasking

· Strictly remember that, threads have execution advantage over normal function but smartly decide when to prefer a thread & when to prefer a normal function. Remember that threads are only to perform, therefore never prefer I/O operation, resulting values, interdependant operations in a thread.

*· Strictly remember, never be dependant on execution order of threads.

Page No.

Saathi

Date __/__/__

- after invoking stop method i.e stop() thread goes to dead state.

Valid calling possibilities :-

1) When one thread starts another thread -
- Both threads executes parallely, the calling thread will not pass any controls to called thread & therefore both threads executes parallely.

2) When one thread calls to a normal function -
- The calling thread transfers its own controls to the normal function & itself gets paused (Because a normal function demands / required controls). Now, this normal function executes in parallel to other ongoing threads.

3) When a Normal function starts thread -
- That normal calling function does not transfer it's own controls to thread but that thread starts by its own & now normal calling function & thread executes in parallel.

4) When a Normal function calls to another Normal function -
- The normal calling function transfers its own controls to normal called function and itself gets paused.

Saathi

Date __/__/__

4/5/23
5/5/23

* Thread Priority *

- We know that, multiple threads executes in parallel & execution of threads can be in any order/sequence this means we cannot be dependant on execution order & result of threads but if logically required, then we can set priority of thread by using "setPriority()" This method is of class thread and defined as "public void setPriority(int priority)".
- We can set priority of thread by giving it a value from 1 to 10. To specify the order of priority this class provides three static constants they are
- Thread.MAX_PRIORITY = 10
- Thread.NORM_PRIORITY = 5
- Thread.MIN_PRIORITY = 1
- It is Not mandatory to use this constants to set the priority we can also use any integer console value from 1 to 10.

Eg:-
```
A a1 = new A();
B b1 = new B();
C c1 = new C();
a1.setPriority(10); or a1.setPriority(Thread.MAX_P);
b1.setPriority(5); or b1.setPriority(Thread.NORM_P);
```

running state by using "yield()", also into blocked state by using "sleep() or suspend()" and also into dead state by using stop().

4) Blocked state :- A thread is in blocked state when it's execution is temporarily paused.
. From blocked state a thread can move into running - runnable state by using resume() and into dead state by using stop().

5) Dead state :- A thread is in dead state when its execution is completed or manually stopped by using "stop()".
. Once a thread goes to dead state, it can never be started again & never be in new born state again.

Thread lifecycle Methods :-

1) start() :- This method starts execution of thread lifecycle, and takes thread to running - runnable state. This method is defined as

```
public void start()
```

2) yield() :- This method is like pass (skip the turn). This method skips execution of thread in current turn & takes thread from running state to runnable & from runnable to running. This method is defined as

```
public void yield()
```

3) sleep() :- This method takes every thread into blocked state & temporarily pause their execution.
. This method is defined as :

```
public static void sleep(int time ms) throws InterruptedExc
```

. This method temporarily pause execution of thread, and automatically resumes when specified no's of miliseconds completes/elapses.

4) suspend() :- This method temporarily pause execution of invoking thread object and takes into blocked state. This method is defined as

```
public void suspend()
```

. If a thread is sent to blocked state by using suspend(), then it will not automatically resume but we must call to "resume()" for same object, to resume execution of that thread.

3/5/23

5) resume() :- This method resumes execution of thread which is paused / blocked by using suspend(). This method is defined as

```
public void resume()
```

6) stop() :- This method is to manually terminate/stop the execution of thread. This method is defined as

```
Public final void stop()
```

Thread lifecycle.

- This is the only lifecycle in which we manually call the methods of lifecycle.
- a2.join(a1) :- This will automatically start Thread 'a2' after complete execution of a1 without calling start().

Saathi

Date __/__/____

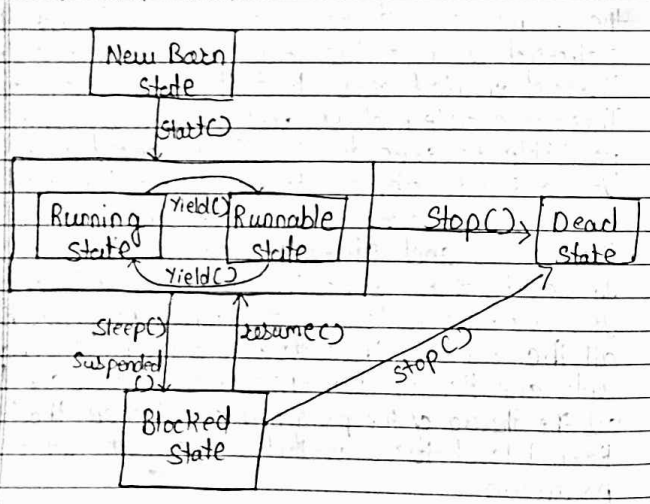
1/5/23

* Thread Lifecycle *

- During the complete life span of an ~~object~~ thread it goes through following five phases :-

1) New Born state
2) Running state
3) Runnable state
4) Blocked state
5) Dead state.

- Following diagram shows valid state transformation of a thread, with thread lifecycle methods responsible for transforming into that state.



Saathi

Date __/__/____

Note:
- Above diagram shows valid state transformation if any invalid transformation is made, then it throws "IllegalThreadStateException".
- Any Thread executes exactly once. Once a Thread reaches to dead state it can never be restarted or sent to new born state again.
- We have to manually call Thread lifecycle methods for legal state transformation. It will not happen automatically.
- All above thread lifecycle methods are defined in class "java.lang.Thread". Therefore extending Thread is recommended.

2/5/23

Thread lifecycle states :

1) New Born State :- A Thread is in new-born state when it's object is just declared, but never started.
   - From new born state, a thread can move into running, runnable state by using "start()".

2) Running state :- A thread is in running state when its execution is currently going on.
   - From running state, a thread can move into runnable state by using "yield()", and into blocked state by using "sleep() & suspended()" and also into dead state by using "stop()".

3) Runnable state :- A thread is in runnable state when it is already started but waiting for it's turn.
   - From runnable state, a thread can move into

→ class Test2
{
public static synchronized void InsertData(--)
{
=
}
}

- Above both static functions are now synchronized and hence only one thread can now enter into it so, that the resources could not be manupulated by any external thread.

2) Calling a function from synchronized block:- Consider we are using a library / built-in fun^n which is not with synchronized but still it is using an external resource. If such functions are required to get synchronized externally, then we have to call such functions from synchronized block. The synchronized block must be used as part of function and requires a thread object as parameter for example:-

→ class A extends Thread
{
public void run()
{ =
synchronized(this)
{ Test1.prMsg("A"); }
} } =

class B extends Thread
{
public void run()
{
=
synchronized(this)
{
Test2.insertData("—");
}
=
}
}

10/5/23

* Inter-thread Communication *

- We know that, Execution of threads is totally independant of each other, & we cannot guess any purposeful output of execution of multiple threads.
- If logically required, then we can make a external coordination between multiple threads just for the purposeful execution order of multiple threads. This mechanism is called as inter-thread Communication.
- In other words ITC is a mechanism where multiple threads communicates with each other just for the purpose of pause-resume of each other's execution. To do this we have three methods:-
  - wait()
  - notify()
  - notifyAll()

```
    }
}
```

→ class Test2
```
{
  public static void InsertData(String tab_name,
      String query, String tab_path)
  {
    Connection con = DM.getConnection(tab_path)
    Statement st = con.createStatement();
    st.executeUpdate(query);
    con.close();
    S.o.p.n("Data Saved...");
  }
}
```

- Both above static functions are definately candidate of concurrent situation when multiple thread affects above function at the nearby/same time.

9/5/23

- Such situations needs to be resolved by the mechanism of thread synchronization. Thread synchronization is the mechanism which applies two phase locking system. When one thread enters into a resource/function then an internal object called "monitor" locks the resource so that other threads cannot enter into it. But when execution of such resource/function ends, the monitor object unlocks the resource so, that other threads

Eg: daemon thread is an internal object/thread which starts automatically when execution of program begins or starts monitor & garbage collector, listner object.

can enter into it. This lock, unlock process is done by an internal daemon thread called as "monitor".

- Here, in Java, to apply thread synchronization, there is a Keyword "Synchronized". To make any resource as synchronized resource we have two options:

1) Apply Keyword "synchronized" with function defination.
2) Call a function from "synchronized" block

Note. Here in Java, We have a few execution blocks such as
- static block
- anonymous block
- Synchronized block
- try - catch - finally block

- Among them static & anonymous block are part of class where as synchronized & try-catch-finally block are part of member function.

1) Keyword "Synchronized" with function defination: When Keyword "synchronized" is applied with function defination, that function comes under monitoring of "monitor" object.

Eg:
```
class Test1
{
  public static synchronized void prMsg(String n)
  { == }
}
```

Saathi

Date __/__/____

Note! · Priority of thread should be set before starting that thread.

- If priority is set other than 1 to 10, then setPriority() will throw "IllegalArgumentException" exception. This exception is an unchecked exception.
- If multiple threads have same/equal priority, then their is no effect of setting priority.
- By default priority of thread is NORM_PRIORITY (5).
- We can obtain priority of any thread by using "getPriority()". It is also defined in final class Thread defined as follows.

```
public int getPriority()*
```

** · Current operating systems does not allow the effect of priority of any thread. It is considered that by giving priority to thread, any ongoing incomplete operations of thread gets postponed. Which affects existing background operating system. Therefore current O.S. does not allow setting thread priority.

Reason of →

- If required to display any exception messages, we can give a name to thread. To give a name to thread, we have setName() method of class Thread, This method is defined as:

```
public void setName(String th_name)*
```

- We can obtain name of any thread by using getName() method of class Thread.

- Daemon thread :- The threads which starts automatically with our main thread as we run our java program.

Saathi

Date __/__/____

Which is defined as:

```
public String getName();*
```

8/5/23

Imp Thread Synchronization

We Know that, multiple threads executes in parallel, without disturbing each others execution. We also Know that threads executes independantly, and in any order/sequence, their is no algorithm from which we can guess execution of thread. So that any purposeful operation gets completed in particular order.

- Consider a situation where our program is using a resource through a function. and multiple threads enters into that same funct^n at nearby/same time then both threads will handle that function & will manaplate/overwrite each other's values. Hence such function will not successfully complete specified task. Their might be either exception or invalid values.
- In Database concepts, such situations are called as concurrent situations where multiple queries at a time affects to a database.

Eg:-

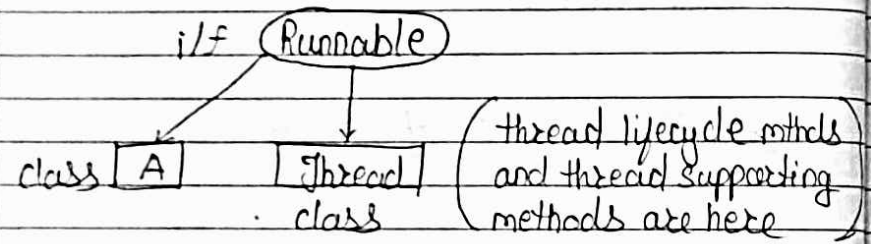
```
class Test
{
    public static int x;
    public static void printMsg(String n)
    {
        int i;
        for(i=1; i<=10; i++)
        S.o.p.ln("In thread"+n+"when i="+i+"x="+x);
```

Post-increment & pre-increment, working is the

Saathi

Date __/__/____

Implementing interface Runnable is valid opⁿ to create a Thread but the point is every thread life-cycle method & other thread supporting methods are defined in class Thread. Therefore if we implement interface Runnable then those methods are unavailable to our class.



- To relate our user defined class with class thread, class Thread provides a parameterized constructor.

Thread(Runnable ref)*

Through this constructor we can relate object of our implementing class with class Thread. To use thread lifecycle mthds & other thread supporting methods for object of our implementing class.

Eg:-
```
class A implements Runnable
{
   p. void run()
   {
      =
   }
}
```


Saathi

Date __/__/____

```
class B implements Runnable
{
   p. void run()
   {
      =
   }
}
class C implements Runnable
{
   p. void run()
   {
      =
   }
}
class Test
{ p.s.v.m(String args[])
  {
     A a1 = new A();
     B b1 = new B();
     C c1 = new C();

     Thread t1 = new Thread(a1);
     Thread t2 = new Thread(b1);
     Thread t3 = new Thread(c1);

     t1.start(); // will execute a1 as thread.
     t2.start(); // ——— " — b1 — " —
     t3.start(); // ——— " — c1 — " —
  }
}
```

Saathi

Date ___/___/___

Saathi

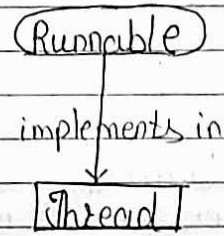
Date ___/___/___

11/5/23

* Interface Runnable :- We know that, to create thread we have two options/ways.

- By extending class "java.lang.Thread".
- By implementing interface "java.lang.Runnable"

. Remember that interface Runnable is implemented in class Thread



. This interface runnable contains abstract declaration of run() method.

```
. public interface Runnable
  {
      public void run();
  }
```

. This means if we implement interface Runnable then overriding run() is compulsory

Eg:-

```
class A implements Runnable
{
    public void run()
    {
        == }
}
```

- All methods having time as parameter throws InterruptedException
- ★ All the overloaded method in Java will have same signature i.e if one of its method consist or throws any exception then all overloaded methods will throw same exception.

Saathi

Date ___/___/___

- The "wait()" pause execution of invoking thread object. this method is an overloaded method and has following three signatures-

final · public void wait() throws InterruptedException
final · public void wait(int time-in-ms) throws -11-
final · public void wait(int time-in-ms, int additional-nanosec) -11-

- notify():- This method resumes execution of invoking thread object (mostly paused by using 1st signature of wait i.e no parameter). This method is defined as

[public void notify()]

- notifyAll():- This method resumes execution of every thread which is paused by any signature of pwait(). This method is defined as

[public void notifyAll()]*

- If any thread is paused by using 2nd & 3rd signature of "wait()", and such thread gets notified even before its timer completes / elapses, then such thread will throw InterruptedException exception.

Note. This InterruptedException exception is checked exception therefore it must be thrown or caught by using try-catch block.

Saathi

Date ___/___/___

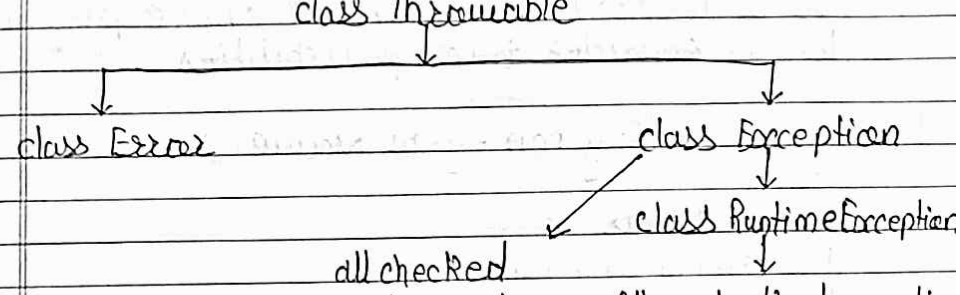
- Above interthread communication related methods works only on synchronized resources.

wait()	sleep()
· It is member funct^n of cosmic super class object.	· It is m.f of class Thread.
· wait() affects to only synchronized resources	· sleep() affects to any method / thread.
· wait() is non-static	· sleep() is static.

- If wait() method is invoked from "run()", then it run() will not allow to throw InterruptedException but here we must have to use try-catch block only.

- ArrayIndexOutOfBoundsException :- occurs when an invalid index no. is provided in array.
- StringIndexOutOfBoundsException :- occurs when any invalid index no. is provided in string operation.
- NULLPointerException :- occurs when any uninitialized reference calls to a member function.
- IllegalArgumentException :- occurs when any illegal actual parameters are passed to a function such as "setPriority()" & to constructors of class Color
- IllegalThreadStateException :- occurs when any illegal thread state transformation happens.
- NumberFormatException :- occured by Parse methods when any invalid is provided to parse.
- InputMismatchException :- occured by member functions of class Scanner. When any invalid value is provided as input which is irrelavent to its datatype.

Exception class hierarchy :- We know that Java provides numbers of library classes that are related to exception situation. following class hierarchy shows the relationship between exception classes



* Effect of Unhandled Exception : We Know that, an unhandled exception means for which we have neither used "throws" nor used try-catch block. If such exceptions occurs then program / Thread execution terminates immediately.

Eg 1
```
class Test
{   p.s.v.m(--)
    {
        int a = 10, b = ;
        float c = 0;
        S.o.pln("main starts");
        S.o.pln("permorming division");
        c = a/(float)b;
        S.o.pln("Division is:" + c);
        S.o.pln("main ends");
    }
}
```

Exception = Situation.

proper exception handeling mechanism.

- Here in Java, we have two options for handling an exception
  a) By using try-catch blocks
  b) By using Keyword "throws"

- Handling an exception means providing a solution or alternate of exception. Based on this division there are two types of exception.
  i) Handled exception ii) unhandled exception

i) Handled exception means for which solut^n /alternate is provided for that exception.
ii) Unhandled exception means for which solut^n /alternate is not provided. If an unhandled exception occurs, then program /thread execution terminates immediately.

Types of exception

- Based on situation of occurance of exception, there are two types of exceptions
  i) checked exception
  ii) Unchecked exception

i) Checked exception:- Checked exceptions are the exceptions which occurs during compile time. Actually there are many library functions that throws (possibility of) exception. If we call such functions

checked exception:- bugs of invalid statement.
unchecked -"- :- bug of invalid statement.
- only funct^n & constructors throw exception.

then the calling function must have to handle or manage that exception otherwise compiler shows error message related to non-handling /unhandling that exception. And hence, feel that exception occured during compile time.

Eg:- IOException - readLine()
SQLException - executeQuery(), executeUpdate()
ServletException - service().
InterruptedException - wait(), sleep().
UnKnownHostException:- getLocalHost(), getByName

ii) Unchecked exception:- These are the exceptions which directly occurs during run time mainly unchecked exceptions are bugs of some statements which could not perform as expected & hence such statements throws exception. Remember that in case of unchecked exceptions, Keyword "throws" will not be helpful we have to prefer only try-catch block.

- If any constructor throws an exception, then the function where its object is declared must also throw/handle that exception.
- If constructor of a class throws an exception, then constructor of it's derived class must also throw the same exception.

14/5

Eg:- ArithmeticException:- Occurs when any illegal arithmetic operat^n is performed.

- Similarly, through this we can apply any thread lifecycle method & other Thread supporting methods to object of implementing class.

12/5/23

- There are two basic function in thread:-
  i) isAlive()
  ii) join()

i) isAlive() method checks whether invoking thread object has gone to dead state or not. (It checks whether invoking thread object is still alive or not) This method is defined as

public boolean isAlive()

- This function returns "true" if thread is still alive on that moment; otherwise returns false.

ii) join() method makes other thread wait to start unless invoking thread object goes to dead state. (In other words other threads will start only after invoking thread goes to dead state)

- This method is an overloaded method
  - public void join() throws InterruptedException
  - public void join(int after-millisec) throws -11-

Eg:
```
b1.start();
c1.start();
a1.start();
a1.join(); ← will make other threads to wait till invoking thread object is not finished
```

- In C++ we have only 3 Keywords: try, catch, throw (in U.C.) Not in use
- In Java we have 5 Keywords: try, catch, finally, throw, throws.

13/5/23

* Exception Handling *

- Exception is a runtime situation that occurs during runtime.
  Previously, in C++ also we had exception handling mechanism but in C++, exception handling is totally manual process. In C++, we have to manually check for exception and sent exception object to catch block manually by using Keyword "throw".
- But here in Java, exception handling mechanism is totally automatic process. Here we have numbers of library classes, related to particular exception situation.
- *** Whenever an exception occurs, Java runtime system generates an object of related exception class (fills it with exception's meta data) and return it to the program.
  And it is program's decision whether to handle that exception using try-catch block or throw that exception object back to calling function by using Keyword "throws"
- Remember that, exception is the situation which occurs, it is neither fault of programmer nor the fault of end user. It is highly impossible to prepare an application/program which does not have any exception but only as a good programmer, we can think about handling that exception or managing the exception using

```
S.o.pln("Out of try-catch Block");
S.o.pln("main ends");
}
}
```

⇒ P-I (if b = 2) (any non-zero value)

```
main starts
try-block starts
Performing division
Division is 5.00
try-block ends
Out of try-catch blocks
main ends.
```

P-II (if b = 0)

```
main starts
try-block starts
Performing division
AE occurs: java.lang.AE
out of try-catch blocks
main ends.
```

- From above outputs, note that, if we handle unchecked exceptions by using try-catch blocks, then it assures handling of exception & hence, program/thread will not terminate

Note

- It is also possible to handle multiple exceptions in single catch block, we do this if multiple exceptions have same solution or message to do this we can use single pip operator (|) between exception classes.

Eg:-

```
try
{
    ≡
    ≡
}
catch (ArithmeticException | ArrayIndexOutOfBoundsException xy)
{
    S.o.pln("Exception occurs:"+xy);
}
```

- It is also possible to manage any checked exception from try-catch block simply call a function from try-block.

Eg:-

```
class Test
{ p.s.v.m(String arg[])
  {
    DIS dis = new DIS(System.in);
    int a,b,c;
    S.o.pln("Enter 2 values \n");
    try
    { a = I.parseInt(dis.readLine());
      b = I.parseInt(dis.readLine());
    }
```

```
try
{
   doubtful statement
   ==
}
catch (exception-class ref)
{
   Solution Statements
   ==
}
```

- To understand execution of try-catch block let us consider two possibilities

P1 If exception did not occur in try block
- then complete try block executes
- catch block gets skipped
- Program execution continues

P2 If exception occurs in try-block
- Controls will be transfered to catch block.
- Complete catch-block executes.
- Program execution continues.
- remaining statements of try-block gets skipped.

- From Both above possibilities note that bcoz of try-catch blocks, program execution did not get terminated even if exception occured

Strictly remember that this exception handling mechanism works if & only if exception occurs in try-block

Note. There are possibilities of numbers of exceptions in one program, in one try-block & also in single statement.
- It is possible that one statement may have possibility of occurance of multiple exception but only one I.e first exception makes effect.

5/5

Eg:-
```
class Test
{
  P.S.V. main(String args[])
  { int a=10, b=__;
    float c;
    S.o.pln("main starts");
    try
    { S.o.pln("try-block begins");
      S.o.pln("Performing division");
      c=a/b;
      S.o.pln("division is "+c);
      S.o.pln("try-block ends");
    }
    catch(ArithmeticException ae)
    {
      S.o.pln("AE occurs"+ae);
    }
```

To understand execution of above program let us consider two possibilities

P1 if b = 2 (Any non-zero value.)

main starts
Performing division
Division is : 5.00
main ends

P2 if b = 0

main starts
performing division
- Arithmetic Exception occurs prog/thread exception terminates.

Eg 2 class Test
{ p.s.v.m(—)
{
int a[] = {20, 30, 10, 40, 50};
int ind = —;
S.o.pln("main starts")
S.o.pln("Requested ele");
S.o.pln("is " + a[ind]);
S.o.pln("main ends");
}
}

· To understand execution of above prog. let us consider below two possibilities.

P1 (If ind = 2)
main starts
Requested ele
is 30
main ends

P2 (If ind = 10)
main starts
Requested ele
Exception occured "ArrayIndexOutOfBoundsException".

From Both above examples we can say if any unhandled exception occurs then program/thread execution terminates immediately this sudden termination will skip/cancel execution of other important statements which might be related to releasing resources or taking any important decision as a programer we cannot afford to skip/cancel execution of such statements which may lead to dead lock or any other resource /issue. Hence such exceptions/situations needs to be properly handled or manage by using "try-catch" blocks or keyword "throws"

try-catch block :- The try-catch block is the only actual exception handling mechanism to use try-catch block general syntax is

Out of all try-catch blocks
main ends

P-III If b=0

main starts
Enter 2 values
10 ↵
0 ↵
Cannot divide by zero.
Out of all try-catch blocks
main ends.

- Here also, in case of try with multiple catch blocks we can specify a catch with multiple exception classes seperated by Single pipe operator (|)

for ex:
```
try
{
    ==
    ==
    ==
}
catch(NumberFormatException | InputMismatch
        Exception ref)
{ S.o.pln("Invalid input is provided."+ref.
        getClass);
}
catch(AIOOBE | SIOOBE ref)
{
  S.o.pln("Invalid index is provided"+ref.getClass
}
```

Note Similar to above example, we can manage & handle many no's of exceptions in Single catch block, using single pipe operator (|) but the only condition is that those classes must not be in relation of Base class & derived class of each other.

Eg:-
```
try
{
    ==
}
catch(ArithmeticException | NumberFormatException
        Exception ref)
{ ==                          is error: already caught
  ==}
```

- Many Programmers have a habit/strategy of directly preparing catch-block with base class Exception.

Eg:-
```
try
{
    ==
    ==
}
catch(Exception exc)
{
    ==
}
```

- This habit/strategy is valid but not preferable. Always prefer Exception only when try with multiple catch blocks is mentioned/used. where their are possibilities of multiple exception

Page No

In catch-block the classes which are already caught are allowed to be caught or thrown.
- It is a compile time error.

- To understand execution of above try with multiple catch blocks consider two possibilities

P-I If exception did not occur in try-block then
- Complete try-block will execute.
- All try-blocks respective catch block will get skip.
- If one try-block with many catch then all catch-blocks gets skipped.
- Program execution continues.

P-II If exception occurs in try-block
- Control immediately transfers to related catch-block
- Complete catch-block executes
- Program execution continues
- Remaining try-block & other catch-blocks gets skipped.

For eg:-
```
class Test
{ p.s.v.m(String arg[])
  { int a,b,c;
    DIS dis = new DIS(System.in);
    S.o.pln("main starts");
    try
    { S.o.pln("Enter 2 values");
      a = I.parseInt(dis.readLine());
      b = I.parseInt(dis.readLine());
      c = a/b;
      S.o.pln("Division is "+c);
    }
    catch(IOException ioe)
    { S.o.pln("Exception:"+ioe.getClass());}
    catch(NumberFormatException nfe)
    { S.o.pln("Invalid input to parse");}
    catch(ArithmeticException ae)
    { S.o.pln("Cannot divide by zero");}

    S.o.pln("Out of all try-catch blocks");
    S.o.pln("main ends");
  }
}
```

12/5/23

To understand above program consider following possibilities

P-I If No exception Occurs

```
main starts
Enter 2 values
10 ↵
5 ↵
Division is 2
out of all try-catch blocks
main ends.
```

P-II If invalid input is provided

```
main starts
Enter 2 values
ABCD ↵
Invalid input to parse ← NFE
```

```
        catch(IOException ioe)
          { }
        c = a+b;
        S.o.pln("Addition is "+c);
    }
}
```

16/5/23

Refer following another example::

```
class Test
{
  p.s.v.m(String ar[])
  {
    S.o.pln("main starts");
    Test2.addition();
    S.o.pln("main ends");
  }
}

class Test2
{ public static void addition()
  { int a,b,c;
    DIS dis = new DIS(System.in);
    S.o.pln("Enter 2 values");
    try
    { a = I.parseInt(dis.readLine());
      b = I.parseInt(dis.readLine());
    } catch(IOException ioe)
      { }
      c = a+b;
    S.o.pln("Addition is "+c);
  }
}
```



```
}
```

. From above example note that checked excepⁿ IOE is handled by using try-catch block therefore addition() did not thrown IOE and therefore its calling function main() also did not thrown or handled IOException exception.

Try with multiple catch :- It is also possible to define a try-block with multiple catch blocks. We do this if a try-block contains possibilities of occurance of multiple exceptions.

. To use try with multiple catch blocks, general syntax is

```
try
{
  doubtful statements;
}
catch(exception-class-I ref)
{ solⁿ for exception-class-I }
catch(exception-class-II ref)
{
  solⁿ statement for exception-class-II }
catch(exception-class-n ref)
{
  solⁿ statement for exception-class-n
}
```

Saathi

Date ___/___/_____

- To understand execution of try-multiple catch-finally block consider following possibilities.

P-I If exception did not occur in try-block.

1
- Complete try-block executes
- All catch-blocks gets skipped
- Complete finally block executes
- Program execution continues.

P-II If handled exception occurs in try-block.

- Controls immediately transfers to related catch-block
- Complete related catch-block executes.
- Complete finally block-executes
- Program execution continues.
- Remaining catch-blocks gets skipped.
- Remaining try-blocks gets skipped.

P-III If unhandled exception occurs in try-block.

- Program terminates immediately.
- But before termination finally block executes.

3] try-finally block:- This is not considered as exception handling because there is no catch block is not present.
- Therefore, every exception is considered unhandled.

Saathi

Date ___/___/_____

To use try-finally block general syntax is

=>
```
try
{
    doubtful statements
}
finally
{
    mandatory statements
}
```

- To understand execution of try-finally block considered following possibilities.

P-I If exception did not occur in try-block.

- Complete try-block executes
- Complete finally-block executes
- Program execution continues.

P-II If exception occurs in try-block (In this every exception is unhandled)

- Program execution terminates immediately.
- But before termination complete finally block executes.

```
catch (exception-class ref)
{
  Solution statements
  ==
}
finally
{
  mandatory statements
  ==
}
```

- To understand execution of try-catch-finally block consider following possibilities

P-I If exception did not occur in try-block.

- Complete try-block executes.
- Catch-block gets skipped
- Complete finally block executes
- Program execution continues

P-II If handled exception (who's catch block is present.) occurs in try-block.

- Controls immediately transfers to related catch block.
- Catch-block executes completely.
- Complete finally block executes
- Program execution continues
- Remaining try-block is skipped.

P-III If unhandled exception occurs in try-block

- Program terminates immediately
- But before terminating finally block executes.

2) Try-multiple catch-finally:- To use try-multiple catch-finally block general syntax is

⇒
```
try
{
  doubtful statements
  ==
}
catch (AE ref)
{ Soln.
  ==
}
catch (exception-class II ref)
{ Soln
  ==
}
catch (exception-class III ref)
{ Soln
  ==
}
finally
{
  mandatory statements
  ==
}
```

in try-block along with possibilities of any certain/unknown exception situation. Then this catch-block with class Exception is suitable. And most importantly such catch block must be last catch-block. Because catch block also follows top-down priority of exceptions.

18/5/23

Eg:-
```
try
{
    ≡
}
catch (ArithmeticException ae)
{
    ≡
}
catch (IOException ioe)
{
    ≡
}
catch (RuntimeException re)   ← is B.C of all unchecked exception.
{
    ≡
}
catch (Exception e)
{
    ≡
}
catch (Throwable t)
{
    ≡
}
```

Keyword "finally" or finally block.

. We know that if exception occurs in try-block, then controls transfers to related catch-block. Because of this remaining statements of try-block gets skipped. And if their are some important statements that we cannot afford to skip, then those statements should written in "finally-block"

*. Remember that, finally block executes whether exception occurs or not & whether exception is handled or unhandled.

*. The only point is, finally-block executes if & only if controls reaches to try-block. In other words if exception occurs before try-block then finally-block will not execute

. To use finally-block, their are three valid combination

1] try-catch-finally
2] try-multiple catch-finally
3] try-finally

1] try-catch-finally :- To use try-catch-finally general syntax is

=>
```
try
{
    doubtful statements
    ≡
}
```

Page No.

• URL is related to ... of ... URL
• URL with values are called U.R.I (uniform Resource identifier).

and if thrown exception is of an unchecked exception class then it is optional for calling function to handle or throw same exception.

* It is also possible that a function can throw multiple exceptions by using keyword "throws" where list of exception classes are seperated by comma. For eg:-

```
public void check () throws IOException,
{                           UnknownHostException
    ≡                         ↓ in java.net
}                             (is checked exception)
```

In above example both thrown exceptions are checked exceptions if require, then we can also make a combination of checked, unchecked and user-defined exceptions.

Eg:-

```
public void check () throws IOException, (checked)
{                   NULLPointerException, (unchecked)
    ≡               NumberTooSmallException (user def)
}
```

Now the calling function of above check() has two options

1) calls check() from try-block.
2) The calling function must throw same exception.

* : For any thrown exception object either we need reference of same exception class or reference of its base class and this is the reason why many programmers prefer catch-block with class "Exception".

Some possibilities of exceptions in try-catch-finally blocks.

1)

```
p. void check ()
{
    ≡  part A
  try
  {
    ≡
  }
  catch (exception-class ref)
  {
    ≡
  }
  finally
  {
    ≡
  }
    ≡  part B
}
```

P-I If exception occurs in part A
- program/thread terminates immediately. (Not execution of finally block)

P-II If exception occurs in catch-block.
- program terminates immediately but finally executes

Saathi

Date __/__/____

- For user defined exception class we have two options

1) To make it as checked exception class inherit/extend class exception or any other checked exception class.
2) To make it as unchecked exception class inherit/extend class RuntimeException or any other unchecked exception class.

Eg:- If we want to create a user-defined exception class for above multiplication situation then we can do it as

- class NumberTooSmallException extends Exception or any checked

{
==
}

- class NumberTooSmallException extends RuntimeException or any unchecked exception.

{
==
}

21/5/2023

Difference between Keyword "throw" & "throws".

Saathi

Date __/__/____

	"throw"	"throws"
1)	Keyword "throw" is prefered to throw user defined/manual exception.	1) Keyword "throws" is prefered to throw exceptn object from called to calling function.
2)	Keyword "throw" is exactly suitable to throw only one exception object.	2) Keyword "throws" is suitable to through possibility of multiple exception objects. but will throw only one exception object.
3)	Keyword "throw" is used within the body of a function	3) Keyword "throws" is used with the signature of the funct^n

* Keyword "throws" * (use of Keyword throws)

Keyword "throws" is another way of exception handling. By using Keyword "throws" with function defination we are able to throw object of occured exception class back to the body of calling function.

Eg:-

```
public void checK() throws exception-class-name
{
==
}
```

In above example if thrown exception is of a checked exception class, then it becomes mandatory for calling function to handle or throw same exept

Saathi

Date __/__/____

20/5/23

* Keyword "throw" OR User defined Exception OR Coustom Exception.

. We Know that every exception class is linked with same predefined situation of occurance when ever that situation occurs, Java Runtime system creates an object of respective exception class (fill it with exception metadata) and send it to program statement from where we decide either to catch or to throw that exception.

. If logically required, then we can also decide a user defined / custom exception situation and manually throw / send any exception object to catch block. To do this we have Keyword "throw"

. Keyword "throw" must be followed with exception class object which we want to throw / send from try-block to catch-block. Refer following program where we multiply three float values. If any one value is less than 0.5 then we will throw a custom exception to catch-block.

→ class Test
{
  p.s.v.m(String a[])
  {
    float a,b,c;
    Scanner sc = new Scanner(System.in);
    S.o.pln("main starts");

Saathi

Date __/__/____

    try
    { S.o.pln("Enter 2 float values");
      a = Sc.nextFloat();
      b = Sc.nextFloat();
      if(a<0.5 || b<0.5)
        throw new ArithmeticException();
      c = a*b;
      S.o.pln("Multiplication is"+c);
    }
    Catch(ArithmeticException ae)
    {
      S.o.pln("Number too small to multiply");
    }
    S.o.pln("Out of try-catch blocks");
    S.o.pln("main ends");
  }

Note In above example the condition of if statement is called as custom exception. When we want to throw any exception. Also note that we have declared and thrown object of class ArithmeticException. but in such case of custom exception we can declare & throw object of any existing exception class.

. In case of any flexible customized messages or in case of providing any readymade solution. We can also define our ot user defined exception classes by simply taking our class in hierarchy of class Throwable

P-V If N.F.E occurs in inner catch block.
- inner finally-block executes
- outer catch block executes completely
- outer finally block executes.
- program execution continues

P-VI If N.F.E occurs in inner finally block.
- control gets transfered to outer catch block.
- Complete outer catch block executes.
- outer finally-block executes.
- program execution continues

4)
```
try
{
    == (part A)
    try
    { ==
    }
    finally
    { ==
    }
    == (part B)
}
finally
{ ==
}
```


P-I If exception did not occur in outer try-block.
- part-A executes
- inner try-block executes
- inner finally executes and part-B executes
- outer finally executes.
- program execution continues.

P-II If exception occurs in Part A.
- program execution terminates but before outer finally executes.

P-III If exception occurs in inner-try block
- inner finally executes
- part-B ~~exec~~ gets skipped.
- outer finally executes.
- prog. execution terminates.

P-IV If exception occurs in inner finally-block.
- part-B gets skipped
- outer finally-block executes.

P-V If exception occurs in part B
- outer finally block executes.
- Program terminates.

P-VI If A.E exception occurs in outer catch-block
- prog. terminates immediately.
- outer finally block executes.

3)
```
try
{
    == (part-A)
    try
    {
    ==
    }
    catch(ArithmeticException ae)
    {
    ==
    }
    finally
    {
    ==
    }
    == (part-B)
}
catch(NumberFormatException nfe)
{
    ==
}
finally
{
    ==
}
```

P-I If exception did not occur in outer try-block.
- part-A executes
- inner-try-block executes
- inner-catch-block gets skipped
- inner finally block executes
- part-B executes.
- Outer-catch-block gets skipped
- Outer-finally-block gets executes
- program execution continues.

P-II If unhandled exception occurs in part-A
- program execution terminates immediately but before it outer-finally-block executes.

P-III If A.E occurs in inner-try block.
- controls immediately transfers to inner catch-block.
- inner catch-block executes.
- inner finally block executes.
- part B executes.
- outer catch block gets skipped
- Outer finally block executes
- program execution continues.

P-IV If NFE occurs in inner-try block.
- inner finally executes.
- Outer catch block executes.
- outer finally block executes.
- program execution continues.

∴ part-B gets skipped.

P-III If exception occurs in finally-block.
- ∴ program terminates immediately.
- part B gets skipped.

P-IV If exception occurs in Part-B
- program terminates immediately
- remaining lines gets skipped.

2)
```
try
{
    == (part A)
    try
    {
      ==
      ==
    }
    catch(ArithmeticException ae)
    {
      ==
      ==
    }
    == (part B)
}
catch(NumberFormatException nfe)
{
  ==
  ==
}
finally
{
  ==
  ==
}
```


P-I If No exception occurs in outer-try-block.
- part A executes
- inner try-block executes
- inner catch-block gets skipped
- part B executes
- outer catch block gets skipped
- finally-block executes
- program execution continues.

P-II If A.E occurs in Part-A
- program execution terminates immediately but before termination finally block executes.

P-III If N.F.E occurs in part-A
- ∴ Controls directly transfers to outer-catch block.
- Complete outer-catch block executes.
- Outer finally block executes.
- program execution continues.
- remaining inner-try-catch-part-B gets skipped.

P-IV If N.F.E occurs in inner-try-block
- Controls directly transfer to outer-catch-block
- Complete catch block executes (outer).
- Complete finally block executes.
- Remaining try block statements gets skipped.

P-V If N.F.E occurs in inner catch-block
- Controls directly transfers to outer catch block
- Complete outer catch block executes
- Complete finally block executes.
- Remaining try block gets skipped.