

Core Java (Java SE)

First Program and Main method

First.java

Everything is inside class

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

Eg : nokia = connecting people

Variable is a storage where we store values

Application = share the information digitally

The way of communication

Its all about data

You work with the data

Saving and retrieving is easy

Processing takes time

JShell (shell prompt)

It is basically a repl ->read evaluate print loop

| Welcome to JShell -- Version 17.0.6

| For an introduction type: /help intro

```
jshell> 2+3
$1 ==> 5
```

```
jshell> 9*4
$2 ==> 36
```

```
jshell> clear
```

```
| Error:  
| cannot find symbol  
|   symbol: variable clear  
| clear  
| ^__^
```

```
jshell> $1+$2  
$3 ==> 41
```

```
jshell> print("hello")  
| Error:  
| cannot find symbol  
|   symbol: method print(java.lang.String)  
| print("hello")  
| ^__^
```

```
jshell> System.out.print("hello")  
hello
```

```
jshell> /  
| Command: '/' is ambiguous: /list, /edit, /drop, /save, /open, /vars, /methods, /types, /imports,  
/exit, /env, /reset, /reload, /history, /debug, /help, /set, /?, /!  
| Type /help for help.
```

```
jshell> /vars  
| int $1 = 5  
| int $2 = 36  
| int $3 = 41
```

Statement when it is characters then enclose it in double quotes[NORMAL TEXT]

=purpose jshell is to check small code snippets

TEXT EDITOR vs IDE's

Vs code extensions
=>extension pack of java
=>code runner

```
System.out.print("Hello World");
```

Or

```
//signature of main method  
public static void main(String args[])
```

```
{  
System.out.print("Hello World");  
}
```

Output:

```
PS C:\Users\siddu\OneDrive\Desktop\JAVA\codes> cd  
"c:\Users\siddu\OneDrive\Desktop\JAVA\codes\" ; if ($?) { javac FirstCode.java } ; if ($?) { java  
FirstCode }  
FirstCode.java:1: error: class, interface, enum, or record expected  
System.out.print("Hello World");  
^  
1 error
```

In Java main() has specific syntax to follow

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

Output:

```
if ($?) { javac FirstCode.java } ; if ($?) { java FirstCode }  
Hello World
```

FirstCode.java

```
|  
| javac (compiler)  
|  
| FirstCode.class (byte code)  
|  
| java (interpreter) byte codes are executed by jvm  
|  
Execution
```

To compile in terminal or cmd >javac <FILE_NAME with extension>

To run >java <CLASS_FILE_NAME without extension>

To see code in terminal or cmd

⇒ to client we give only class files not java code

⇒ when the class access specifier is public then main method class name in program has to same as of file name.

⇒ every class in code create that many number of class files.

.jar = consists of lot of class files

(eg: 100java files contains 600class then it creates 600 class files that can be bundled as 1 jar file)

Windows os = .exe

Mac os = .pkg

Linux os = .deb

Java = .jar

Every program starts with main method

App/Software/program

We are write programs of java SE that are console based

METHOD (and function is same)

=task or work or activity to perform using programming we use methods

Method will have

1.name

2.parameters

3.body

4.return type

```
method_name(parameter_inputs)
{
//stmts task|body
}
```

=====

Calling method

```
int add()
{
    return 1+3;
}
```

=====

Caller

```
add();
=====
```

Jvm is coded in such a way that if program contains main method then only program will execute

Why static ?

To accessible to jvm

Without object creation of the class, can use it.

Java is well structured and oop language

main is the name of the method and starting point to jvm

Void is return type

String args[] is parameter cmd line arguments

Public is for visibility

Static if for accessibility without obj creation

Statically typed vs Dynamically typed

String args[] is to receive cmd line arguments

From cmd prompt we instruct or pass something to program so to take that as inputs we need cmdLineArguments

```
Class Launch
{
    public static void main(String args[])
    {
        System.out.println(args[0]);
    }
}
```

Behind the scene an array is created for String args[]
and it stores

Cmd prompt

```
>javac Launch.java
>java Launch HaiEveryone
```

Main method

Different syntax or signatures possible

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

Whatever data we give to program to store it, we have variables in program

Variable = variable is to store data or information

Statically typed (type of data has to be specified)

Variable declaration \Rightarrow int a = 10;

Syntax means how the language works like grammar to english.

Eclipse IDE

1. Start click on OPEN PERSPECTIVE
2. Close everything except Package Explorer in eclipse
3. Click File and create a new java project
4. Give a project name and click on next
5. Don't create module for it
6. finish

Expand project folder

Select src and create a class

cmdLineArguments in Eclipse

Eg: Demo.java

Right click on it ->Select "Run As" ->click "Run Configurations"

Select your project : you have to see there project name

Also select class name

In Tabs : select "Arguments" tab

And write cmd line argument

To see click "show command line"

Click Run

10)13-10-22

OOPs (basic Introduction)

(object orientation principles)

What are objects => real world things, entities , real time instances
Object means real time instance.

Every object in real time will have 2 parts

Eg: car

1.what it has?

eg:brandName

 noOfWheels

 Model

 speed

2.What it does?

 Move

 Accelerate

 break

JAVA CODE OF basic oops

```
class Car
{
    //Has part of an Obj is represented as a Variable
    String brandName;
    int noOfWheels;

    //Does part of an Obj is represented through “methods”
    public void move()
    {
        //logic of moving a vehicle
    }
    public void accelerate()
    {

    }
}

class Student
{
```

```
//Has part  
String name;  
int id;  
float height;  
  
//Does part  
public void Study(){}
public void Play(){}
}
```

Identifiers(means names in a java program)

Identifiers are classnames,variable names,method names

```
class Main
{
    public static void main(String args[])
    {
        Int x = 10;
    }
}
```

Find the no: of identifiers in above program?

5 identifiers

That are Main,main,String,args,x.

```
class Test{
    public static void main(String args[])
    {
        System.out.println("hello Ram")
    }
}
```

//total 7 identifiers

```

class Test
{
    public static void main(String[] args)
    {
        int x= 10;
    }
}

class Test
{
    public static void main(String[] args)
    {
        System.out.println("sachin");
    }
}

```

All reserved words are lower case

All the class names and interface names starts with capital letters

Rules(syntax of compiler and jvm) for writing an identifiers

Rule 1 : The only allowed characters in java identifiers are

A to Z , a to z , 0 to 9 , _ , \$

Rule 2 : if we use any other character it would result in invalidation

Or compilation error

Rule 3 : identifiers are not allowed to start with digits

Rule 4 : java identifiers are case sensitive

Rule 5 :There is no length limit on java identifiers , but still it is a good practice to keep the length of the identifier not more than 15 characters

Eg: int priorityOfThreadWithMinVallue = 1;

Rule 6 : Reserved words are not allowed to use as identifiers

Rule 7 : Predefined class names can be used as identifiers

NOTE : But not recommended

Integer ⇒ predefined classname

String ⇒ inbuilt classname

Runnable ⇒ interface name

Student ⇒ user defined name

class Test

```

{
    public static void main(String args[])
    {
        //predefined word
        int Integer = 10;
        System.out.println(Integer);
    }
}

```

Interview Questions

Int If = 10;
sop(IF); //10 if and If is different

int Integer = 10;
sop(Integer); //10 Integer is predefined

Reserved Words are inbuilt words/Keyword

Which has predefined meaning to it.
(=information already known to compiler about reserved words)

Totally 53 reserved words

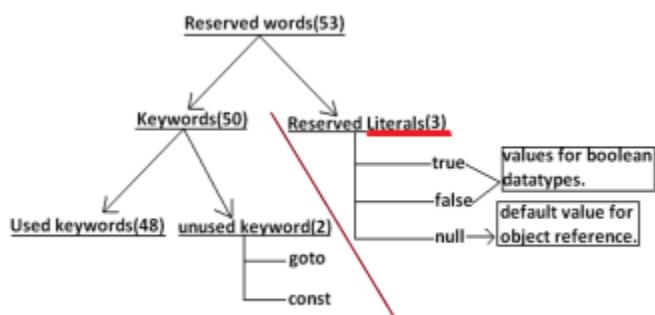
Keywords are 50

Reserved literals are 3

Used keywords are 48

Unused keywords are 2

(think like compiler and jvm)



Reserved words for data types: (8)

- 1) byte
- 2) short
- 3) int
- 4) long
- 5) float
- 6) double
- 7) char
- 8) boolean

Reserved words for flow control:(11)

- 1) if
- 2) else
- 3) switch
- 4) case
- 5) default
- 6) for
- 7) do
- 8) while
- 9) break
- 10) continue
- 11) return

OOP's in Java**Keywords for modifiers:(11)**

- 1) public
- 2) private
- 3) protected
- 4) static
- 5) final
- 6) abstract
- 7) synchronized
- 8) native
- 9) strictfp(1.2 version)
- 10) transient
- 11) volatile

Exception Handling**Keywords for exception handling:(6)**

- 1) try
- 2) catch
- 3) finally
- 4) throw
- 5) throws
- 6) assert(1.4 version)

void -> keyword associated with method
unused keywords: goto,constant

Class related keywords:(6)

- 1) class
- 2) package
- 3) import
- 4) extends
- 5) implements
- 6) interface

Object related keywords:(4)

- 1) new
- 2) instanceof
- 3) super
- 4) this

```
class Test
```

```
{
```

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

```
{
```

```
//data is identifier and variable it need type means what kind of data is //storing in it
```

literal = Any constant value which can be assigned to a variable.

```
//In java every statement is terminated with a semicolon.
```

```
    int data = 10;  
}  
}
```

```
Class Test
```

```
{
```

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

```
{
```

```
    boolean result = true;  
    boolean flag = false;
```

```
    boolean examResult = false;
```

```
}
```

```
}
```

NOTE: for boolean data type only allowed values are true and false. Other than this leads to compile time error

Q's

- 1.What is class?
- 2.What is variable and identifier ?
- 3.What are the rules associated with identifiers ?
- 4.What is the method ?
- 5.Can the main method be overloaded ? yes
- 6.What are reserved words ?

Which of the following list contain only reserved words /keywords / builtinwords?

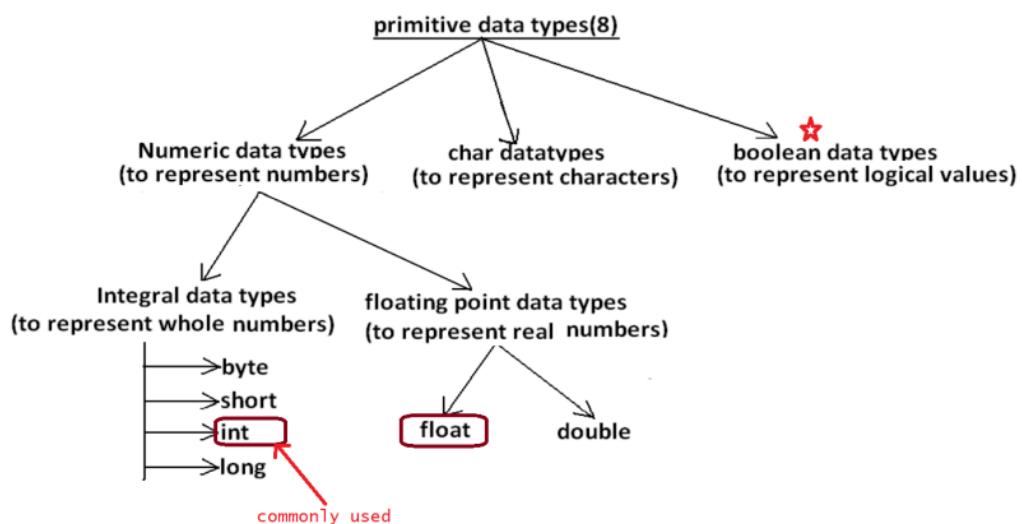
1. final,finally,finalize
ans. finalize is not a reserved word, it is a method in Object class.
2. break,continue,return,exit
ans. exit is not a reserved word, it is a method in System class
3. byte,short,Integer,long
ans. Integer is not a reserved word,it is a predefined class
4. throw,throws,thrown
ans. thrown is not a reserved word,it is a user defined variable.

Data Types

Every variable has a type,every expression has a type and all types are strictly typed/define in java

becoz java is a strictly typed /statically typed language.

Compiler role -> Compiler will check the value stored can be handled by datatype or not This checking which is done by compiler is called "TypeChecking/Strict Type checking".



Primitive data types

=====

meaning -> data which is commonly used and supported by any language to store directly.

a. Numeric values => to store number

a. whole number

b. real number

b. character values => to store character type of data

c. boolean values => to store logical value

```
public class DataTypeSize {
```

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

```
        System.out.println("size of byte :" + Byte.SIZE);
```

```
        System.out.println("Min size of byte :" + Byte.MIN_VALUE);
```

```
        System.out.println("Max size of byte :" + Byte.MAX_VALUE);
```

```
}
```

```
}
```

output

size of byte :8

Min size of byte :-128

Max size of byte :127

```
public class Test{
```

```
    public static void main(String[] arg){
```

```
        byte a = true;           //CE: incompatible type
```

```
        byte b = "siddu";       //CE: incompatible type
```

```
        byte c ='a';
```

```
}
```

Number data

To store whole numbers we have 4 datatypes

a. Byte

b. Short

c. int

d. long

datatype information like

a. size of datatype(how much memory is allocated on the ram for that datatype by JVM)

b. minvalue what it can keep c. maxvalue what it can keep

byte

note: System.out.println("Size of byte is :: "+Byte.SIZE); System.out.println("MINVALUE of byte is :: "+Byte.MIN_VALUE); System.out.println("MAXVALUE of byte is :: "+Byte.MAX_VALUE);

byte: size -> 8

bits minvalue -> -128 maxvalue -> 127

eg: byte marks=35 //valid byte marks = 135;

//CE: possible loss of precision

byte marks = -1;//valid

byte a = true;//CE: incompatible types

byte b = "nitin";//CE: incompatible types

When to use byte data type?

It is commonly used when we handle the data which is coming from a streamnetwork.

stream -> java.io package

" " -> means String data

' ' -> char data

short

System.out.println("Size of short is :: "+Short.SIZE); System.out.println("MINVALUE of short is :: "+Short.MIN_VALUE); System.out.println("MAXVALUE of short is :: "+Short.MAX_VALUE); size : 16 bits(2 byte)

minvalue: -32768

maxvalue: +32767

eg: short data=137;//valid short data = true;

//CE: incompatible types short data = "sachin";

//CE:incompatible types Note: This data is not at all used in java and this data type is best suited only if u have old processors like 8086.

int

System.out.println("Size of int is :: "+Integer.SIZE); System.out.println("MINVALUE of int is :: "+Integer.MIN_VALUE); System.out.println("MAXVALUE of intt is :: "+Integer.MAX_VALUE);

size: 32 bits(4 bytes) minvalue:-2147483648 maxvalue: 2147483647 eg: int data = 323445;

int result = true;//ce:incompatible types

int result ="pass";//ce:incompatible types

note: The most commonly used datatype for storing whole number is "int" only and by default if we specify any literal of number type compiler will try to keep it as "int" only, but we can keep either in short or byte also.

long

```
System.out.println("Size of long is :: "+Long.SIZE); System.out.println("MINVALUE of long is :: "+Long.MIN_VALUE); System.out.println("MAXVALUE of long is :: "+Long.MAX_VALUE); size: 64bits(8bytes)
```

minvalue:-9223372036854775808

maxvalue:9223372036854775807

Eg: long data = 10;

```
long data = 9223372036854775807 ;
```

If the data goes beyond the range of int, then to keep the data inside long data type we need to explicitly suffix the data with 'L' or 'l' otherwise it would result in "CompileTimeError".

eg:

```
long firstData=9223372036854775807;//CE  
long secodData=9223372036854775807;//CE  
long firstData=9223372036854775807L;  
long secondData=9223372036854775807l;  
long data = 10L;//now also no problem becoz 10 is int and we have added 'L'  
long number = 5l;
```

Note: When int is not enough to hold the big values, then we use long data type. When we work with large files, data would come to java program in terms of GB's.

```
long size = file.length();
```

boolean

Only allowed values are true and false

That are reserved literals

11)14-10-22

By default fractions are treated as double, for float suffix f is represented

Commonly used is float

Float

```
float firstData = 2334.344f;
```

```
System.out.println("Size of float : "+Float.SIZE);
System.out.println("
```

Double

To map primitive data as Object in java from JDK 1.5 concept of "Wrapper class" is introduced.

Char- 2 bytes of memory

=whenever to store any character or symbol char data type is used.
This data types follow specific format to store the literals or symbols

In memory every thing is stored in 0 and 1 s

Eg: A B @ # 1 3 all this to store in char has specific format.

American's discovered there are total 128 characters which can be used in applications.so they gave binary representation and hex decimal representation that is called ASCII(American standard code of information interchange)

C follows ASCII

JAVA follows Unicode.

In java we can use multiple languages

In entire world there are many languages that symbols and characters are listed and said the will be 65536

UTF-16 UNIVERSAL TRANSFORMATION FORMAT

<https://home.unicode.org/>

Memory always comes with standardization

After 128 they added the further symbols and characters

Create a table for english alphabets and decimal representation

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	\$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[END OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

SIZE OF DATA TYPES

Follows Base2 format

byte 1B

short 2B

Int 4B

long 8B

Follows IEEE format

float 4B

double 8B

Follows unicode format

char 2B

boolean ?B

Long can be stored in float,IEEE is efficient than Base2

Declaration and initialization of variables

```
int a;  
a = 10;
```

```
2|45  
2|22 1  
2|11 0
```

1101 is binary of 45

Every type data has there format,so that real world data stored in specific formats of it.

Long can be stored in float

Assignment gives right side value to left side

```
Byte a = 45;
```

```
Double b;
```

TypeCasting

=changing the type of data from one type to another type

If in happens internally then it is called as

Implicit type casting = numeric type promotion

TypeConversion (explicit type conversion)

```
Double a = 45.5;
```

```
Byte b = (byte)a;
```

=possibility of loss of data.

code snippets

```
Byte ab = 10;
```

```
Byte cd = 5;
```

```
Byte result = ab*cd; // int result = ab*cd;
```

```
System.out.println(result); //error
```

Q&A

Can we create object of interface?

Can we create an object of an abstract class ?

Operators

5+10 = 15; //special symbol + is used that is arithmetic operator
int a = 15 ;

1) Incrementation and Decrementation

int a = a+1;
Incrementing an existing value by 1 .
So , it can be a++

a-- //decrementation
Like wise we have
Pre and post

Pre incrementation

++a

Post incrementation

a++

Pre decrementation

--a

Post decrementation

a--

Expression

int a = 5;
int b = a++ + ++a - --a ;

1)CodeSnippets(JDK8 version)

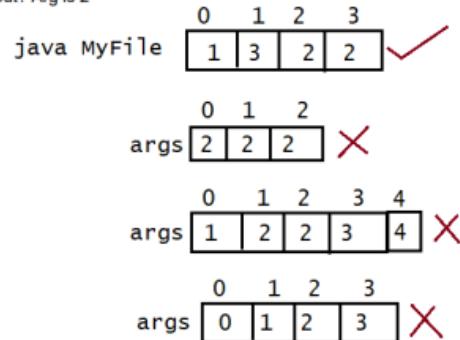
Given the code snippet from a compiled Java source file:

```
public class MyFile
{
    public static void main (String[] args)
    {
        String arg1 = args[1]; //3
        String arg2 = args[2]; //2
        String arg3 = args[3]; //2
        System.out.println("Arg is " + arg3);
    }
}
```

Arg is 2
Arg is 2

Which command-line arguments should you pass to the program to obtain the following output? Arg is 2

- A. java MyFile 1 3 2 2
- B. java MyFile 2 2 2
- C. java MyFile 1 2 2 3 4
- D. java MyFile 0 1 2 3



For the code below, what should be the name of java file?

1.

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

- A. Hello.java
- B. World.java
- C. HelloWorld.java
- D. helloworld.java

2.

Does below code compile successfully?

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

- A. yes
- B. no

Ans: A

In java every statement should be terminated with ; symbol

Note: ; means ending.

3.

What is the signature of special main method?

- A. public static void main(String args) //[] is missing
- B. public static void main(String[] a) //correct
- C. public static void main()//missing String[] args
- D. private static void main(String[] args)// private can't be for a main method

4.

What will be the result of compiling and executing Test class?

java Test good morning everyone

```
private class Test{  
    public static void main(String args[]) {  
        System.out.println(args[1]);  
    }  
}
```

- A. compilation error //valid answer becoz the main method class can't be private
- B. good
- C. morning
- D. everyone

5.

For the class Test, which options, if used to replace /*INSERT*/, will print

"Hurrah! I passed..." on to the console? Select 2 options.

```
public class Test {  
    /*INSERT*/ {  
        System.out.println("Hurrah! I passed...");  
    }  
}
```

- A. static public void main(String[] args) // static and public can be interchanged so valid
- B. public static void main(String[] a)//valid
- C. static public void Main(String[] args)
- D. public void main(String[] args)
- E. protected static void main(String[] args)
- F. public void static main(String[] args)//here static and void is interchanged is not possible

6.

Suppose you have created a java file, "[MyClass.java](#)".

Which of the following commands will compile the java file?

- A. javac MyClass //invalid becoz .java is missing
- B. java MyClass // javac is command is used for compilation
- C. javac MyClass.class //invalid becoz .java is missing
- D. javac [MyClass.java](#) //valid

E. java [MyClass.java](#) //javac is the command for compilation and java is command for execution

Check

```
String [] args  
java MyClass.java → auto compilation and execution where .class file not be generated from  
JDK11Version
```

```
Float a = 12.3f;  
int b = 12.0f;
```

12)15-10-22 dbt

Editor -> editplus / notepad

Ide -> using Eclipse , later we move to STS(Spring tool Suite)

```
byte a = 10;  
a = a+1;// CE : possible loss of precession  
System.out.println(a);
```

```
byte a =10;//JVM will give memory fo 1 byte and data 10 is stored  
a++;//no problem for compiler //a = (byte)(a+1) ; // 11 will be stored as a byte by the JVM  
System.out.println(a);
```

```
check  
byte a = 10;  
a+=1;//short hand operation ⇒ a = a+1;  
System.out.println(a);
```

```
class Test  
{  
}
```

Ans: above code gets compiled , but doesn't run , it leads to exception at runtime because the main method does not exist.

support "Internationalization"

Internationalization -> Supporting the application to the end users as per there country regional language.

13)17-10-22

Get trying and give it practice.

Practice practice a lot.

Addition ,subtraction , multiplication , div programming lang's provides operator

In JAVA, operators

1. Increment and Decrement Operator
2. Arithmetic Operator
3. Assignment Operator (single assignment , chained assignment , compound assignment)
4. Relational Operator
5. Logical Operator
6. Conditional Operator (Ternary operator)
7. Bitwise and Shift Operator

2)Arithmetic Operator

+ - * / % (modulo operator)

Modulo operator gives reminders

```
package in.sid.main;

public class ArithmeticOperator {

    public static void main(String[] args) {
        int a = 20;
        int b = 40;
        int res = b/a;
        System.out.println("mul of b/a is "+res);
    }
}
```

3)Assignment Operator (single assignment , chained assignment , compound assignment)

=

Single assignment

```
int a = 10;  
Rightside value is assigned to leftside variable always.
```

Chained assignment

```
int a;  
int b;  
int c;  
int d;  
All has same value then  
int a,b,c,d;  
a=b=c=d=10; //chained assignment
```

Compound assignment

While assigning operation is happening
int a=10;
a+=5; //a = a+5; a=15;

4) Relational Operators (comparsion)

> < >== <== == !=

Return result as only true or false

5) Logical Operators

	&&		!
AND \Rightarrow	&&		
Exp1	exp 2	exp 3	res
T	T	T	T
T	F	F	F
T	F	T	F
F	F	F	F

T&&T&&T = T
T&&F&&T = F

If and only if all are true then only true,
Ever one is false then result is false

OR $\Rightarrow \parallel$

Exp1	exp 2	exp 3	res
T	T	T	T
T	F	F	T
T	F	T	T
F	F	F	F

Even one is true then result is true

$T \parallel T = T$

$T \parallel F \parallel T = T$

$F \parallel F = F$

NOT $\Rightarrow !$

$!F = T$

$!T = F$

UNARY OPERATOR

One operand is sufficient to perform operations

Only ASSIGNMENT , INCREMENT AND DECREMENT OPERATOR ARE UNARY OPERATORS, all other operators are binary

```
int a = 10;  
a++;  
a==2;  
a+=2;  
(dbt a!=a check)  
Check  
boolean e = true;  
    e=!e; // is it also a unary operator ?  
    System.out.println(e);
```

BINARY OPERATOR

More than one operand is required

6) Conditional Operator (Ternary operator)

? :

Means it acts as if-else block

```
int a = 10;  
int b = 20;
```

```
if(a>b)  
{
```

```
        int res = a-b;
        System.out.println(res);
    }
else
{
    int res = b-a;
    System.out.println(res);
}
```

Output \Rightarrow 10

//any one block will be executed

Write a java program to find least num in three numbers

```
if(a<b)
{
    if(a<c)
    {
        System.out.println("A is the least "+a);
    }
    else
    {
        System.out.println("C is the least "+c);
    }
}
else if(b<c)
{
    System.out.println("B is the least "+b);
}
else
{
    System.out.println("C is the least "+c);
}
```

+ Means concatenation in java,

This entire code can be written in one line

Using ternary operator

```
int res = (a<b)?(a<c?a:c):(b<c?b:c);
```

Loops \Rightarrow pattern programming \rightarrow loops + conditionals + operators

7)Bitwise and Shift Operator

Switch Case

```
switch(condition){  
    case 1 : stmt ;  
        break;  
    case 2 : stmt ;  
        break;  
  
    default : stmt ;  
}
```

Without break , fall through will happen if any of the above default cases is matched.

```
package in.sid.main;  
  
public class SwitchCase {  
  
    public static void main(String[] args) {  
        int number =100;  
  
        switch(number)  
        {  
            case 10: System.out.println("1st case");  
                break;  
            case 20: System.out.println("2nd case");  
                break;  
            case 100: System.out.println("3rd case");  
                break;  
            default: System.out.println("default is executed");  
        }  
    }  
}
```

2)CodeSnippets

2:41:00

7.

Given code of Test.java file:

```
public class Test {  
    public static void main(String[] args){  
        args[1] = "Day!";  
        System.out.println(args[0] + " " + args[1]);  
    }  
}
```

And the commands:

```
javac Test.java
```

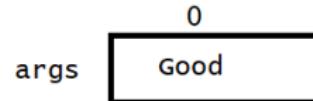
```
java Test Good
```

What is the result?

- A. Good
- B. Good Day!
- C. Compilation Error
- D. JVM would create a problem during execution

answer : D

```
java Test Good  
         ↓  
args[1] = "Day!";  "Exception"  
  
System.out.println(args[0] + " " + args[1]);
```



Consider below code of Test.java file:

```
public class Test {  
    public static void main(String[] args){  
        System.out.println("Welcome " + args[0] + "!");  
    }  
}
```

And the commands:

```
javac Test.java
```

```
java Test "James Gosling" "Bill Joy"
```

What is the result?

- A. Welcome James Gosling!
- B. Welcome Bill Joy!
- C. Welcome "James Gosling"!
- D. Welcome "Bill Joy"!
- E. Welcome James!
- F. Welcome Gosling!
- G. Welcome Bill!
- H. Welcome Joy!

answer : A

```
java Test "James Gosling" "Bill Joy"  
System.out.println("Welcome " + args[0] +"!");
```

args	0	1
	James Gosling	Bill Joy

10.

Consider below code of Test.java file:

```
public class Test {  
    public static void main(String[] args) {  
        boolean b1 = 0;  
        boolean b2 = 1;  
        System.out.println(b1 + b2);  
    }  
}
```

What is the result of compiling and executing Test class?

- A. 0
- B. 1
- C. true
- D. false
- E. compilation error

answer : E

11.

Given:

```
35. String #name = "Jane Doe";  
36. int $age = 24;  
37. Double _height = 123.5;  
38. double ~temp = 37.5;
```

Which two statements are true? (Choose two.)

- A. Line 35 will not compile.
- B. Line 36 will not compile.
- C. Line 37 will not compile.
- D. Line 38 will not compile.

answer : A,D

12.

What will be the result of compiling and executing Test class?

```
public class Test {  
    public static void main(String[] args){  
        byte b1 = ( byte ) ( 127 + 21 ); // byte b1 = (byte)(148)  
        System.out.println(b1);  
    }  
}
```

}

A. 148

B. Compilation Error

C. -108

D. -128

JVM : minrange + (result-maxrange-1)

= -128 + (148 - 127-1)

= -128 +(148-128)

= -128 +(20)

= -108

13.

Consider below code of Test.java file:

```
public class Test {  
    public static void main(String[] args) {  
        char c1 = 'a'; //ASCII code of 'a' is 97  
        int i1 = c1; //Line n1 // char----> int (implicit typecasting) jvm only //understands it internally  
        System.out.println(i1); //Line n2  
    }  
}
```

What is the result of compiling and executing Test class?

- A. a
- B. 97
- C. Line n1 causes compilation failure
- D. Line n2 causes runtime error.

answer : B

14.

Given code of Test.java file:

```
public class Test {  
    public static void main(String[] args) {  
        byte b1 = 10; //Line n1  
        int i1 = b1; //Line n2 Compiler : byte----> int(Implicit type casting)  
        byte b2 = i1; //Line n3 Compiler : int-----> byte (hey not possible u  
        explicitly tell)  
        System.out.println(b1 + i1 + b2);  
    }  
}
```

What is the result of compiling and executing Test class?

- A. Line n1 causes compilation error
- B. Line n2 causes compilation error.
- C. Line n3 causes compilation error.
- D. 30 printed on the console.

answer : C

15.

For the given code what is the output?

```
int x=100;  
int a=x++; // a = 100, x = 101  
int b= ++x; // b = 102, x = 102  
int c= x++; // c = 102, x = 103  
int d= (a<b) ? (a<c) ? a: (b<c)? b: c :x;  
System.out.println(d); //100
```

true true value
//int d= (100<102)?(100<102):100
//if false it will execute x

- A. 100
- B. 101
- C. 102
- D. 103
- E. compilation fails

answer: A

16.

```
class Test  
{  
public static void main(String[] args)  
{  
int a=100; // a= 10-  
System.out.println(-a++); //System.out.println(-100); now a = 101  
}  
}
```

- A. -101
- B. 99
- c. Compilation error
- d. -100
- e. -99

answer : d

fallthrough in switch

```
-----  
int a = 97;  
switch(a){  
case 97: System.out.println("hello");  
case 98: System.out.println("hiee");  
}
```

```
output: hello
hiee
since there is no break automatically control executed the next case also,this
condition in java under switch we call as
"fallthrough".
Sir Please You Can Explain For What We Use = a"""+b"""+c"""
int a= 10,b=20,c=30;
System.out.println(a + " " + b + " " + c);//10 20 30
```

Importance of cmd line arguments

```
javac Test.java
java Test ind.txt
java Test aus.txt
public class Test{
    public static void main(String args[]){
        //file name will be supplied from the command line arguments
        //code written to open the file and read the contents from the file
    }
}
```

ind.txt

rahul

rohit

kohli

aus.txt

warner

finch

smith

```
Char c = 'a';
System.out.println(++c + ++c);
```

Output is 195

```
check
char c = "123".charAt(1);
System.out.println(c++ + ++c);
>javap java.lang.String //to check method present in particular class
```

“ ” are discarded during execution, technically called as coating.

14)18-10-22

Eclipse and sts are free IDE'S

System.out.print() ⇒ prints in same line

System.out.println() ⇒ prints in new line

Loops

for()

```
public class VerticalPrintingStar {  
  
    public static void main(String[] args) {  
        //repeated task  
        //use loops  
  
        //manually  
        System.out.println("manually");  
        System.out.println("*");  
        System.out.println("*");  
        System.out.println("*");  
        System.out.println("*");  
  
        System.out.println();  
        //for loop  
        //for(initialization;condition;update)  
        //{
/>        // body  
        //i is dynamic declaration  
        System.out.println("using for loop");  
        for(int i=0;i<5;i++)  
        {  
            System.out.println("*");  
        }  
    }  
}
```

```
int n=5;
for(i=0;i<n;i++)
    System.out.println("*");
```

while()

⇒ as long as

As long as data is present in db , extract it.

```
int n = 5;
int i = 0;
while(i<n)
{
    System.out.println("*");
    i++;
}
```

do-while()

⇒ Executes at least once even if conditional fails

```
int i = 0;
do
{
    System.out.println("*");
    i++;
}i<n);
```

NOTE : initialization and updation is optional

for-each(enhanced for loop)

Pattern programming

```
public class FrameSquarePattern {

    public static void main(String[] args) {
        int n=10;
        for(int i=0;i<n;i++)
```

```

{
for(int j=0;j<n;j++)
{
    if(i==0||i==n-1||j==0||j==n-1)
        System.out.print("*");
    else
        System.out.print (" ");
}
System.out.println();
}
=====
```

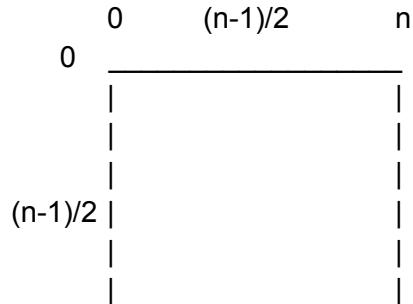
```

public class Hpattern {

    public static void main(String[] args) {
        int n=10;
        for(int i=0;i<n;i++)
        {
            for(int j=0;j<n;j++)
            {
                if(j==0||j==n-1||i==(n-1)/2)
                    System.out.print("*");
                else
                    System.out.print(" ");
            }
            System.out.println();
        }
    }
}
```

```
=====
```

Remember the box model



n |_____|

=====
(j==0&&i!=0)|| (i==0&&j>0&&j<(n-1)/2)

A pattern

3)CodeSnippets

Q>

Consider below code:

```
public class Test {  
    public static void main(String[] args) {  
        char c = 'Z';  
        long l = 100_00l;//from JDK1.7 for a literal we can give _ also, if we give  
        compiler will remove that _ in .class file  
        int i = 9_2;//from JDK1.7 for a literal we can give _ also, if we give  
        compiler will remove that _ in .class file  
        float f = 2.02f;  
        double d = 10_0.35d;//from JDK1.7 for a literal we can give _ also, if we  
        give compiler will remove that _ in .class file  
        l = c + i;//char + int = int int ----> long (implicit)  
        f = c * l * i * f;//char * long*int*float = float  
        f = l + i + c;//long+int+char = long long--> float(implicit)  
        i = (int)d;//double----> int(explicit)  
        f = (long)d;//double---> long , long ---> float (implicit)  
    }  
}
```

Does above code compile successfully?

A. Yes

B. No

Answer : A

Give practical values and check

Q>

```
class Test  
{  
    public static void main(String[] args)  
    {  
        int a = 20; // a = 18  
        int var= --a * a++ + a-- - -a;// var = 19 * 19 + 20 -18 = 363  
        System.out.println("a = " + a);// a = 18  
        System.out.println("var = " + var);//var= 363  
    }  
}
```

```
}
```

A.
a = 18
var=363

B.
a = 363
var=363

C. Compilation Error

D.
a = 25
var= 363

answer : A

Q>

```
class Test
{
    public static void main(String[] args)
    {
        int i = 5; // i = 5,6,7
        //5 < 6(true)
        if (i++ < 6) //evaluation
        {
            System.out.println(i++); //System.out.println(6)
        }
    }
}
```

A. 5

B. 6

C. Program executes successfully but nothing is printed on to console

D. 7

Ans: B

Q>

```
int x = 4;//line-n1
int y = 4++;//line-n2 whether it is post or pre-increment it can only be done on variables not on
direct literals
System.out.println(x);
System.out.println(y);

```

A. line-n1 CompileTimeError

B. x=4

y=5

C. x=5

y=5

D. line-n2 CompileTimeError

Answer: D

Q>
24
int x = 4;//line-n1
int y =++(++x);//line-n2 whether it is post or pre-increment it can only be done
on variables not on direct literals
System.out.println(x);
System.out.println(y);
A. line-n1 CompileTimeError
B. x=4
y=5
C. x=5
y=5
D. line-n2 CompileTimeError
Answer: D

Q>
boolean b=true;//line -n 1
b++;//line-n2 Ans. increment and decrement is applicable only for
integral,floating type and character type not for boolean type
System.out.println(b);
A. line-n1 Compile Time Error
B. line-n2 Compile Time Error
C. false
D. true
E. None of the above
Ans : B

Q>
int b,c,d;//declaring the variables

int a= b = c = d=10;//initializing the values for b=c=d and finally giving the
value to a variable with declaration

Will the code compile?

- A. yes
- B. no

Answer: A

Q>
int a = b = c = d = 20;//b,c,d not declared but using so CompileTime Error
System.out.println(a);
Will the code compile?
A. yes

B. no

Answer : B

Q>

byte c = (10> 20) ? 30 : 40;//literals are involved so compiler performs operation

byte c =40;

byte d =(10<20) ? 30 : 40;//literals are involved so compiler performs operation

byte d =30;

System.out.println(c);//40

System.out.println(d);//30

A. 30

40

B. 40

30

C. 10

20

D. 20

10

E. CompletetimeError

Answer : B

Q>

int a = 10, b = 20;//type checking is valid no problem

byte c = (a>b) ? 30 : 40;//literals are not involved in operation, so compiler would just check type checking of result

Compiler will see 30,40 type it knows is int,

so the result should be of int type only.

if compiler only perform the operation it will

try to map with casting chart otherwise it wants the exact type.

byte d =(a<b) ? 30 : 40;

System.out.println(c);

System.out.println(d);

A. 30

- 40
B. 40
30
C. 10
20
D. 20
10
E. CompiletimeError

Answer: E

Q>

Consider below statements:

1. int x = 5____0;// Literal values can be with _ also but it should be in b/w not at the beginning or at the end
2. int y = ____50;//invalid becoz starts with _
3. int z = 50____;//invalid becoz it ends with _
4. float f = 123.76_86f;//valid
5. double d = 1_2_3_4;//valid

How many statements are legal?

- A. One statement only
- B. Two statement only
- C. Three statement only
- D. Four statement only
- E. All 5 statements only.

Answer : C

Note:

Compiler -> it checks only the syntax and makes the jvm execution smooth

JVM -> Creates the memory for the variables and perform type casting and generates the result.

15)19-10-22 dbt session

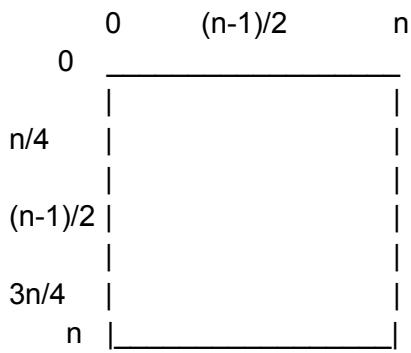
Increment and decrement operators cannot be applied on direct literals

Ternary operator

(a<b)?(c>d)?a:(e<f)?(a>d)?a:x;

16)19-10-22

=pattern programming is used to improve logical thinking of programmer



B pattern

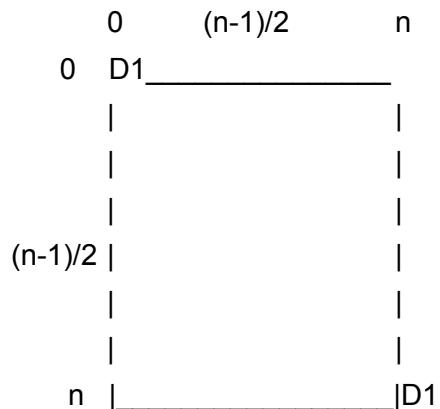
G pattern

P pattern

Internal O pattern

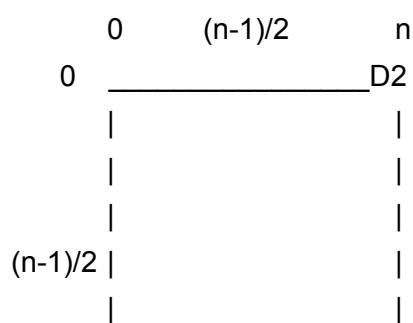
Print name of person

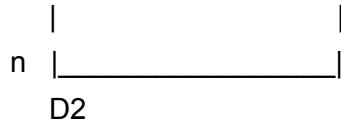
Alphabets in sequence



Diagonal line D1 = backward slash type

$i==j$

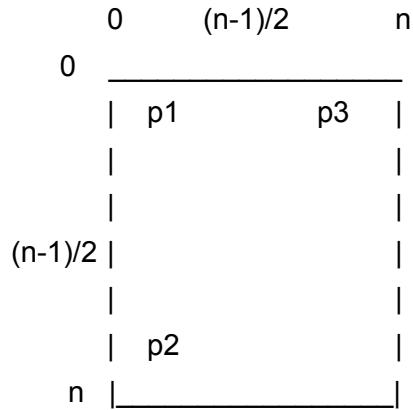




Diagonal line D2 = forward slash type

$$i+j==n-1$$

Rhombus type



Left side

$$P1 / \Rightarrow i+j==(n-1)/2$$

$$P2 \backslash \Rightarrow i-j==(n-1)/2$$

Right side

$$P3 \backslash \Rightarrow j-1==(n-1)/2$$

$$P4 / \Rightarrow i+j==n+(n-1)/2$$

Z pattern

Q pattern

Shapes

4)CodeSnippets

```

Q>
int x=10 , y=15 ;
if(++x < 10 & ++y > 15) { //11<10(false) & 16>15(true)
x++;
}

```

```
else {  
y++;//17  
}  
System.out.println(x+"----"+y);
```

predict x and y value by replacing the operators with

`||` => x and y value ?

`||` -> it is called as "Short circuit OR operator"

second operand evaluation will happen only if the first operand result is false

`x = 12`

`y = 16`

`&&` => x and y value

`&&` -> it is called as "Short circuit AND operator"

second operand evaluation will happen only if the first operand result is true

`x = 11`

`y = 16`

`|` => x and y value

`|`=> It is called as "Logical OR operator"

both the operands result will be evaluated.

`x = 12`

`y = 16`

`&` => x and y value ?

`&` -> It is called as "Logical AND operator"

both the operands result will be evaluated

`x =11`

`y=17`

```
Q>  
int x=10 ;  
if(++x < 10 && ((x/0)>10) ) { //11<10====> if(false)  
System.out.println("Hello");  
}  
else {  
System.out.println("Hi");  
}  
A. Hello  
B. Hi  
C. Compile Time Error
```

D. Exception

E. None of the above.

Answer : B

Check if $x \neq 0$ then check

&

Then exception is answer

Q>

Give

int i=10;//10

int j=20;//30 short circuit thing

int k= (j+=i)/5;// k = (j= j+i)/5 //short hand operator

k= (j=20+10)/5

k = (j=30)/5

k= 30/5

k= 6

System.out.println(i+":"+j+":"+k);

A. 10:30:6

B. 10:22:22

C. 10:22:20

D. 10:22:6

Answer : A

Q>

Syntax of if

```
if(boolean){  
stmt-1;  
stmt-2;  
}else{  
stmt-3;  
stmt-4;  
}  
int x =10;
```

if(x){//CE: unexpected type required: boolean,found:int

System.out.println("hello");

}else{

System.out.println("hiee");

}

A. hello

B. hiee

C. CompileTime Error

D. Some problem by jvm at the execution

E. None of the above

Answer: C

Q>

```
int x =10;  
if(x=20){//CE: unexpected type required: boolean,found:int  
System.out.println("hello");  
}  
else{  
System.out.println("hiee");  
}
```

- A. hello
- B. hiee
- C. CompileTime Error
- D. Some problem by jvm at the execution
- E. None of the above

Answer: C

Q>

```
int x =10;  
if(x==20){//operator used is Equality operator ==, != output is boolean  
System.out.println("hello");  
}  
else{  
System.out.println("hiee");  
}  
A. hello  
B. hiee  
C. CompileTime Error  
D. Some problem by jvm at the execution  
E. None of the above
```

Answer: B

Q>

```
boolean b= false;  
if(b=true){//assignment operator is evaluted on boolean data type, JVM if(true)  
System.out.println("hello");  
}  
else{  
System.out.println("hiee");  
}
```

- A. hello
- B. hiee
- C. CompileTime Error
- D. Some problem by jvm at the execution
- E. None of the above

Answer : A

Q>

```
boolean b= false;
if(b==true){//Equality operator result is boolean type , JVM if(false == true) ----
> if(false)
System.out.println("hello");
}else{
System.out.println("hiee");
}
A. hello
B. hiee
C. CompileTime Error
D. Some problem by jvm at the execution
E. None of the above
```

Answer: B

Q>

```
if(boolean)
stmt-1;
Note: if there is only statement which needs to be a part of if, then {} is
optional.
if(true)
System.out.println("hello");
A. Compile Time Error
B. hello
C. Some problem by jvm at the execution
D. None of the above
```

Answer : B

Q>

```
public class Test{
public static void main(String args[]){
if(true);
}
}
A. Compile Time Error
B. hello
C. Some problem by jvm at the execution
D. No Output
```

Answer : D(becoz ; is also valid java statement)

Q>

Note: if there is only statement which needs to be a part of if, then {} is optional, but that statement should not be a declarative statement.

```
public class Test{  
    public static void main(String args[]){  
        if(true)  
            int x=10; //CE: declaration not allowed here  
    }  
}
```

- A. Compile Time Error
- B. hello
- C. Some problem by jvm at the execution
- D. No Output

Answer: A

Q>

```
public class Test{  
    public static void main(String args[]){  
        if(true) {  
            int x=10; //valid for compiler becoz of {}  
        }  
    }  
}
```

- A. Compile Time Error
- B. hello
- C. Some problem by jvm at the execution
- D. No Output

Answer: D

Q>

```
public class Test{  
    public static void main(String args[]){  
        if(true)  
            System.out.println("hello");//Dependent of if statement  
        System.out.println("hiee");//Independent of if statement  
    }  
}
```

How many statements are independent of if?

- A. 0-stmt
- B. 1-stmt
- C. 2-stmt
- D. 3-stmt

Answer: B

17)20-10-22

Number patterns
And its manipulations
For ;i<n;
For ;j<n-i;
Pattern programming last part
Triangles , right angle triangles , left triangle, other triangles.

Logic of drawing face and logo's
Or
Complex patterns

Rocket diagram

=====

Introduction to oops

Types of variables

Division 1

- A.primitive variable
- B.reference variable

Division 2

- A.instance variable
- B.local variable
- C.static variable

JVM Area of execution

- a.Method Area (.class data/static data)
- b.Heap Area (instance variables/object data)
- c.Stack Area (local variables)
- d.PC register
- e.Native method Area

OOP's

It is actually a theory concept , which is implemented by many programming languages like c++, java, python,...

Any real time problem can be solved if we follow oop's principle.

NOTE : Software means collection of many programs

Programs means a set of instructions..

To write instructions we need to have a language.

Many scientists are tested and follow oop's language and build software

While solving the problem,

- 1.we need to first mark the objects
- 2.Every Object we mark should have 2 parts
 - a.HAS-Part/fields/attributes (store the informations variables)
 - b.Does-part->indicates what it can do
- 3.To represent an Object, first we need to have a blueprint of an Object.
- 4.we use "new" keyword/reserve word to create an Object for a blueprint(class).
- 5.Every Object Should always be in constant interaction
- 6.Useless Object doesn't exist.

Object = physical existence of any element we say as object

Realtime example : BookMyShow

Objects : Person, Ticket , Ticket Issuer , cinemahall , chair , 3D-glasses , Screen

eg: Student

|=> sid,
name,age,gender,email,address(variables/identifiers)
|=> play,study,drink,sleep(methods)

What is a blueprint in java and how to represent it?

In java to represent a blueprint we have a reserve word called "class".

Conventions

Conventions followed by java developers while writing a class is

a. className should be in "PascalConvention".

eg:BufferedReader,FileReader,InputStream,OutputStream,String,...

b. variables are represented in "camelCase".

eg: regNo,firstName,lastName,length,javaFullStack

c. methods are represented in "camelCase".

eg: toUpper(),toLower(),toString(),nextInt(),.....

eg#1.

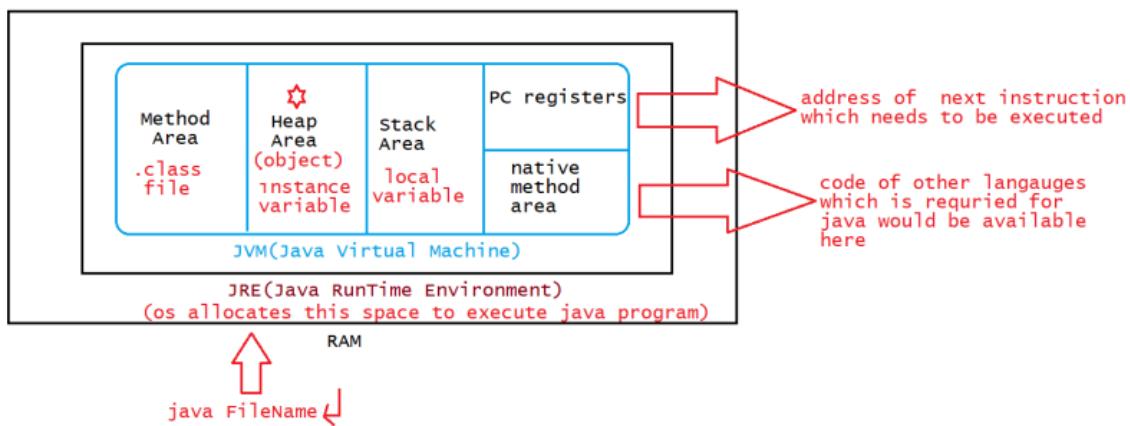
```
//Blueprint of Student Object
class Student{//Student -> PascalConvention
//HAS-Part ----> camelCaseConvention
int sid;
String name;
```

```

int age;
char gender;
String address;
//Does-Part ----> camelCaseConvention
void play(){}
void study(){}
void drink(){}
void sleep(){}
}

```

JVM DATA AREAS



new keyword

To create an object in java we use "new" keyword

Syntax:

ClassName variable=new ClassName();

new -> it is a signal to JVM to create some space for the Object in the heap area.

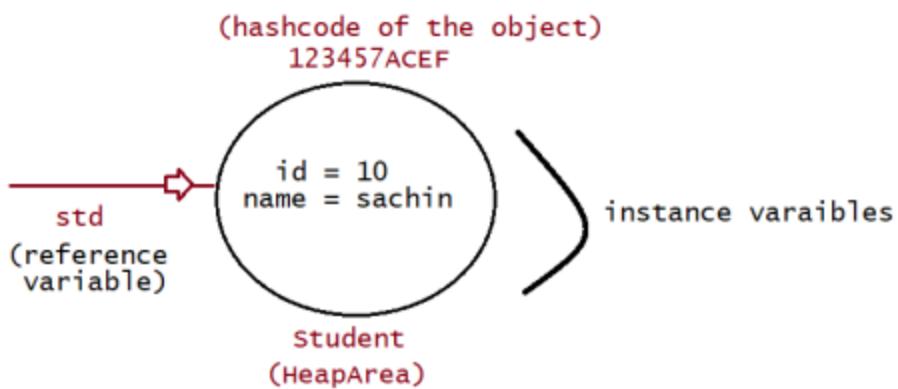
Tell the className, we inform the classname, JVM create the object and sends the

"hashCode" to the user.

Users should collect the hashCode through "reference variable".

```
student std = new Student();
```

↓ behind the scenes



realtime example : BookMyShow

Objects : Person,Ticket,Cinemahall,Chair,3D-glasses,Screen,.....

Note : Software means collection of many programs

Programs means a set of instructions.

To write instructions we need to have a language.

Types of variables

=====

Division 1 : Based on the type of value represented by a variable all variables are divided into 2 types.

They are:

1. Primitive variables
2. Reference variables

Primitive variables:

Primitive variables can be used to represent primitive values.

Example: int x=10;

Reference variables:

Reference variables can be used to refer to objects.

Example: Student s=new Student();

's' is the reference variable

instance variable:

If the variable is declared inside the class, but outside the methods such variables are called as "instance variables".

(or)

if the value of the variables changes from object to object then such variables are called as "instance variables"

eg#1.

```
Student std1= new Student();//id = 10, name =sachin
```

```
Student std2= new Student();//id = 7, name=dhoni
```

When will the memory for instance variable be given?

Ans. Only when the object is created JVM will create a memory and by default JVM will also assign the default value based on the datatype of the variable.

eg: int -> 0, float-> 0.0f, boolean -> false, char ->

, String -> null,....

Note: scope of instance variable would be available only when we have reference pointing to the object,

if the object reference becomes null, then we can't access "instance variables".

Key points about instance variables

Instance variables:

=> If the value of a variable is varied from object to object such type of variables are called instance variables.

=> For every object a separate copy of instance variables will be created.

=> Instance variables will be created at the time of object creation and destroyed at the time of object destruction

hence the scope of instance variables is exactly the same as the scope of objects.

=> Instance variables will be stored on the heap as part of the object.

=> Instance variables should be declared with in the class directly but outside of any method or block or constructor.

=> Instance variables can be accessed directly from the Instance area. But cannot be accessed directly from the static area.

=> But by using object reference we can access instance variables from static areas.

```
eg#1.  
public class Test {  
boolean b;  
public static void main(String[] args) {  
Test t=new Test();  
System.out.println(t.b);//false  
}  
}
```

```
eg#2  
public class Test {  
int i=10;//instance variable  
public static void main(String[] args) {  
System.out.println(i);//CE: instance variable can't be accessed  
directly in static context  
Test t=new Test();//Object created i = 10 is stored in heap area  
System.out.println(t.i);//10  
t.methodOne();  
}  
public void methodOne(){  
//inside instance method instance variable can be directly  
accessed.  
System.out.println(i);//10 because it is an instance variable  
}  
}
```

local variables

1. Variables which are created inside the method are called local variables and memory for those variables will be given in the stack area.
2. During the execution of the method the memory for local variables will be given, and after the execution of the method the memory for variables will be taken out from the stack area.
3. Local variables default value will not be given by the JVM, programmer should give the default value.
4. If the programmer doesn't give default value and if he uses the variable inside the method then the program would result in "CE".

```
class Test  
{  
    public static void main(String []arg)  
    {  
        //local variables, memory would be on the stack.
```

```

int a = 10;
int b = 20;
int c = a+b;
System.out.println(c);

int d; //local var's no default values by jvm
//if it is used then CE
}
}

```

Key Points of Local variables

Local variables:

=> Some times to meet temporary requirements of the programmer we can declare variables inside a method or block or constructors such type of variables are called local variables or automatic variables or temporary variables or stack variables.

=> Local variables will be stored inside the stack.

=> The local variables will be created as part of the block execution in which it is declared and destroyed once that block execution completes. Hence the scope of the local variables is exactly same as scope of the block in which we declared.

=> It is highly recommended to perform initialization for the local variables at the time of declaration at least with default values.

eg#1.

```

public class Test {
public static void main(String[] args) {
int i=0;
for(int j=0;j<3;j++)
{
i=i+j;
}
System.out.println(i);//valid
System.out.println(j);//CE: 'j' variable not declared
}
}

```

eg#2.

```

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

```

```

        int i=Integer.parseInt("ten");
    }
catch(NullPointerException e){
    System.out.println(i);//CE: 'i' not declared
}
}

```

eg#3.

```

class Test{
public static void main(String[] args){
int x;
System.out.println("hello");//hello
}
}

```

Note: code would be compiled becoz variable x is not used anywhere.

eg#4

```

class Test{
public static void main(String[] args){
int x;
System.out.println(x);//CE: 'x' not initialized
}
}

```

eg#5

```

class Test{
    public static void main(String[] args)
{
    int x;
    if(args.length>0){
        x=10;
    }
    System.out.println(x);
}
}

```

java Test 1

5)CodeSnippets

Q>

```

public class Test{
public static void main(String args[]){
int x=10;
switch(x)

```

```
{  
System.out.println("hello"); //Statement is not a part of case label so //CompileTime Error  
}  
}  
}  
}
```

- A. CompileTimeError
- B. hello
- C. JVM will create problem at the runtime
- D. None of the above

answer : A

Q>

```
switch(args){  
case label1: stmt-1;  
case label2: stmt-2;  
default : stmt-n  
}
```

label in switch should be "Compile Time Constants", meaning the value should be known to compiler otherwise CE>

```
public class Test{  
public static void main(String args[]){  
int x= 10;  
int y = 20; //this type of variables value is assigned by jvm at runtime  
switch(x)  
{  
case 10: System.out.println("hello");  
        break;  
case y:System.out.println("hiee"); //CE: 'y' value is not CompileTime //Constant  
        break;  
}  
}  
}
```

- A. CompileTimeError
- B. hello
- C. hiee
- D. JVM will create problem at the runtime
- E. None of the above

Answer: A

Q>

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

```
//final is the access modifier
final int y = 20;//final means compiler will get to know the value and
compiler treats it as "CompileTime Constant".
switch(x)
{
case 10: System.out.println("hello");
break;
case y:System.out.println("hiee");
break;
}
}
}
A. CompileTimeError
B. hello
C. hiee
D. JVM will create problem at the runtime
E. None of the above
```

Answer: B

Q>

```
public class Test{
public static void main(String args[]){
int x=10;
switch(x+1)
{
case 10:
case 10+20:
case 10+20+30:
}
}
}
```

- A. CompileTimeError
- B. No Output
- C. JVM will create problem at the runtime
- D. None of the above

Answer: B

Q>

```
switch(args){
case label1: stmt-1;
case label2: stmt-2;
default : stmt-n
}
```

label in switch should be "Compiletime Constants", meaning the value should be known to compiler otherwise CE>

label value should be within the range of switch argument type otherwise it would result in "CE".

```
public class Test{
public static void main(String args[]){
byte x=10;
switch(x)
{
case 10:System.out.println("hello");
break;
case 100: System.out.println("hiee");
break;
case 1000: System.out.println("byee");//CE: possibly loss of
precession from byte to int
break;
}
}
}
```

- A. CompileTimeError
- B. hello
- C. JVM will create problem at the runtime
- D. hiee

Answer : A

Q>

```
switch(args){
case label1: stmt-1;
case label2: stmt-2;
default : stmt-n
}
```

label in switch should be "Compiletime Constants", meaning the value should be known to compiler otherwise CE>

label value should be within the range of switch argument type otherwise it would result in "CE".

```
public class Test{
public static void main(String args[]){
byte x=10;
switch(x+1) //byte + int ----> int , so switch(int)
{
case 10:System.out.println("hello");
break;
case 100: System.out.println("hiee");
break;
}
```

```
case 1000: System.out.println("byee");
break;
}
}
}
```

- A. CompileTimeError
- B. hello
- C. JVM will create problem at the runtime
- D. hiee
- E. byee
- F. no output

Answer : F

Q>

```
switch(args){
case label1: stmt-1;
case label2: stmt-2;
default : stmt-n
}
```

label in switch should be "Compiletime Constants", meaning the value should be known to compiler otherwise CE>

label value should be with in the range of switch argument type otherwise it would result in "CE".

case lables value can't be duplicated, if we try to do it would result in "CE".

```
public class Test{
public static void main(String args[]){
int x=97;
switch(x){
case 97:System.out.println("97");
break;
case 99:System.out.println("99");
break;
case 'a':System.out.println("100"); // int x= 'a'; x = 97
break;
}
}
}
A. 97
```

- B. CompileTimeError
- C. JVM will create problem at the runtime
- D. 99
- E. 100

Answer: B

Q>

What will be the output of compiling and executing the Test class?

```
public class Test {  
    public static void main(String[] args) {  
        int a = 5;  
        int x = 10;  
        switch(x) {  
            case 10: a *= 2; // a = a*2 = 5*2 = 10, a = 10  
            case 20: a *= 3; // a = a*3 , a=10*3 = 30, a = 30  
            case 30: a *= 4; // a = a*4, a =30 * 4 =120, a= 120  
        }  
        System.out.println(a); //120  
    }  
}
```

A. 5

B. 10

C. 30

D. 120

E.CompileTimeError

Answer: D

```
public class Test{  
    public static void main(String arg[])  
    {  
        int x = 10;  
        int y = 20;  
        switch(x)  
        {  
            case 10: System.out.println("hello");  
                break;  
            Case y : System.out.println("hiee");  
                break;  
        }  
    }  
}
```

output:// compileTimeError

18)21-10-22

Method

1. Name
2. Input (parameter)
3. Body
4. Return type

syntax:

```
return_type method_name(type parameter,...)  
{  
    //task;  
}
```

Stack Area and Heap Area

(33:00 -)

Diagram

Java_program.java → javac(compiler) → Java_program.class (ByteCode) → jvm(interpreter) → execution

Any code if to execute task then it has to get into to stack area
Thread → line of execution

Stack frame memory for ref var's
Heap area for instance var's

Jvm gets methods to stack for execution
After the task is over then the method records are deleted.

When no ref pointing to heap area then garbage collector then it will deallocates the heap area

//method type 1 ⇒ NO INPUTS AND NO RETURNS

```
Class Calculator1  
{
```

```

Int a,b,c;
//this is the method type 1
void add()
{
    a = 10;
    b = 20;
    c = a + b;
    System.out.println(c);
}
}

public class Test{
    Calculator1 calc = new Calculator1();
    calc.add();
}

```

//method type 2 ⇒ INPUTS AND NO RETURNS

```

Class Calculator1
{
    int res;
    //this is the method type 2
    void add(int a , int b)//a and b are declared as within the //boundaries of the methods that
    are local variables
    //parameters because if user wants to give the inputs to perform task.
    //the method that accepts the data that are called as parameters.
    //taker will call it as parameters
    {
        res = a+b;
        System.out.println(res);
    }
}

public class Test{
    Calculator1 calc = new Calculator1();
    calc.add(3,5);
    //the arguments are required to pass, when
    //calling the specific method that contains the parameters.
    //giver gives the arguments
    //3,5 are arguments
}

```

Memory level

Control of execution first goes to main method,
Main method frame is in stack frame and ref variable

Body of main method starts executing through thread, add it calls the add() method during execution of add method , then control will return to main method.after main method execution. Memory is deallocated.

Stack frame life is as long as the task is present in it.

//method type 3 ⇒ NO INPUTS BUT RETURNS

```
class Calculator
{
    int a,b,res;

    int add()
    {
        a = 10;
        b = 20;
        res = a+b;
        return res;
    }
//whenever method is returning something then it is user wish to //whether to collect it or not
public class Test{
    public static void main(String[] args)
    {
        Calculator calc = new Calculator();
//to collect the value
        int c = calc.add();
        System.out.print(c);
    }
}
```

//method type 4 ⇒ WITH INPUTS AND RETURNS

```
class Calculator
{
    int add(int a, int b)
    {
        int res = a+b;
        return res;
    }
//whenever method is returning something then it is user wish to //whether to collect it or not
public class Test{
```

```

public static void main(String[] args)
{
    Calculator calc = new Calculator();
    //to collect the value
    int c = calc.add(10,20);
    System.out.print(c);
}

```

How address of obj is stored inside the ref var's?

ans.Using the constructor.

6)CodeSnippets (1:45:00)

Q>

```

public class Test{
    public static void main(String args[]){
        int x=?;
        switch(x)
        {
            default:System.out.println("default");
            case 0:System.out.println("0");
                break;
            case 1: System.out.println("1");
            case 2:System.out.println("2");
        }
    }
}

```

Note: replace x with 0,1,2,3 and predict the output

x = 0 output = 0

x = 1 output = 1

2

x = 2 output = 2

x = 3 output = default

0

Q>

```

Boolean b1 = true;    //Wrapper class , jvm b1=true,false
boolean b2 = false;   //primitive , jvm b2 =false
boolean b3 = true     //primitive , jvm b3= true
if ((b1 & b2) | (b2 & b3) & b3)          // bitwise operator return type -> boolean ,jvm if( false |
(false) & true),if(false&true), if(false)
    System.out.print("alpha ");
if ((b1 = false) | (b1 & b3) | (b1 | b2)) //bitwise operator return type->boolean,
    System.out.print("beta "); //jvm if( false | (false & true) |(false|false)),if(false | false|false),if(false)

```

What is the result?

- A. beta
- B. alpha
- C. alpha beta
- D. Compilation fails.
- E. No output is produced.
- F. An exception is thrown at runtime.

Answer: E

Q>

```
1. class Maybe {  
2. public static void main(String[] args) {  
3. boolean b1 = true;  
4. boolean b2 = false;//b2=true  
5. System.out.print(!false ^ false);// true ^ false => true  
6. System.out.print(" " + (!b1 & (b2 = true)));//false &true => false  
7. System.out.println(" " + (b2 ^ b1));//true ^ true => false  
8. }  
9. }
```

Which are true?

- A. Line 5 produces true.
- B. Line 5 produces false.
- C. Line 6 produces true.
- D. Line 6 produces false.
- E. Line 7 produces true.
- F. Line 7 produces false.

Note: \wedge -> xor => both operands same means false, otherwise true.

eg: true \wedge true => false
true \wedge false => true

\sim => bitwise complement
 $!$ => boolean complement

Answer: A,D,F

Q>

Given.

```
class Sixties {  
public static void main(String[] args) {  
    int x = 5;//JVM x= 5  
    int y = 7;//JVM y = 7  
    System.out.print(((y * 2) % x));// S.o.p( (7*2)%5) , S.o.p( 14%5), S.o.p(4);  
    System.out.print(" " + (y % x));// S.o.p( 7 % 5) , S.o.p(2);  
}  
}
```

What is the result?

- A. 1 1

- B. 1 2
- C. 2 1
- D. 2 2
- E. 4 1
- F. 4 2
- G. Compilation fails.
- H. An exception is thrown at runtime.

Answer: F

Q>

```
class Hexy {
    public static void main(String[] args) {
        Integer i = 42;//JVM i = 42
        String s = (i<40)?"life":(i>50)?"universe":"everything";// (42<40)? true: (42>50) ? true :
        "everything";
        System.out.println(s);//S.o.p(everything)
    }
}
```

What is the result?

- A. null
- B. life
- C. universe
- D. everything
- E. Compilation fails.
- F. An exception is thrown at runtime.

Answer: D

Q>

Given.

```
public class Foozit {
    public static void main(String[] args) {
        Integer x = 0;
        Integer y = 0;
        for(Short z = 0; z < 5; z++)
            if((++x > 2) || (++y > 2))
                x++;
        System.out.println(x + " " + y);
    }
}
```

What is the result?

- A. 5 1
- B. 5 2
- C. 5 3
- D. 8 1
- E. 8 2
- F. 8 3
- G. 10 2
- H. 10 3

- I. Compiletime Error
- J. Some problem created by JVM during execution

Answer: E

Constant revision is required

19)27-10-22

Method Overloading

= multiple methods with same name and different parameter.(return type has nothing to do)
//refers to process of writing more than one method with same name and different parameters
within a class

Compile time polymorphism

false polymorphism

Earlytime binding

code it in eclipse

```
class Calculator
{
    int add(int a,int b)
    {
        return a+b;
    }
    int add(int a, int b, int c)
    {
        return a+b+c;
    }
    float add(float a,float b)
    { return a+b; }
    float add(int a ,float b,float c)
    { return a+b+c; }
    double add(int a,int b, double c)
    { return a+b+c; }

}
public class Test{
    public static void main(String[] arg)
    {
        Calculator calc=new Calculator();
        int a = 10 , b = 30 , c = 20 ;
        float m = 23.5f , n = 22.5f , o = 30.4f;
```

```

        double x = 15.5, y = 34.22222;
        System.out.println(calc.add(a,b));
        System.out.println(calc.add(n,a));
        System.out.println(calc.add(x,y));
        System.out.println(calc.add(a,b,c));
    }
}

```

One application is developed by multiple developers,
 Every developer handles different modules and classes.
 The one developer who uses the method then he has to remember the methods name;

He thinks that one method is overloaded with multiple tasks, but it is one task;

1:M is also called as polymorphism
 add method is the false polymorphism

One class \Rightarrow many methods with same name

Many methods with same name and different parameters

Compiler resolves this issue so only it is called as compile time polymorphism.

1. Based on number of parameters
2. Data type of particular parameter
3. Order of the type of the parameter

CAN WE OVERLOAD THE MAIN METHOD ?

YES, but jvm will call only a particular main method signature.

```

class Calculator
{
    int add(int a,int b)
    {
        return a+b;
    }
    void add(int a, int b)
    {
        return a+b;
    }
    float add(float a,float b)
    { return a+b; }
    float add(int a ,float b,float c)
    { return a+b+c; }
    double add(int a,int b, double c)
}

```

```

    { return a+b+c; }

}

public class Test{
    public static void main(String[] arg)
    {
        Calculator calc=new Calculator();
        calc.add(2,3);
        int a = 10 , b = 30 , c = 20 ;
        float m = 23.5f , n = 22.5f , o = 30.4f;
        double x = 15.5, y = 34.22222;

        System.out.println(calc.add(a,b));
        System.out.println(calc.add(n,a));
        System.out.println(calc.add(x,y));
        System.out.println(calc.add(a,b,c));
    }
}

//IT WILL GIVE COMPILE TIME ERROR

//Method Overloading with Numeric type promotion
class Calculator
{

    float add(float a , int b)
    { return a+b; }
    float add(float a,float b,float c)
    { return a+b+c; }

}

public class Test{
    public static void main(String[] arg){

        Calculator calc = new Calculator();
        System.out.println(calc.add(10,20));
    }
}

```

Main method overload
 (String arg[])
 (int arg[])
 (double arg[])

Arrays

=array is the concept that is present all programming languages.

To store data we use variables

For example to store the marks of two students

To store 5 or 20 values we create 5 or 20 var's

Instead of it we can store it in arrays.

At a single time variable can store only one value.

If other value then previous value is over written.

If 200 var's then in var's concept we store in 200 var's with diff names.

For large volume of data, for efficient ways is to store in under single name is using arrays.

=Array is indexed based data structure to store large volume of data using single name

=Arrays can store homogeneous data

=Arrays in java are treated as objects and stored in heap area.

To create a object new keyword

```
int[] a=new int[5]; //5 locations are given in heap area.  
//a is an array of int[] type data
```

```
//to initialize  
a[0] = 2;  
a[1] = 20;
```

//to access

```
sop(a[1]);
```

CASE 1 : to store marks of 5 students

1D

Students

```
int[] ar=new int[5];
```

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

index	→	0	1	2	3	4
value	→	1	2	3	4	5

arr[0] arr[1] arr[2] arr[3] arr[4]

CASE 2 : to store marks of 3 class each with 4 students

2D

Class Students

0	4
1	4
2	4

//array declaration and creation

```
int[][] ar = new int[3][4];
```

ar[0][0]

ar[0][1]

Ar

CASE 3 : to store 2 colleges each collage having 3 classes with 4 students

3D

College Classes Students

0	0	0
	0	1
	0	2
	0	3
	1	0
	1	1
	1	2
	1	3
	2	0
	2	1
	2	2
	2	3
1	0	0
	0	1
	0	2
	0	3
	1	0
	1	1
	1	2
	1	3
	2	0
	2	1
	2	2
	2	3

```
int[][][] ar = new int[2][3][4];
```

Irregular Array = jagged Array

```
int[][] ar=new int[3][];
//like this array is not created
//then
ar[0] = 13;
ar[1] = 20;
ar[2] = 30;
```

```
int[][][] ar = new int[2][][];
ar[0] = new int[2][];
ar[1] = new int[1][];
```

```
ar[0][0] = new int[20];
ar[0][1] = new int[10];
```

```
ar[1][0] = new int[23];
```

IN JAVA WE CAN CREATE MULTI DIMENSIONAL ARRAYS.

Need of array,what are arrays,why arrays ?
How to create arrays ?

7)CodeSnippets

Q>

Syntax:

```
while(boolean){
stmt-1;
stmt-2
}
public class ExampleWhile{
public static void main(String args[]){
while(1){//1 is not boolean in java
System.out.println("hello");
}
}
```

A. CompileTime Error

- B. Infinite times hello
- C. hello
- D. some problem by jvm during execution

Answer: A

Q>

```
public class ExampleWhile{  
public static void main(String args[]){  
while(true){  
System.out.println("hello");  
}  
}  
}
```

- A. CompileTime Error
- B. Infinite times hello
- C. hello
- D. some problem by jvm during execution

Answer: B

Q>

Syntax

```
while(true)  
stmt-1;  
; -> it is a valid java statement  
public class ExampleWhile{  
public static void main(String args[]){  
while(true);  
}  
}
```

- A. CompileTime Error
- B. Infinite times hello
- C. hello
- D. some problem by jvm during execution
- E. Infinite loop with no output

Answer: E

Q>

```
public class ExampleWhile{  
public static void main(String args[]){  
while(true)  
int x =10;//declarative statements are not allowed  
}  
}  
A. CompileTime Error
```

- B. some problem by jvm during execution
- C. Memory for x will be given 4bytes becoz of type int
- D. None of the above

Ansver: A

Q>

```
public class ExampleWhile{
    public static void main(String args[]){
        while(true){
            int x =10;
        }
    }
}
```

- A. CompileTime Error
- B. some problem by jvm during execution
- C. Memory for x will be given 4bytes becoz of type int during execution
- D. None of the above

Answer: C

Q>

```
public class ExampleWhile{
    public static void main(String args[]){
        while(true){
            System.out.println("hello");//line-n1
        }
        System.out.println("hiee");//line-n2
    }
}
```

- A. CompileTime Error at line-n1
- B. hello infinite times
- C. hiee
- D. some problem by jvm during the execution
- E. CompileTime Error at line-n2
- F. None of the above

Answer: E(unreachable code)

Q>

```
public class ExampleWhile{
    public static void main(String args[]){
        while(false){
            System.out.println("hello");//line-n1
        }
        System.out.println("hiee");//line-n2
    }
}
```

}

- A. CompileTime Error at line-n1
- B. hello
- C. hiee
- D. some problem by jvm during the execution
- E. CompileTime Error at line-n2
- F. None of the above

Answer: A(unreachable code)

Q>

```
public class ExampleWhile{  
    public static void main(String args[]){  
        int a =10,b=20;  
        while(a<b){ 10<20 =====> while(true)  
            System.out.println("hello");//line-n1  
        }  
        System.out.println("hiee");//line-n2  
    }  
}
```

- A. CompileTime Error at line-n1
- B. hello
- C. hiee
- D. some problem by jvm during the execution
- E. CompileTime Error at line-n2
- F. None of the above

Answer: F

Q>

Note: Whenever variables are marked as final, compiler will get to know the value of those variables and it will use the values in the Expression to get the result.

```
public class ExampleWhile{  
    public static void main(String args[]){  
        final int a =10,b=20;//compiler knows the value of the variables a, b.  
        while(a<b){// while(10<20) ----> while(true)  
            System.out.println("hello");//line-n1  
        }  
        System.out.println("hiee");//line-n2  
    }  
}
```

- A. CompileTime Error at line-n1
- B. hello
- C. hiee
- D. some problem by jvm during the execution

E. CompileTime Error at line-n2

F. None of the above

Answer: E

Q>

```
public class ExampleWhile{  
    public static void main(String args[]){  
        final int a =10;  
        while(a<20){// Compiler ===== while(10<20) ---> while(true)  
            System.out.println("hello");//line-n1  
        }  
        System.out.println("hiee");//line-n2  
    }  
}
```

A. CompileTime Error at line-n1

B. hello

C. hiee

D. some problem by jvm during the execution

E. CompileTime Error at line-n2

F. None of the above

Answer: E

Q>

```
public class ExampleWhile{  
    public static void main(String args[]){  
        int a =10;  
        while(a<20){//JVM===== while(10<20) ===== while(true)  
            System.out.println("hello");//line-n1  
        }  
        System.out.println("hiee");//line-n2  
    }  
}
```

A. CompileTime Error at line-n1

B. hello

C. hiee

D. some problem by jvm during the execution

E. CompileTime Error at line-n2

F. hello infinite times

G. None of the above

Answer: F

Q>do-while

```
public class ExampleDoWhile{
```

```
public static void main(String args[]){
do
System.out.println("hello");//line-n1
while(true);
}
}

A. CompileTime Error at line-n1
B. hello infinite times
C. some problem by jvm during the execution
D. hello
E. None of the above
```

Answer: B

Q>

Note: If we are using do while then there should be atleast one statement as the body otherwise it would result in "CompiletimeError".

```
public class ExampleDoWhile{
public static void main(String args[]){
do
while(true);
}
}
```

- A. Compile time error
- B. some problem by jvm during the execution
- C. No output
- D. None of the above

Answer: A

Q>

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

```

```
do;
while(true);
}
}
```

- A. Compile time error
- B. some problem by jvm during the execution
- C. No output
- D. None of the above

Answer: C

Q>

```
public class ExampleDoWhile{  
    public static void main(String args[]){  
        do  
            int x =10; //line-n1  
            while(true);  
        }  
    }  
A. Compile time error at line-n1  
B. some problem by jvm during the execution  
C. No output  
D. None of the above
```

Answer: A

Q>

```
public class ExampleDoWhile{  
    public static void main(String args[]){  
        do{  
            int x =10; //line-n1  
        }while(true);  
    }  
A. Compile time error at line-n1  
B. some problem by jvm during the execution  
C. No output  
D. None of the above
```

Answer: C

Q>

```
public class ExampleDoWhile{  
    public static void main(String args[]){  
  
        do while(true)  
        System.out.println("hello");  
        while(true);  
  
    }  
A. CompileTime Error  
B. hello infinite times  
C. some problem by jvm during the execution  
D. hello  
E. None of the above
```

Answer: B(refer to the changes made by compiler as shown below)

Compiler will do

```
-----  
do  
    while(true)  
        System.out.println("hello");  
while(true);
```

20)28-10-22

(Scanner)User input from console

Scanner class

```
class Scanner{  
    //so many methods present in it./it is present in util package  
}
```

Coding has done by java

```
Scanner scan = new Scanner(System.in);  
int a = scan.nextInt();  
String str = scan.next();
```

A program using scanner

Arrays Programs

To store marks of 5 students

To take user inputs

```
int[] a = new int[5];  
Scanner scan = new Scanner(System.in);  
  
for(int i = 0;i<5;i++)  
{  
    a[i] = scan.nextInt();  
}
```

Here $i < 5$ is called upper limit.

It should not be declared like that.

length is the property of arrays
Duty is to get length of array.

a.length

=gives the length of array declared

So $i < a.length$ is recommended in program, upper's limit
= this is called as generalistic approach

JAVA IS MOST STRUCTURED AND WELL ORGANIZED PROGRAMMING, STRONGLY TYPED LANGUAGE

```
sop("enter the number of students: ");  
int size = scan.nextInt();  
int[] std = new int[size];
```

Works Check

```
int [][] a = new int[2][3][3];
```

Write array programs for college,classes,students

Every is byte is RAM.

RAM is the collection of bytes.

BUFFEROVER

Java don't have bufferover run

Arrays in java are guarded with boundaries

Compiler duty is to check syntax.

Array we can create collection of objects

Real world data is heterogenous type data.

Disadvantages of Arrays

- =>It can store only homogenous type of Data
- =>The memory of Array is fixed in size.(it can't grow or shrink)
is changed then new array object created and it refers to it.
- =>Array demands contiguous memory locations (it can't use dispensed locations)
- =>Max range of array is $2^{pow(n-1)}$ to $2^{pow(31)}$
Out of memory error is exceeded

Recommended(For readability)

```
int[] a = new int[n];
```

Not other styles are recommended\

```
//below syntax is used for practice purpose,not in real applications
int[] a = {1,2,3,4};
char[] c = {'a','b'};

int[][] ar = {{10,20},{2,4}};

int[] arr = {{23,43},{2,3}},{{23,34},{23,23}}};

int[][][] ad = new int[][][]{{{23,43},{2,3}},{{23,34},{23,23}}};
//allowed
int[] a = new int[]{1,2,3,4};
```

For each loop (enhanced loop)

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

for(int elem:a)
{
    sop(elem);
}
//index is not required
//you can iterate only in from first
```

8)CodeSnippets

Q>

```
public class ExampleDoWhile{
public static void main(String args[]){
do{
    System.out.println("hello");//line-n1
}while(true);
System.out.println("hi");//line-n2
}
}

A. CompileTime Error at line-n1
```

- B. hello infinite times
- C. hi
- D. some problem by jvm during the execution
- E. CompileTime Error at line-n2
- F. None of the above

Answer: E

Q>

```
public class ExampleDoWhile{  
public static void main(String args[]){  
do{  
System.out.println("hello");//line-n1  
}while(false);  
System.out.println("hi");//line-n2  
}  
}
```

- A. CompileTime Error at line-n1
- B. hello
- C. hi
- D. some problem by jvm during the execution
- E. CompileTime Error at line-n2
- F. hello

hi

- G. None of the above

Answer: F

Q>

```
public class ExampleDoWhile{  
public static void main(String args[]){  
int a=10,b=20;  
do{  
System.out.println("hello");//line-n1  
} while(a<b);// JVM ----> while(10<20) ---> while(true)  
System.out.println("hi");//line-n2  
}  
}
```

- A. CompileTime Error at line-n1
- B. hello
- C. hi

- D. some problem by jvm during the execution
- E. CompileTime Error at line-n2
- F. hello infinite times
- g. hi infinite times
- h. None of the above

Answer: F

Q>

```
public class ExampleDoWhile{  
    public static void main(String args[]){  
        int a=10,b=20;  
        do{  
            System.out.println("hello");//line-n1  
        } while(a>b);// JVM ---> while(10>20) ---> while(false)  
        System.out.println("hi");//line-n2  
    }  
}
```

- A. CompileTime Error at line-n1
- B. hello
- hi
- C. hi
- hello
- D. some problem by jvm during the execution
- E. CompileTime Error at line-n2
- F. hello infinite times

hi

- G. None of the above

Answer: B

Q>

```
public class ExampleDoWhile{  
    public static void main(String args[]){  
        final int a=10,b=20;  
        do{  
            System.out.println("hello");//line-n1  
        } while(a<b);//Compiler----> while(10<20) ---> while(true)  
        System.out.println("hi");//line-n2  
    }  
}
```

- A. CompileTime Error at line-n1
- B. hello
- hi
- C. hi
- hello
- D. some problem by jvm during the execution
- E. CompileTime Error at line-n2
- F. hello infinite times
- hi
- G. None of the above

Answer: E(concept of unreachable)

Q>

```
public class ExampleDoWhile{  
    public static void main(String args[]){  
        final int a=10,b=20;  
        do{  
            System.out.println("hello");//line-n1  
        } while(a>b);//Compiler ----> while(10>20) ---->  
        while(false)  
        System.out.println("hi");//line-n2  
    }  
}
```

- A. CompileTime Error at line-n1
- B. hello
- hi
- C. hi
- hello
- D. some problem by jvm during the execution
- E. CompileTime Error at line-n2
- F. hello infinite times
- hi
- G. None of the above

Answer: B

Q>

```
int i=0,j=0; //line -n1  
int i=0,Boolean b=true; //line-n2  
int i=0,int j=0; //line -n3
```

How many statements are valid?

- A. line -n1 and line -n3
- B. line -n2
- C. line-n1, line-n2 and line-n3
- D.line -n3
- E. line -n1

Answer: E(after , in declaration we need to just specify the variables only)

Q>

Syntax:

```
for(stmt1;stmt2;stmt3){  
    stmt4;  
}
```

stmt1 -> can be any statement, but suggested for initialisation

stmt2 -> compulsorily should be boolean statement only

stmt3 -> can be any statement, but suggested for inc/dec on a variable

stmt4 -> can be any statement, suggested for repetitive logic

```
public class ExampleFor{  
    public static void main(String args[]){  
        int i=0;  
        for(System.out.println("hello u r sleeping");i<3;i++){  
            System.out.println("no boss, u only sleeping");  
        }  
    }  
}
```

Predict the Output:

- A. Compile Time Error
- B. Some problem occurred by jvm during execution
- C. hello u r sleeping
- D. no boss, u only sleeping
- E.

hello u r sleeping

No boss, u only sleeping

No boss, u only sleeping

No boss, u only sleeping

Answer: E

Q>

```
public class ExampleFor{
```

```
public static void main(String args[]){
int i=0;
for(System.out.println("hello");i<3;System.out.println("hi")){
i++;
}
}
}
```

Predict the Output:

- A. Compile Time Error
- B. Some problem occurred by jvm during execution
- C. hello
- hi
- D. hello
- hi
- hi
- hi
- E. hi
- hi
- hi
- hello
- F. None of the above

Answer: D

Q>

```
public class ExampleFor{
public static void main(String args[]){
for(;;){//Compiler--> boolean value will be evaluated as 'true'
System.out.println("hello");
}
}
}
```

Predict the Output:

- A. Compile Time Error
- B. Some problem occurred by jvm during execution
- C. hello
- D. infinite times hello
- E. None of the above

Answer: D

Q>

```
public class ExampleFor{  
    public static void main(String args[]){  
        for(int i=0;true;i++){  
            System.out.println("hello");//line-n1  
        }  
        System.out.println("hi");//line-n2  
    }  
}
```

Predict the Output:

- A. Compile Time Error at line-n1
- B. Compile Time Error at line-n2
- C. Some problem occurred by jvm during execution
- D. hello
- hi
- D. infinite times hello followed by hi
- E. None of the above

Answer: B

Q>

```
public class ExampleFor{  
    public static void main(String args[]){  
        for(int i=0;false;i++){  
            System.out.println("hello");//line-n1  
        }  
        System.out.println("hi");//line-n2  
    }  
}
```

Predict the Output:

- A. Compile Time Error at line-n1
- B. Compile Time Error at line-n2
- C. Some problem occurred by jvm during execution
- D. hello
- hi
- D. infinite times hello followed by hi
- E. None of the above

Answer: A(remember the concepts of unreachable)

Q>

```
public class ExampleFor{
public static void main(String args[]){
for(int i=0;i++){
System.out.println("hello");//line-n1
}
System.out.println("hi");//line-n2
}
}
```

Predict the Output:

- A. Compile Time Error at line-n1
- B. Compile Time Error at line-n2
- C. Some problem occurred by jvm during execution
- D. hello

hi

- D. infinite times hello
- E. None of the above

Answer : B

Q>

```
public class ExampleFor{
public static void main(String args[]){
int a=10,b=20;
for(int i=0;a<b;i++){//JVM ----> 10<20 ( true)
System.out.println("hello");//line-n1
}
System.out.println("hi");//line-n2
}
}
```

Predict the Output:

- A. Compile Time Error at line-n1
- B. Compile Time Error at line-n2
- C. Some problem occurred by jvm during execution
- D. hello

hi

- E. infinite times hello
- F. None of the above

Answer: E

Q>

```
public class ExampleFor{
public static void main(String args[]){
final int a=10,b=20;
for(int i=0;a<b;i++){//Compiler ---> 10<20 (true)
System.out.println("hello");//line-n1
}
System.out.println("hi");//line-n2
}
}
```

Predict the Output:

- A.Compile Time Error at line-n1
- B.Compile Time Error at line-n2
- C. Some problem occurred by jvm during execution
- D. hello
- hi
- E. infinite times hello
- F. None of the above

Answer: B

Note: if a variable is marked as final, then those values are known to compiler so we say final variables as "CompiletimeConstants".

if a variable is marked as final, then the value for those variables should never be changed in the program, if we try to change it would result in "CompileTimeError".

In java memory for a variable is given by JVM as per its datatype specification and value also will be assigned.

compiler will not allocate memory for the variables and it will not initialize the value for the variable.

```
eg: int a =10;
a++;
System.out.println(a);//11
eg: final int a =10;
a++;//a = a +1;//CE: value can't be re-assigned
System.out.println(a);
```

21)29-10-22 dbt

Object o[] = new Object[5];

Now it can hold any type of data of object like student,employee..

It is generic

Wrapper class object

To make java 100% pure object oriented then instead of primitive's, wrapper classes are used in project.

To hold normal values as objects wrapper class is used.

3d array

```
int a[][][] = {  
    {  
        {1,3}  
    },  
    {  
        {2,3}  
    }  
}
```

JAVA doesn't support pointers because of its trickiness in other languages like c,c++

Pointers code always leads to program crash if it is not handled properly, so in java they removed the concept of pointers

SIZE OF OBJECT

```
Class Student{  
    int sid;// 4byte  
    String name;//depends on no; of char's in string eg: "sachin"/12byte  
}
```

Total object size of "16 bytes"

Java programmer won't thinks about memory, it is handled by jvm for objects.

DAO class

It is an object which interacts with database and stored the record details in object.

```
int a = 10;  
sop(a);
```

```
Integer a = 10;
```

```
sop(a.SIZE);
```

float , double

Any operation on float we do result can be stored in “float” until and unless u expect 11 decimal precession.

```
float a = 10;  
float b = 3;  
float c = a*b; // compiler accepts it ,
```

22)31-10-22

Arrays

for each loop for the jagged array iteration and printing

2D and 3D

=to access entire array in forwarded direction then for each loop is best without knowing length.
We don't need to worry about starting and ending indices.

LIMITATIONS

1. We can't access index's (no index control)
2. It can traverse or iterate only in forward direction

Problems

Access array in reverse direction.

Write a program to fetch all alternative elements

Valid declaration of arrays

```
int[] a;  
int[][] a;  
int a[][],  
int []a[];
```

```
int[] a,b;  
a = new int[4];  
b = new int[5];
```

int [] a[],b;//before and after var's - square brackets

//a will be 2D , b will be 3D

//int []arr,[]bb; //CE

Usually we go with collection

Whenever we know exact size, similar data, primitive values then we use array

Entire is object, but inside obj = primitive values.
Or we can use wrapper classes instead of primitive values.

Uses of array

Collections can store only objects.
But array can store both primitive and object.

Size of array can't be long,float,double,boolean
(exception)
Java compiler can compile size is negative, but it cause exception.

It can be short,byte,int,char

Char to int is implicit casting

```
int[] a = new int['a'];
```

We can create array of strings
String[] s = new String[4];

~~>Exception are the type of errors occurred at runtime error or mistake.
Exceptions : arrayStoreException, Array out of memory

If class is not there then you can't create object.
But int[] a = new int[4];
Every type of array has specific class inbuilt, that are for java language
We can't use that, because that are for internal purpose.

Internal classes for arrays inbuilt classes

```
Int[] i = new int[4];
sop(i.getClass().getName());
char[] c = new char[4];
sop(c.getClass().getName());
double[] d = new double[4];
```

But they have provided utility classes that are Arrays=>sort,fill,binary search

Static and non static approach

Array Classes

All method inside array class are static methods

Directly with class name we can call.

Arrays.sort

To sort

We need to write a logic,
So instead of it they provided the support and classes.

Arrays.sort[a];

To add the elem in an array

Default values of arrays to fill all with same

int[] a = new int[5];

Logically we can write

for(int i=0;i<5;i++)

 a[i]=5;

With the help of utility class and support class

With method

Arrays.fill[a,5]

Program

To add all elements

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

```
int sum = 0;
```

```
for(int i=0;i<a.length-1;i++)
```

```
    sum = sum + a[i];
```

```
sop("the sum is " + sum);
```

problems

Sum of elems in array

Max elem of array

Min elem of array

Reverse the array

Alternative values of array

Copy one array to another array
Searching and sorting problems

(for hacking java is not suitable)

9)CodeSnippets

transfer statements

=====

- 1. break
- 2. continue
- 3. return

Q>

```
class Test{  
public static void main(String args[]){  
int x=0;  
switch(x){  
case 0:System.out.println("hello");  
break ;//It is used to avoid fallthrough in switch  
case 1:System.out.println("hi");  
}  
}  
}
```

Predict the Output:

- A.Compile Time Error
- B. Some problem occured by jvm during execution
- C.hello
- hi
- D. hello

Answer: D

Q>

```
class Test{  
  
public static void main(String args[]){  
  
for(int i=0; i<10; i++) {// i = 0,0<10(true),i =1,1<10(true)  
    if(i==5)  
        break;//control will come out of the executing loop  
    System.out.print(i);//0,1,2,3,4  
}
```

}

}

Predict the Output:

- A. Compile Time Error
- B. Some problem occurred by jvm during execution
- C. 0 1 2 3 4
- D. 0 1 2 3 4 6 7 8 9

Answer: C

Q>label

```
class Test{  
public static void main(String args[]){  
//label (means block to come out as mentioned)(means any valid identifier  
int x=10;  
l1 : {  
    System.out.println("begin");  
    if(x==10)  
    break l1;  
    System.out.println("end");  
}  
System.out.println("hello");  
}  
}
```

Predict the Output:

- A. Compile Time Error
- B. Some problem occurred by jvm during execution
- C. begin
- end
- D. begin
- end
- hello
- E. None of the above

Answer: E(begin and hello will be printed)

Q>

```
class Test{  
public static void main(String args[]){  
int x=10;  
if(x==10)  
break;//only at switch,loop and label //this three places only //other places compiler time error  
System.out.println("hello");
```

```
}
```

```
}
```

Predict the Output:

- A. Compile Time Error
- B. Some problem occurred by jvm during execution
- C. hello
- D. None of the above

Answer: A (break can be used in switch,loop and labelled block,otherplace compile time error)

Q>continue

```
class Test{  
public static void main(String args[]){  
int x=2;  
for(int i = 0; i<10;i++){  
if(i%x==0)  
Continue;  
//it will skip the current iteration and proceeds with next iteration  
System.out.println(i);  
}  
}  
}
```

Predict the Output:

- A. Compile Time Error
- B. Some problem occurred by jvm during execution
- C. 0 2 4 6 8
- D. 1 3 5 7 9
- E. None of the above

Answer: D

Q>

```
class Test  
{
```

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

```
int x=10;  
if(x==10)  
continue;//continue can't be used here  
System.out.println("hello");  
}  
}
```

Predict the Output:

- A. Compile Time Error
- B. Some problem occurred by jvm during execution
- C. hello
- E. None of the above

Answer: A

Q>

```
class Test{
public static void main(String args[]){
int x=0;
switch(x){
case 0:System.out.println("hello");
continue;
case 1:System.out.println("hi");
}
}
}
```

Predict the Output:

- A.Compile Time Error
- B. Some problem occurred by jvm during execution
- C.hello

hi

- D. hello

Answer: A(continue can be used only in loops and labelled block,otherplaces compile time error)

Note:

```
I1:
for()
{
I2: for()
{
I3: for()
{
break;//break I3; // goto stmt-1
break I2; //goto stmt2
break I1;// goto stmt3
}
stmt-1;
}
stmt-2;

}
stmt-3;
```

```

Q>
class Test{

public static void main(String args[]){

I1:for(int i=0;i<3;i++) // i = 0 ,0<3(true) , i =1, 1<3(true), i =2 , 2<3(true), i = 3 , 3<3(false)
{
for(int j=0;j<3;j++)// j =0,0<1(true), j =1, 1<3(true),j =2 ,2<3(true)
{
if(i==j)
stmt1;
System.out.println(i+" "+j);//10 20 21

}
}

}

}

replace stmt1 with break and predict the output?
Answer:1 0 2 0 2 1
replace stmt1 with break I1 and predict the output?
Answer: no output
replace stmt1 with continue and predict the output?
Answer: 01, 02,10,12,20,21
replace stmt1 with continue I1 and predict the output?
Answer: 10 20 21

```

```

Q>
class Test{

public static void main(String args[]){
while(true){

System.out.println("hello");//line-n1

}
System.out.println("hi");//line-n2
}

}

Predict the output
A. Compile time error at line-n1

```

- B. some problem occurred during jvm execution.
- C. Compile time error at line-n2

D. hello

hi

E. hello infinite times

F. None of the above

Answer: C

Q>

```
class Test{
```

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

```
if(true){
```

```
System.out.println("hello");//line-n1
```

```
}
```

```
else{
```

```
System.out.println("hi");//line-n2 //it is dead code.
```

```
}
```

```
}
```

```
}
```

Predict the output

A. compile time error at line-n1

B. some problem occurred during jvm execution.

C. Compile time error at line-n2

D. hello

hi

E. hello

F. hi

G. None of the above

Answer: E

Concept of unreachable holds good only for loops(for,while,dowhile), compiler will ignore unreachable for

if else and switch syntax.

Q>

```
class A
```

```
{
```

```
void add(byte a, byte b)
```

```
{
```

```
//stmt;
```

```
}
```

```
}
```

```
class Test
{
public static void main(String[] args)
{
A obj=new A();
obj.add(10,20); //Line 1 => CE: Why it is throwing CE.
//The value 10 isn't between byte range(-128 to 127 )
//add(10,20)//will happen at jvm level,method calling will happen by jvm,compiler with its know
//datatype it will check
int i1=10;
byte b1=10; // Line 2 => Here Compiler accepting the value 10.
byte b2=i1; //Line 3 => CE
}
}
```

23)01-11-23 30:00

Searching and Sorting
Linear Search / Sequential Search
=Every index it has to go
Binary Search (array must be sorted)
=less number of comparisons
Eg: key = num;

YT : Najafi Code

Bubble Sort / Exchange Sort / Sinking Sort
Multiple steps

Merge Sort
Quick Sort
Selection Sort

Arrays class
Arrays.fill(arr,3)
Arrays.sort()

10)CodeSnippets

Q>
class Test{
public static void main(String[] args){

```
int x= 0;
do{
    ++x;
    System.out.print(x);//1 4 8
    if(++x<5)
        continue;
    ++x;
    System.out.print(x);//6 10
}while(++x<10);
}
```

JVM

x = 0,1,2,3,4,5,6,7,8,9,10,11

Output: 146810

Q>

```
class Test{
public static void main(String[] args){
for(int i =0;i++<5;i++){
    System.out.println("hello");//3 times printed
    i=i++;
}
}
}
```

How many times hello will be printed?

- A. 4
- B. 3
- C. 5
- D. 2

Answer: B

JVM

i =0,0<5(true) , i =1,2

i = 2,2<5(true), i =3,4

i = 4,4<5(true), i =5,6

i = 6,6<5(false), i =7

Q>

```
public class Test {
public static void main(String[] args) {
int i = 5;
System.out.print(i+++"");//5
}
```

```
System.out.print(i+"");//6
System.out.print(++i+"");//7
System.out.print(++i+i+++""");//8 + 8 = 16
}
}
```

Predict the Output?

- A. 56716
- B. 66616
- C. 57817
- D. 67817

JVM

i = 5,6,7,8,9

Answer : A

Q>

```
public class Test {
public static void main(String[] args) {
int[] a ={0,2,4,1,3};
for (int i=0;i<a.length ;i++ )
a[i] = a[(a[i]+3)/a.length];
System.out.println(a[1]);
}
}
```

Predict the Output?

- A. CompileTime Error
- B. Some problem occurred by jvm during execution
- C. 0
- D. 1
- E. 3

F. None of the above

Answer: F(a[1] =2)

Q>

```
public class Test {
public static void main(String[] args) {
int i =0,j=5;
for (;i<3) && (j++<10); ){
System.out.print(" " + i + " " +j);
i++;
}
System.out.print(i + " " +j);
}
```

}

}

Predict the Output?

- A. CompileTime Error
- B. Some problem occurred by jvm during execution
- C. 06172838
- D. 06172839
- E. None of the above

JVM

i = 0,j =6 , 0<3(true) && 5<10(true)

0 6

i =1, j= 7, 1<3(true) && 6<10(true)

1 7

i =2, j =8 , 2<3(true) && 7<10(true)

2 8

i = 3, j =8 3<3(false)

3 8

Answer: C

Q>

```
public class Test {  
    public static void main(String[] args) {  
        for (int i=0;i<10 ;i+=2 ){// i = i + 2  
            System.out.print(i);  
        }  
    }  
}
```

Predict the Output?

- A. CompileTime Error
- B. Some problem occurred by jvm during execution
- C. 02468
- D. 13579
- E. None of the above

Answer: C(print even numbers)

Q>

```
public class Test {  
    public static void main(String[] args) {  
        int x =5;  
        x*= 3*5 + 7*x-1 + ++x; // x *= 3*5 + 7*x -1 + ++x  
        = 3*5 + 7*5-1 + 6
```

```
= 15 + 35-1+6  
x* = 55  
x = x*5  
= 55*5 = 275  
System.out.println(x);  
}  
}
```

Predict the Output?

- A. CompileTime Error
- B. Some problem occurred by jvm during execution
- C. 330
- D. 275
- E. None of the Above

Answer: D

```
public class Test {  
public static void main(String[] args) {  
int a =3;  
switch (a){  
case 1: ++a;  
case 2: a++;  
case 3: a++;//4  
default: ++a;//5  
}  
System.out.println(a);//5  
}  
}
```

Predict the Output?

- A. CompileTime Error
- B. Some problem occurred by jvm during execution
- C. 7
- D. 5
- E. 6
- F. None of the Above

Answer: D

Q>

```
public class Test {  
public static void main(String[] args) {  
int i;  
for (i=0;i<6 ;i++ ){
```

```
if (++i>3)
continue;
}
System.out.println(++i);
}
}
```

Predict the Output?

- A. CompileTime Error
- B. Some problem occurred by jvm during execution
- C. 7
- D. 8
- E. 6
- F. None of the Above

i=1, 0<6(true), 1>3(false)

i=3, 2<6(true), 3>3(false)

i=5, 4<6(true), 5>3(true)

i=6, 6<6(false)

i = 7

Answer: C

24)2-11-22

Bubble Sort / sinking sort / exchange sort

Step 1 : largest number is pushed to last.

Step 2 : compare the first num with next num to it.

Problem: write a program to swap two numbers.

If you won't get it google it and learn

First approach is first learn programming language then problem solving

Mini Project Guesser Game

Simplified code

Enhance the project

Guesser
=guessNum //memory name
=guessNum()

Guess num ⇒ memory

Empire
=call guesser and says the guesser to guess num

Player
=number of players participate in the game
=they guesses the number
If correct then win
Or loss

String

int = primitive data type and reserved word
String = It is classname, String is basically an inbuilt class for which object can be created.

Package that string class present in.

Java
|=>lang
 |=>String

To represent the data in string then enclose it in double quotes
“Suryachandra”

String it refers to an object in java present in package called java.lang.String
String refers to collection of characters.

As per oops we can create object using new keyword

```
String s1 = "surya";  
System.out.println(s1); //surya
```

```
String s2 = new String("RamaNarayana");  
sysout(s2);
```

Strings two types

1. Immutable (no change)
2. Mutable (change)

```
1.String(C)
class String{
//instances variables

//methods
}
```

2.Mutable

1. StringBuffer
2. StringBuilder(1.5v)

In Java String objects are by default immutable , meaning once the object is created we can't change the object, if we try to change then those changes will be reflected on the new object not on the existing object.

```
String s1 = "surya"; //it is stored in heap area //s1 ref var in stack area
s1.concat("Chandra");//new obj is created but it don't have ref // then it is garbage obj, garbage
collector will collect it.
sysout(s1);//surya
```

```
StringBuilder s1 = new StringBuilder("surya");
s1.append("chandra"); // it is mutable then changes will happen in same memory
sysout(s1);
```

CASE 1 :

```
String s1 = "sachin";// stored "sachin" in heap and s1 points to it.
String s2 = new String("sachin");// stored "sachin" in heap and s2 points to it.
sysout(s1 == s2);//repect to operator      //false
sysout(s1.equals(s2)); //respect to method    //true
"==" is used to compare the references (is it pointing to same obj or different obj)
>equals() is used to check or compare the content
```

```
"sachin".equals("sachin");
```

CASE 2::

```
String s1 = new String("sachin");// stored "sachin" in heap and s1 points to it.
String s2 = new String("sachin");// stored "sachin" in heap and s2 points to it.
sysout(s1 == s2);//repect to operator      //false
```

```

sysout(s1.equals(s2)); //respect to method //true

StringBuilder s1 =new StringBuilder("sachin");// stored "sachin" in heap and s1 points to it.
    StringBuilder s2 = new StringBuilder("sachin");// stored "sachin" in heap and s2 points
to it.
    sysout(s1 == s2); //repect to operator //false
    sysout(s1.equals(s2)); //respect to method //false

```

.equals() method in stringBuilder acts are comparing referencing

CASE 3 ::

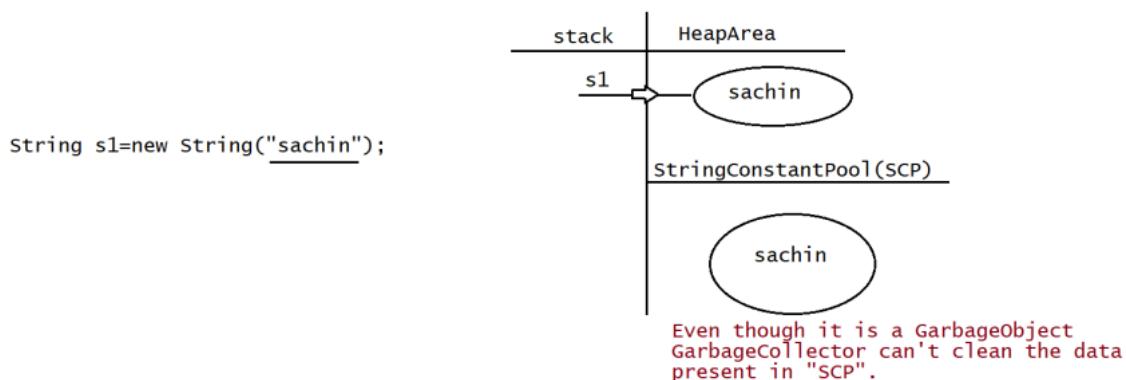
```

String s1 = new String("sachin");
//how many obj are created in above stmt
//two obj are created that
//whenever working with string type of data, it will be stored in heap area
// heap area stores it again in "string constant pool area"(SCP)

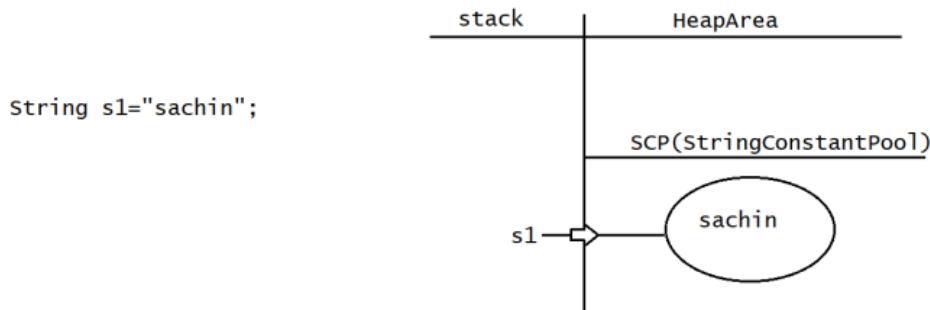
```

Whenever with new keyword then it will be stored in heaparea and SCP

But in SCP there will be no reference



When you work with string obj the special memory it will be stored for direct literals



For new keyword then two copies
If direct literal the only one copy that will be in scp

11)Code Snippets

2:48:00

ooooooo

2-11-22 dbt

Q>

sir how to take input from the user in the same line? If we write like this

```
Scanner scan= new Scanner (System.in);
System.out.println("Enter the value of a:");
int a=scan.nextInt();
System.out.println("Enter the value of b:");
int b= scan.nextInt();
then for b value it will go to next line
//var args will takes as many as variables
java FileName 10 20
Integer a = (Integer) arg[0]
Integer b = (Integer) arg[1];
```

Q> I pass the array to function that function simply only return what it will do i
got from one mcq

(14:00–)

```
class A {
psvm(String[] args){
int arr[] ={1,2,3,4,5};
m1(arr)
for(int elem: arr)
System.out.print(arr);//1 ,2 ,3,50,5
```

}

```
public static void m1(int[] arr){
arr[3] = 50;
return;
}
}
```

Q>

for (int i=0;i<10 ;i+=2) sir in for loop after boolean it is necessary

increment/decrement
but here initialize i+=2
for(stmt1;stmt2;stmt3){
//body of loop
}
stmt1=> any valid java statement but suggested in initialisation one
stmt2 => compulsorily boolean only
stmt3 => any valid java statement but suggested in increment/decrement

eg: i++, ++i, i = i+1, i+=1,i+=2

Q> sir first statement we can write sop(" hii")

Answer :yes

Q>how to define 2D ARRAY confusion in brackets?

int[] a ={10,20,30};

int[][] a = {

{10,20,30},

{40,50,60}

}

int[][][] a = {

{

{10,20,30},

{40,50,60}

},

{

{10,20,30},

{40,50,60}

}

};

Q>

how to declare array of class?

int[] arr =new int[5];//array of 5 integers

Student[] std=new Student[5];//array of 5 Students

Q>

what does in.ineuron.folder1.*(implicit import); mean? what I want to ask is, is it going to

add or load all the classes at folder1's level or it adds all the classes

let's say in.ineuron.folder1.abc;(explicit import)

```
in.ineuron.folder1.xyz;
```

Q> Stack extends Vector, we have addElement(Object e) from Vector class, also we have push(Object e) so what is the difference in these and which to use when?

what makes them different?

Answer: Code would result in CompileTime Error.

Q>

```
int i=1,j=10;
```

```
do{
```

```
if(i++>--j)// 4>6
```

```
continue;
```

```
}while(i<5);
```

```
sop(i+" "+j); // 5 6
```

sir what is the output?

JVM

i = 1,2,3,4,5

j = 9,8,7,6

Comparable and Comparator

```
=====
```

26)03-11-22

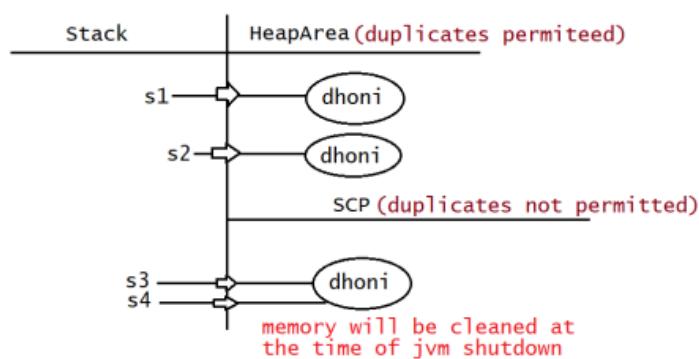
(String with Memory Map in Detail)

```
>set path=kkjfjkjkf;
```

CASE 4 ::

```
String s1=new String("dhoni");
String s2=new String("dhoni");
System.out.println(s1==s2); //false

String s3="dhoni";
String s4="dhoni";
System.out.println(s3==s4); //true
```

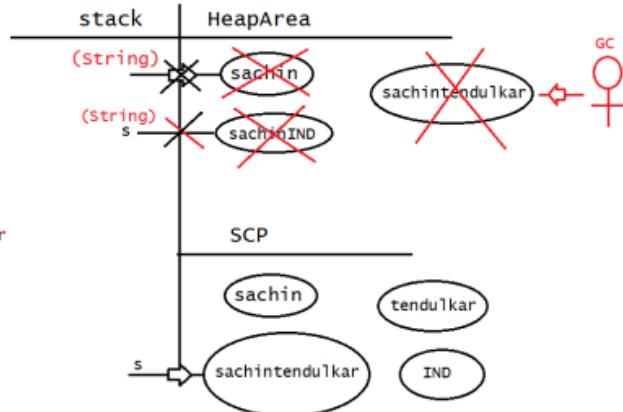


SCP, obj will be created if and only not same obj in scp

Scp (duplicates not permitted)

Memory will be cleared in SCP only when JVM shutdown.

```
String s = new String("sachin");
s.concat("tendulkar");
s=s.concat("IND");
s="sachintendulkar";
System.out.println(s); //sachintendulkar
```



Because of method call,

A method call will happen at runtime

Method call will be executed by JVM, change will be always created in heap area not SCP

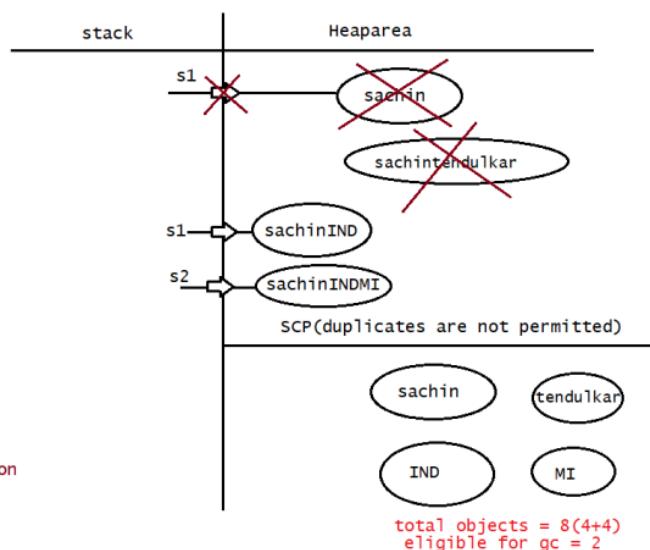
Output:: Direct literals are always placed in SCP, Because of runtime operation if object is required to create compulsorily that object should be placed on the Heap, but not on SCP.

CASE 5

```
String s1= new String("sachin");
s1.concat("tendulkar");
s1+="IND"; s1 = s1 + "IND";
String s2=s1.concat("MI");

System.out.println(s1); //sachinIND
System.out.println(s2); //sachinINDMI

String + "IND"
↑
addition
both operands are number type
addition
if one operand is string then concatenation
```



CASE 6

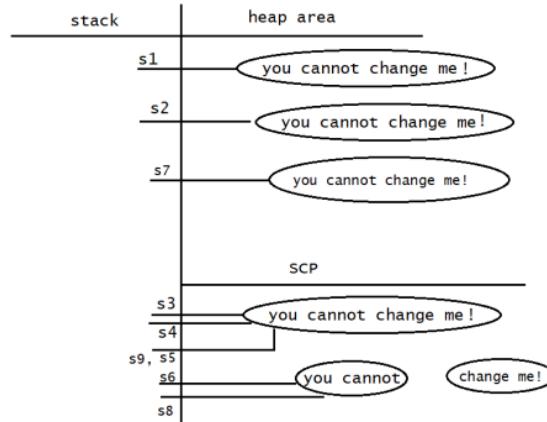
```
String s1=new String("you cannot change me!");
String s2=new String("you cannot change me!");
System.out.println(s1==s2); //false

String s3="you cannot change me!";
System.out.println(s1==s3); //false
String s4="you cannot change me!";
System.out.println(s3==s4); //true

String s5="you cannot " + "change me!"; //you cannot change me!
System.out.println(s3==s5); //true

String s6="you cannot ";
String s7=s6+"change me!"; //you cannot change me!
System.out.println(s3==s7); //false

final String s8="you cannot ";
String s9=s8+"change me!"; //you cannot change me!
System.out.println(s3==s9); //true
System.out.println(s6==s8); //true
```



S5 is + expr is evaluated by compiler

One var + one literal then jvm comes into picture(not compiler)

So it is created in heap area

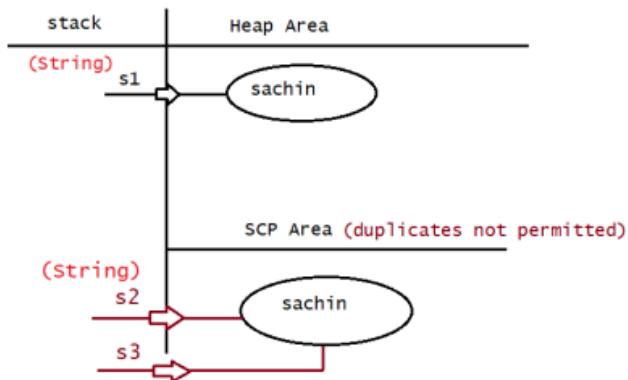
If one var is final the it is known to compiler

CASE 7 :: Interning

```
String s = new String("sachin");
//how to know whether data is in both heap area and scp or not
String s2 = s1.intern();
System.out.println(s1==s2);
```

```
String s1 = new String("sachin");
String s2 =s1.intern();
System.out.println(s1==s2); //false

String s3 = "sachin";
System.out.println(s2==s3); //true
```

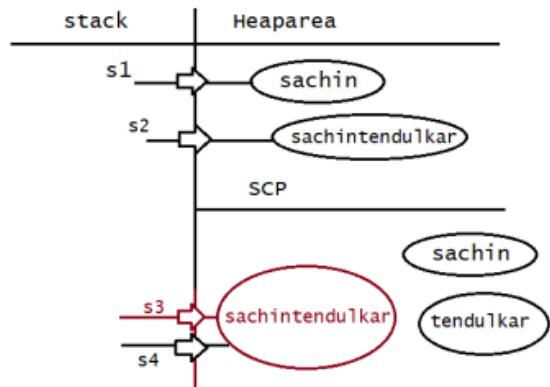


INTERN METHOD

```
String s1 = new String("sachin");
String s2 = s1.concat("tendulkar");

String s3 = s2.intern();

String s4 = "sachintendulkar";
System.out.println(s3==s4); //true
```



Intern method role is something to do in scp
If not present in scp
Then it will create at scp and points

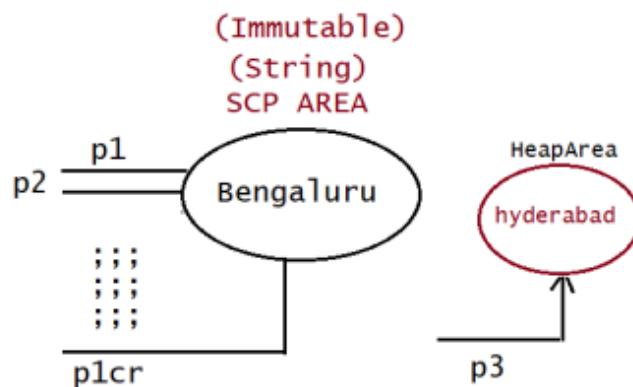
//Why concept of scp is available in Strings?
//why strings are classified into two types?

Need of scp

Eg: Aadhar application

Need of SCP

Adhar Card Application	
NAME	<input type="text"/>
FATHERNAME	<input type="text"/>
DOB	<input type="text"/>
CITY	<input type="text"/>
PHOTO	<input type="text"/>
<input type="button" value="submit"/>	



Name is for all users is not same
And there father's name is also not same

And if persons1 location is city : Bangalore

Instead of all other user, even for 1 core Bangalorian

Instead of separate location.

One obj and every one is pointed to it.

If p3 person change then address ref is just changed for only that person.

Importance of SCP

=====

1. In our program if any String object is required to use repeatedly then it is not recommended to create multiple object with same content
it reduces performance of the system and effects memory utilization.
2. We can create only one copy and we can reuse the same object for every requirement. This approach improves performance and memory utilization we can achieve this by using "scp".
3. In SCP several references pointing to same object the main disadvantage in this approach is by using one reference if we are performing any change the remaining references will be impacted. To overcome this problem sun people implemented immutability concept for String objects.
4. According to this once we creates a String object we can't perform any changes in the existing object if we are trying to perform any changes with those changes a new String object will be created hence immutability is the main disadvantage of scp

Public class Test

{

```
    Public static void main(String arg[])
    {
        //API level = String class is at api level
        //application programming Interface
```

//API means someone wrote the code and he will give .class file end users will use and take the benefits

```
        //String.class
```

```
        //Entire java we are learning as API only.
        //System.out.println("welcome to MyNotes");
        //println in method that is written by some one.
```

}

Note: if the class name and method name is same in a class then that method is called "Constructor".

String class Constructor

(=method name is same as that of the class name)

=====

String s =new String() => Creates an Empty String

Object

String s =new String(String literals) => Creates an Object with String literals on Heap

eg: String str = new String("sachin");

String s =new String(StringBuffer sb) => Creates an equivalent String object for StringBuffer

String s =new String(char[] ch) => Creates an equivalent String object for character array

String s =new String(byte[] b) => Creates an equivalent String object for byte array

Above all are constructor overloading

Same method and class name with diff parameters to methods

Character array

Public class Test

{

 Public static void main(String arg[])

 {

 Char[] ch = {'j','a','v','a'};

 String s1 = new String(ch);

 System.out.println(s1); //java

 System.out.println();

 byte[] b = {65,68,67,69};

 String s2 = new String(b);

 Sysout(s2); //ABCD

}

public class Test

{

 public static void main(String arg[])

 {

 StringBuffer sb = new StringBuffer("Sachin");

 sysout("StringBuffer data is :: "+sb);

```

String s1 = new String(sb);
System.out.println("String data is : "+s1);

}

```

To see API

`javap java.lang.String`

Important methods of String

- =====
- 1. public char charAt(int index)
- 2. public String concat(String str)
- 3. public boolean equals(Object o)
- 4. public boolean equalsIgnoreCase(String s)
- 5. public String subString(int begin)
- 6. public String subString(int begin,int end)
- 7. public int length()
- 8. public String replace(char old,char new)
- 9. public String toLowerCase()
- 10. public String toUpperCase()
- 11. public String trim()
- 12. public int indexOf(char ch)
- 13. public int lastIndexOf(char ch)

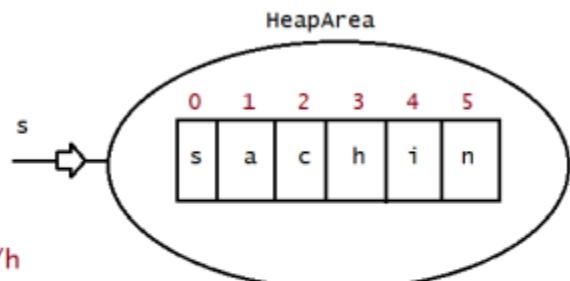
```

String s =new String("sachin");
System.out.println(s[3]);//CE

System.out.println(s.charAt(3));//h

System.out.println(s.charAt(-1));//StringIndexOutOfBoundsException
System.out.println(s.charAt(500));//StringIndexOutOfBoundsException

```



Note:

```

package java.lang;
class String

```

```

{
//method name
public int length(){
}
}

Integer Array Class(proxy class)
=====
class [I
{
//property name
//property is a variable of the obj
int length;
}

eg:
String s ="sachin";
System.out.println(s.length());//6
int[] arr= {10,20,30};
System.out.println(arr.length());//3

```

Instead of remember all this
Using eclipse ide we can see classes easily and use it.

One + operator behaves like addition and concationation.
It is like operator overloaded
It is only at jvm level.
We can't overload operators in java

12)CodeSnippets

2:44:00

Which among the following declarations is valid?

1. int[] a;
2. int a[];
3. int []a;
4. int[6] a;//can't specify the size

Predict the answer.

- A. 1,2,3
- B. 1,2,4
- C. 2,3,4
- D. None of the above

Answer: A

Q>

1. int[] a;
- a=new int[5];

2. int[] a =new int[5];

Both the declarations are they same?

A. yes

B. no

Answer: A

Q>

```
3. int n[][] = {{1,3},{2,4}};  
for(int i=n.length-1;i>=0;i--){  
for(int y:n[i])  
System.out.print(y);  
}
```

A. 1234

B. 2313

C. 3142

D. 4231

E. 2413

F. compilation error

G. Some problem by jvm at the runtime

Answer: E

```
Q> int nums1[] = {1,2,3};  
int nums2[] ={1,2,3,4,5};  
nums2 = nums1; // Compiler for array assignment compiler will check only the  
type not the length  
for(int x:nums2)  
System.out.print(x+":");
```

What is the result?

A. 1:2:3:4:5

B. 1:2:3:

C. Compilation fails

D. ArrayIndexOutOfBoundsException

Answer: B

```
Q> int data[] = {2010,2013,2014,2015,2014};  
int key = 2014;  
int count=0;  
for(int e:data){  
if(e!=key){  
continue;  
count++;  
}  
}
```

System.out.println(count+" found");

What is the result?

A. Compilation fails

B. 0 found

C. 1 found
D. 3 found
E. 2 found
data[0]= 2010
data[1]= 2013
data[2]= 2014
data[3]= 2015
data[4]= 2014
key = 2014,count= 0
e =2010

Answer: B(at any time count value will not change becoz of continue, or even if condition fails)

Q>

```
class Test{  
public static void main(String[] args){  
int numbers[];  
numbers =new int[2];  
numbers[0] = 10;  
numbers[1] = 20;  
numbers = new int[4];  
numbers[2] = 30;  
numbers[3] = 40;  
for(int x: numbers)  
System.out.print(" " + x);  
}  
}
```

What is the result?

- A. 10 20 30 40
- B. 0 0 30 40
- C. Compilation fails
- D. An exception is thrown at runtime

Answer: B

Q>

```
int wd = 0;  
String days[] = {"sun","mon","wed","sat"};  
for(String s:days){  
switch(s){  
case "sat":  
case "sun":  
wd-=1;  
break;  
case "mon":  
wd++;  
case "wed":
```

```
        wd+=2;
    }
}
```

What is the result?

- A. 3
- B. 4
- C. -1
- D. compilation fails

JVM

====

```
days[0] = "sun", days[1] = "mon", days[2] = "wed", days[3] = "sat"
wd = 0, s = "sun", change wd = wd-1 = 0-1 = -1
wd = -1, s = "mon", wd = -1+1 = 0, wd = 0+2 = 2
wd = 2, s = "wed", wd = 4
wd = 4, s = "sat", wd = 4-1 = 3
```

Answer: A

Q>

```
=> concat(String)
=> String object by default Immutable(changes made will reflect in new memory)
String[] str = {"A", "B"};
int idx = 0;
for(String s: str){
    str[idx].concat(" element " + idx);
    idx++;
}
//using for loop
for(idx = 0; idx < str.length; idx++){
    System.out.print(str[idx]);
}
```

What is the result?

- A. AB
- B. A element0B element1
- C. A NullpointerException is thrown at runtime
- D. A 0B 1
- E. Compilation Error

Answer: A

Q>

```
int[][] arr = new int[2][4];
arr[0] = new int[]{1, 3, 5, 7};
arr[1] = new int[]{1, 3};
for(int[] a : arr){
    for(int i:a){
        System.out.print(i + " ");
    }
}
```

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

A. compilation fails

B. 1 3

1 3

C. 1 3

followed by AIOBE

D. 1 3

1 3 0 0

E. 1 3 5 7

1 3

F. 1 3 5 7

1 3 0 0

27)04-11-22

(String in java and mutable string)

Discussed methods are

Intern

Equals

charAt

length()

Concat

```
String s = "java";
```

```
sysout(s.equals("java")); //TRUE
```

```
sysout(s.equalsIgnoreCase("JAVA")); //TRUE
```

```
substring(int begin);
```

```
substring(int begin,int end);
```

Eg: credentials(gmail username not case sensitive)

Substring = means somepart of the string.

Replace

```
String name = "sbchin";
```

```
Sysout(String data = name.replace('b','a'));
```

```
String data = "ababab";
```

```
sysout(data.replace('a','b'));
```

```
//it replaces the every occurrences
```

```
String name = "sAchIn"; //mixedCase
```

```
sysout(name.toUpperCase());//SACHIN
```

```
sysout(name.toLowerCase());//sachin
```

```
String name = "Sachin IND";
sysout(name.length());
```

```
String name = " IND ";
sysout(name.trim());
sysout(name.trim().length());
```

```
String name = "hyderAbbas";
sysout(name.length());
sysout(name.indexOf('A'));//5
sysout(name.indexOf('a'));//8
sysout(name.indexOf('z'));//-1
sysout(name.lastIndexOf(char ch));
```

If found then break at that particular index.

Predict the output

=====

Q>

```
String s1="sachin"; // s1,s3 -> sachin (scp)
String s2=s1.toUpperCase(); // s2->SACHIN(heap area)
String s3=s1.toLowerCase();
System.out.print(s1==s2);//false
System.out.print(s1==s3);//true
```

```
class Student
{
    String name = "sachin";
    int id = 20;
}
```

IMP

```
public string toString();
```

Note: whenever we print any reference , by default JVM will call `toString()` on the reference.

```
public class Test{
public static void main(String arg[])
{
Student std = new Student();
sysout(std); //Student@HexaDecimalValue
sysout(std.toString()); //Student@HexaDecimalValue
//whenever an object is called then toString method is called.
String name = new String("dhoni");
```

```

sysout(name);//dhoni
sysout(name.toString());//dhoni
}}

```

Return type of two string

```

public String toString()
{
    return "";
}

```

Q>

```

String s1="sachin"; // s1,s2-> sachin (SCP)
//goes to s1 and it is already in the string so nothing changes
String s2=s1.toString();
System.out.print(s1==s2);//true

```

Q>

```

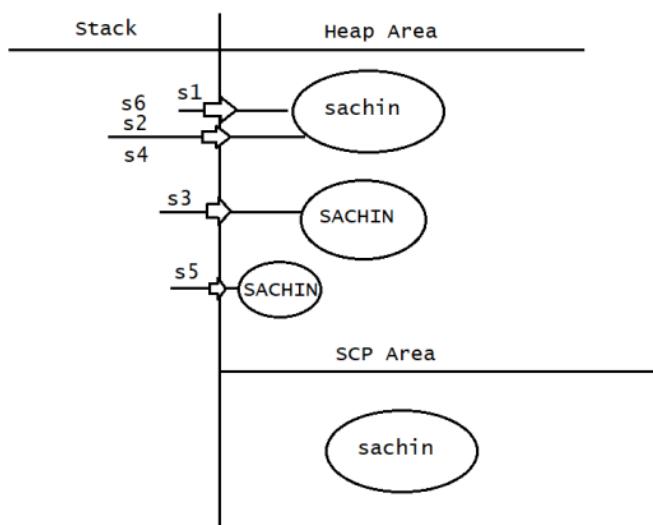
String s1=new String("sachin");
String s2=s1.toString();
String s3=s1.toUpperCase();
String s4=s1.toLowerCase();
String s5=s1.toUpperCase();
String s6=s1.toLowerCase();
System.out.print(s1==s6);//true
System.out.print(s3==s5);//false

```

```

Q>
String s1=new String("sachin");
String s2=s1.toString();
String s3=s1.toUpperCase();
String s4=s1.toLowerCase();
String s5=s1.toUpperCase();
String s6=s1.toLowerCase();
System.out.print(s1==s6); //true
System.out.print(s3==s5); //false

```

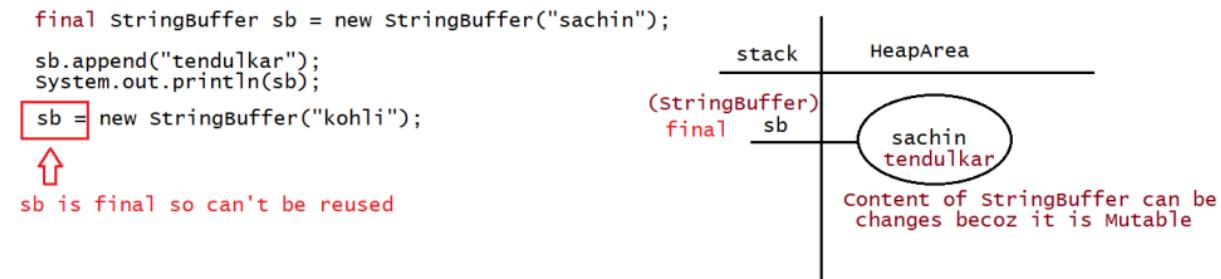


Concept of Mutability

```
StringBuffer sb = new StringBuffer("sachin");
sb.append("tendulkar");
System.out.println(sb); //sachintendulkar
```

```
Int a = 10;
a+=10;
System.out.println(a);
```

```
final StringBuffer sb = new StringBuffer("sachin");
sb.append("tendulkar");
System.out.println(sb);
```



Because of final reference can't be pointed to other.

final vs Immutability(1:59:00-)

=> final is a modifier applicable for variables, whereas immutability is applicable only for Objects.

=> If reference variable is declared as final, it means we cannot perform reAssignment for the reference variable,

it does not mean we cannot perform any change in that object.

=> By declaring a reference variable as final, we won't get immutability nature.

=> final and Immutability is different concept.

eg:: final StringBuilder sb=new StringBuilder("sachin");
sb.append("tendulkar");
System.out.println(sb);
sb=new StringBuilder("dhoni"); //CE::Cannot assign a value to final
variable

Note::

final variable(valid),
final object(invalid),
immutable variable(invalid)
immutable object(valid)

`StringBuilder, StringBuffer` are by default mutable.

All Wrapper classes(`Byte, Short, Long, Integer, Float, Double, Boolean, Character`) are by Default Immutable.

Mutable -> can be changed

Immutable => can't be change

(-2:10:00)

Important methods of `StringBuffer/StringBuilder`

a. public int length()

b. public int capacity()

c. public char charAt(int index)

d. public void setCharAt(int index,char ch)

e. public StringBuffer append(String s)

f. public StringBuffer append(int i)

g. public StringBuffer append(long l)

h. public StringBuffer append(boolean b)

i. public StringBuffer append(double d)

j. public StringBuffer append(float f)

k. public StringBuffer append(int index,Object o)

l. public StringBuffer insert(int index,String s)

m. public StringBuffer insert(int index,int i)

n. public StringBuffer insert(int index,long l)

o. public StringBuffer insert(int index,double d)

p. public StringBuffer insert(int index,boolean b)

q. public StringBuffer insert(int index,float f)

r. public StringBuffer insert(int index,Object o)

public StringBuffer delete(int begin,int end)

public StringBuffer deleteCharAt(int index)

public StringBuffer reverse()

public void setLength(int Length)

public void trimToSize()

public void ensureCapacity(int capacity)

>javap java.lang.StringBuffer

```
{
```

```
StringBuffer sb1 = new StringBuffer();
```

```
sysout(sb1.length()); //no of characters stored = 0
```

```
sysout(sb1.capacity()); //how many no of characters can be stored ? // 16
```

```
sb1.append("abcdefghijklmnp");
sysout(sb1.length()); //16
sysout(sb1.capacity()); //16

sb1.append("q");
sysout(sb1.length()); //16
sysout(sb1.capacity()); //newCapacity = (oldcapacity+1)*2 //34
}

//StringBuffer sb2 = new StringBuffer(100);
sysout(sb2.length());
sysout(sb2.capacity());

//StringBuffer sb3 = new StringBuffer("sachin");
sysout(sb3.length());
sysout(sb3.capacity());
```

28)05-11-22 dbt

29)07-11-22

(Mutable String and String Programming)

a.StringBuffer

b.StringBuilder

For both of the above has the same

StringBuffer

setCharAt()

Append is like concat

```
StringBuffer sb = new StringBuffer();
sb.append("The value of PIE is: ");
sb.append(3.1414);
sb.append(true);
sysout(sb);
```

check

Public StringBuffer append(int index, Object o);

```
StringBuffer sb = new StringBuffer("abcdefgh");
sb.insert(2,"xyz");
sb.insert(9,11);
System.out.println(sb); //abxyzcdef11gh
sb.insert(sb.length(),"dhoni");
System.out.println(sb);
```

Delete

deleteAt

StringBuffer vs StringBuilder

(1.0jdk) (1.5jdk)

Memory is most important
And memory is costly operation
sb.ensureCapacity(1000);

Various method is StringBuffer

Q>Why does stringBuilder come when StringBuffer exists already ?

But StringBuilder is came later
Based on the demands of the developers
1.5jdk ver is the game changer
>javap java.lang.StringBuffer
=has the synchronized in StringBuffer but it is not there in StringBuilder
Only this one is different
Along with return type access modifier is attached
That is synchronized
This concept present in Multithreading

Many people can use the application simultaneously by using the concept of StringBuilder.(it is thread safe)

But it is not possible in stringBuffer.

When do you use String , StringBuffer and StringBuilder ?
Content is fixed and it won't changes frequently.

Content is not fixed and it changes frequently and not worried about performance.thread safety is required.

Content is not fixed and it changes frequently but ThreadSafety is not required

Now a days applications are stringBuilder
For banking applications that are stringBuffer.

METHOD CHAINING //concept

```
String name = "sachin";
String data = name.toUpperCase();
Int count = data.length();
System.out.println(count);

//method chaining
System.out.println(name.toUpperCase().length());

StringBuffer sb = new StringBuffer("virat");
sb.append("kohli").insert(10,"anushka").reverse().append("IND").insert(sb.length(),"RCB").revers
e().delete(0,6);
System.out.println(sb);
```

>HOW CHAINING IS DONE ?

Memory stack is done through this reference.

Method chaining is important concept in microservices and springboot this is only done completely..

TO GET COMFORTABLE AT ARRAYS AND STRINGS CONCEPT
DO PROBLEM SOLVING ON IT.

- 1)copyString to another string
- 2)and reverse the string using the logic and method

The return type of method and the object has to be same then only it can be invoked.

1:25:00-

13)CodeSnippets

30)08-11-22

(String Programming, Encapsulation , Github Eclipse Integration)

anagram,pangram,palindrome

(06:00- 1:06:00)

1:16:00 OOPs

Github integration with eclipse

OOP's

Java is object oriented only but not pure object oriented.

But pure object oriented can be made through wrapper classes

1. Encapsulation/Privacy or security /Data Hiding and dataBinding / providing controlled access to data members
(private, setters, getters , this keyword and shadowing problem)
2. Inheritance promotes the code reusability
3. Polymorphism promotes code flexibility
4. Abstraction implementation hiding, only features are visible

Encapsulation

//is also called as data hiding

```
Class Student
{
    //instance variables//Data Members //fields //properties
    private int age;
    private String name;
    private String city;
    //to provide the security to instance variables from direct //access to other class then use
the private

    //activity
    //setter= taking value from outside class and setting value to instance variable is called
    setter method.
    //please give name as set
```

```

//that should not return anything (void)
    void setAge(int a)
{
    if(a>0)
        age = a;
    else
        sysout("invalid age");
}

//to use the data
//using getter arrangement they can get the data
//getter should not accept the any inputs or parameters
    int getAge()
    {
        return age;
    }
    void setName(String x)
    {
        Name = x;
    }
    String getName()
    {
        return name;
    }
    Void setCity(String y)
    {

    }

}

public class LaunchEncap{
    public static void main(String[] arg)
{
    //instansionation = means creation of object
    //Student st = new Student();
    //St.age = 28;
    //St.name = "sid";
    //St.city = "Bengaluru";
    st.setAge(28);
    int age = st.getAge();
    sysout(age);
    //encapsulation means providing data hiding to member of class for security and not to
do direct access.
}

```

}

Eclipse Integration

Send the local project to github server

1. Right Click on project
2. Team
3. Share project
4. Configure git repository—select local repo
5. Git staging to unstaged by clicking ++
6. Then add message
7. Click on commit and push
8. User name and password setup
9. Click on preview>>preview >>push
10. completed

Cloning

1. Open project and copy link
2. Eclipse → open perspective → Git
3. Select clone option
4. Give project link
5. Click next next
6. Give local destination → browse
7. Click on finish

Right click on project → import projects → click on finish → expand and working tree→Project folder >> change code and do the pushing git process the it will be pushed

14)Code Snippets

(2:45:00–

31)9-11-22 dbt

32)09-11-22

(encapsulation, Constructor , This vs This())
>source→ generate setters and getters

setAge

Above Age is the property

Syntax of setter and getter : known

class LaunchStudent

{

 private char GradeClass;

 private int id;

 private String name;

 public char getGradeClass() {

 return GradeClass;

 }

 public void setGradeClass(char gradeClass) {

 GradeClass = gradeClass;

 }

 public int getId() {

 return id;

 }

 public void setId(int id) {

 this.id = id;

 }

 public String getName() {

 return name;

 }

 public void setName(String name) {

 this.name = name;

 }

}

public class Student {

 public static void main(String[] args) {

 LaunchStudent std = new LaunchStudent();

 std.setName("harish");

 System.out.print("std.getName());

 std.setName("rana");

 System.out.print("std.getName());

 }

}

==recommend to use it with public access specifier==

Recommendation whenever it is boolean

```
public boolean getMarried()
{
    return married;
}
```

Recommendation is instead of getMarried use isMarried.

```
public boolean isMarried()
{
    return married;
}
```

Bean

= if all members in a class are private then that class is called bean
 That is known as achieving encapsulation

Return can return only one value

this keyword uses to currently running object

```
class LaunchStudent
{
    private char GradeClass;
    private int id;
    private String name;

    public char getGradeClass() {
        return GradeClass;
    }
    public void setGradeClass(char gradeClass) {
        GradeClass = gradeClass;
    }
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
```

```

        this.name = name;
    }

}

public class Student {

    public static void main(String[] args) {
        LaunchStudent std = new LaunchStudent();
        std.setName("harish");
        System.out.print("std.getName()");

        std.setName("rana");
        System.out.print("std.getName()");
    }

}

```

Constructor

//constructor is the specialized having the same name as of the class name used instead of common setter also.

//it does not have an explicitly return type.

//parameter rules hold the constructor as that of the like methods.

//constructor is invoked or called automatically when the object is created.

//whenever you want to execute at the time of object instantiation or creation, constructor is used.

//constructor won't returns anything

//explicitly calling is required, whenever obj created then it is called.

```

class Employee{
    private String name;
    private int empld;
    private float salary;

    //constructor
    public Employee(String name, int empld, float salary)
    {
        this.name = name;
        this.empld = empld;
        this.salary = salary;
    }

    public String getName() {
        return name;
    }
}

```

```

    }
    public int getEmpld() {
        return empld;
    }
    public float getSalary() {
        return salary;
    }
}

public class Constructor {
    public static void main(String[] args) {
        Employee emp = new Employee("ram", 12321, 24444.5f);

        System.out.println(emp.getName());
        System.out.println(emp.getEmpld());
        System.out.println(emp.getSalary());
    }
}

```

If constructor is not written,
Then jvm will include the default constructor behind the scenes.

Default constructor

=Default constructor is zero parameterized.

Parameterized constructor

When parameterized constructor is specified the default constructor will be gone.
Then we need to write the default constructor.

Javac compiler will include,
Jvm will execute

Constructor Overloading

=Having multiple constructors within a class with different parameters

In every constructor the first line is Super().

super() is behind the scene
super() method will call the parent class constructor
Implicitly every class parent is Object class
Javac compiler will include super() method.

In the first line there is either super() or this() only one.
super() or this() has to be in first line only

Constructor Chaining

=calling one constructor to the another constructor
this() method will call constructor of same class.

Stack Overflow

Stack area
disp() method is called
Within disp() again disp() method
It will be in recursion then the memory of stack will overflow.

Program

```
class Dog
{
    Name;
    cost;
    Dog()
    {
        this("sheru",9999);
    }
    Dog(String name, int cost)
    {
        this.name = name;
        this.cost = cost;
    }
    //getters
}
main()
{
```

```
Dog d1 = new Dog();
Dog d2 = new Dog("robo",1000);
}
```

```
=====
```

Method chaining vs constructor chaining

```
add()
{
disp()
}
disp()
{
}

Student()
{
this("hey",10);
}
Student(String name, int rollno)
{

}
=====
```

this vs this()

super vs super()

Method vs constructor

```
=====
```

Connect ot DB at the time of obj creation

```
class DBConnectivity
{
    public DBConnectivity()
    {
        System.out.println("Connect to database");
    }
}
class Test
{
    public static void main(String arg[])
    {
```

```
DBConnectivity db = new DBConnectivity()  
}  
}
```

15)CodeSnippet

(2:37:00 -

33)10-11-22

(static keyword)

Static

Inside class we can have

```
class  
{  
    Static var(heap area)  
    Static block(heap area)  
    Static method  
  
    Non static var (instance var)  
    Java block  
    Non static method  
}
```

Static var can be accessible to static block, static method,java block and non static method.

Instance var can be accessible to java block and non static method

But static block and static method can't access the instance vars

Instance variables are object dependent.

Sequence of execution inside java class

1. Static variables
2. Static block
3. Static method

4. Instance variables
5. Non static block
6. constructor
7. Methods

JVM ARCHITECTURE

With in jvm

1. Class loader subsystem(loads class)
 - a. Loading
 - i. Bootstrap loader (general class , api class)
 - ii. Extension loader (we written classes)
 - iii. Application loader (we written classes)
[bottom-up approach app to boot]
 - b. Linking
 - i. Verify stage (byte code verifier)
 - ii. Prepare stage (check are there any static var then on heap area it is allocated and default values are given)
 - iii. resolve
 - c. Initialization
 - i. Initialization stage (static block will be executed)
2. Data areas
 - a. Method area
 - b. Heap area(objects)
 - c. Stack area(ref var's,)(execution)
 - d. PC registers(every stmts have address)
 - e. Native Method Stack
3. Execution Engine(JIT, native references)
 - a. Interpreter(main method to 0's and 1's) , (converts 0's and 1's)
 - b. JIT Compiler
 - i. Intermediate code generator
 - ii. Code optimizer
 - iii. Target code generator

Profiler

GARBAGE COLLECTION

NATIVE METHOD INTERFACE(JNI)

NATIVE METHOD LIBRARY

Static var and block gets executed during class loading only.

When you create obj 3 things will happens

1. Memory of instance vars will be allocated inside obj
2. Instance initial block executed.(java block)
3. constructor will be executed.

Behind the scene constructor will include the java block inside it.

After super() only java block is executed

To count no; of obj created in a program

Static variable points

Static method vs non static method

1. Static method can be called using class reference
 2. Static method can also be called using object reference
 3. Static method is object independent
-
1. Non static method can't be called using class reference
 2. Non static method can only be called using object reference
 3. Non static methods are object dependent

16)CodeSnippets

(2:37:00-

34)11-11-22

(static and Inheritance)

Static variables vs Instances variables

Both memory will be in heap area

Core java replica application for farmer to give loans

=(static vars vs instance vars)

Principle amount

Time

Rate of interest (static)(common)

Simple interest

Principle amount for different farmer will be different
It is instance var

//if we write all activities in one place. Then there is not modularity

```
import java.util.Scanner;

class Farmer
{
    //Bean
    private float pa;
    private float td;
    private float si;
    private float ri;

    //activity to ask farmer for principle amount and time
    void input()
    {
        Scanner scan = new Scanner(System.in);
        System.out.println("Kindly, Enter the principle amount : ");
        pa = scan.nextFloat();

        System.out.println("Enter required time duration : ");
        td = scan.nextFloat();

        ri = 2.5f;
        //scan.close();
    }

    void compute()
    {
        si = (pa*td*ri)/100;
    }

    void disp()
    {
        System.out.println("simple interest is "+si);
    }
}

public class LaunchLoan {

    public static void main(String[] args) {
        Farmer f1 = new Farmer();
```

```

Farmer f2 = new Farmer();

f1.input();
f1.compute();
f1.disp();

f2.input();
f2.compute();
f2.disp();

}

```

//in above code all variables are instance var's only
//but for ri , u can make static because it is same for all

```

import java.util.Scanner;

class Farmer
{
    //Bean
    private float pa;
    private float td;
    private float si;
    private static float ri;

    static
    {
        ri= 2.5f;
    }

    //activity to ask farmer for principle amount and time
    void input()
    {
        Scanner scan = new Scanner(System.in);
        System.out.println("Kindly, Enter the principle amount : ");
        pa = scan.nextFloat();

        System.out.println("Enter required time duration : ");
        td = scan.nextFloat();

    }

    void compute()

```

```

{
    si = (pa*td*ri)/100;
}

void disp()
{
    System.out.println("simple interest is "+si);
}
}

public class LaunchLoan {

    public static void main(String[] args) {
        Farmer f1 = new Farmer();
        Farmer f2 = new Farmer();

        f1.input();
        f1.compute();
        f1.disp();

        f2.input();
        f2.compute();
        f2.disp();
    }
}

//above code
//Memory is saved by using once (static) (heap area)

```

Difference between static var vs instance var

Generic vars, class vars , static vars are the same
Object level , instance , fields are the same

Static methods are called as generic methods
Non static methods are called as specific methods

Inheritance

(45:00-

=establishing the relationship between classes and interfaces using inheritance

=by using extends keyword

Code Reusability.

Java supports relationships

1. is-A
 - a. Parent-child
 - b. base/super -sub
 - c. Is-a
 - d. existing-derived
2. has-A
 - a. Dependency injection
 - b. Aggregation

1.is-a is achieved using extends keyword

```
class Demo
{
    String name="Srinivas";
    int age = 47;

    void disp()
    {
        System.out.println("Demo "+name+" "+age);
    }
}
class Demo2 extends Demo
{

}
public class LaunchIn1 {

    public static void main(String[] args) {
        Demo2 d = new Demo2();
        d.disp();
    }
}

//properties and behaviors of demo got inherited to demo2.
def=inheritance refers to the process of one class acquiring the properties and behaviors of the another class.
```

Taking class is called as sub,child,derived class

Key Points of inheritance

1. Single inheritance is allowed
= one class can extend another class
2. Object class is parent of all class
3. Multilevel inheritance is allowed
=grandFather⇒father extends grandFather⇒son extends father
4. Hierarchical Inheritance
=one parent can have multiple childs.
5. Hybrid Inheritance
=combination of multilevel and hierarchical inheritance
6. Multiple inheritance is not allowed.
=can't have two parents
=one child having multiple parents is the problem, then compiler is ambiguous
=it lead to diamond shaped problem

One can extend only one class at a time.

One can't extend more than one class.

7. Cyclic inheritance is not allowed.

Using concept of Interface something similar can be done.

Private members of a class doesn't participate in inheritance
(to preserve encapsulation)

Eg:

```
class Father
{
    private String smoke;

    void disp()
    {
        System.out.println("smoking is bad habit");
    }
}
class Son extends Father
{
    smoke = "gold flake";//private mem's won't participate//CE
}
public class PrivateMemWont {

    public static void main(String[] args) {

    }
}
```

constructor will not participate in inheritance.

At Least one constructor has super() method, otherwise it leads to recursion.

MEMORY MAP + INHERITANCE + SUPER() AND THIS() + constructor

(java classes are like chain.)

=creating object of child is also creating of parent.

```
class Parentt
{
    int a, b;

    Parentt()
    {
        a=10;
        b=20;
        System.out.println("Parentt Const");
    }

    Parentt(int a, int b)
    {
        this.a=a;
        this.b=b;
        System.out.println("Parentt para Const");
    }
}

class Childd extends Parentt
{
    int x, y;

    Childd()
    {
        this(111,222);
        x=100;
        y=200;
    }
}
```

```

Childd(int x, int y)
{
    this.x=x;
    this.y=y;
}
void disp()
{
    System.out.println(a);
    System.out.println(b);
    System.out.println(x);
    System.out.println(y);

}
}

public class LaunchConst {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Childd ch=new Childd();
        ch.disp();
        Childd ch1=new Childd(1000,2000);
        ch1.disp();

    }
}

```

this() is used to call the constructor of same class constructor.

super() is used to call the constructor of parent class constructor.

How constructor will not involved in inheritance?

constructor means same name that of the class name, illogical tha it will participate in inheritance

17)CodeSnippets

(2:41:25–

35)12-11-22 dbt

36)14-11-22

(inheritance(inherited vs overridden vs specialised methods) , super ,final keyword in java)(15:00–

Plane animal loan

cargo plane tiger home loan

Loans has common are personal, car, home, Education

Access Specifiers

There are 4 access specifiers in java

1. public
2. protected
3. default
4. private

This are applied to class, method , constructor, variables

Whenever a project is developed there are multiple functionality

Eg : calculator = addition , subtraction, multiplication

Different classes are made.

Folder technically called as packages

One project has many packages.

Within a class	outside a class	outside package (is-a)	outside package
Yes	yes	yes	yes
Yes	yes	yes	no
Yes	yes	no	no
Yes	no	no	no

Visibility increases from private to public

Strongest access specifier public

Recommendation for methods and constructor to use public

Something you are inheriting from parent

Parent has eye,nose,height,color

Child has eye and nose are modified then that are called as overridden methods

Child has same height and color then its called inherited method;

Child has additionally smoking and drinking then that are specialized methods

UML

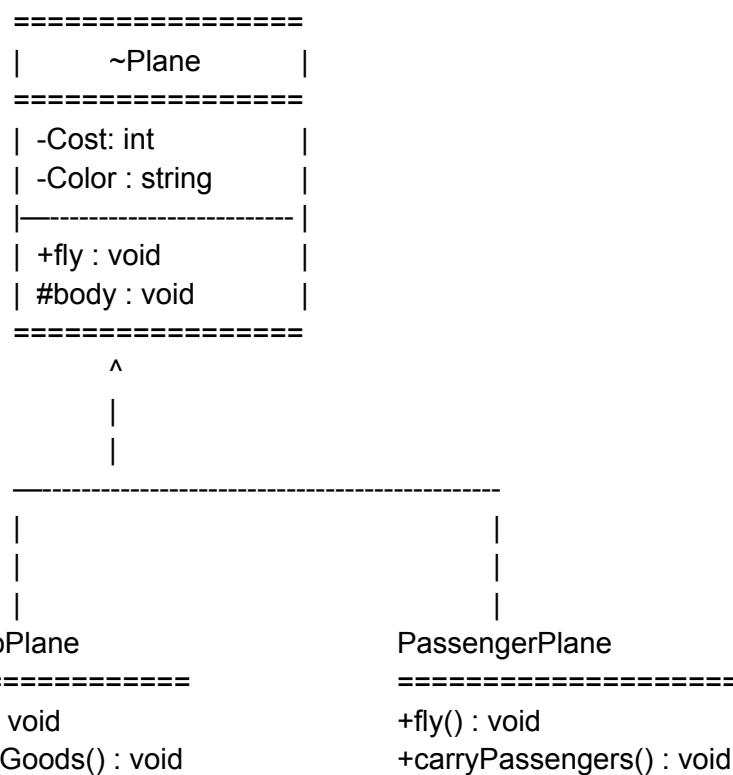
=pictorial representation of what code you want to write

+ Means public

means protected

~ default

- private



Fly is the overridden method
carryGoods and CarryPassengers are the specialized methods

```
class Plane
{
    Private int cost;
    private String color;

    Public void fly()
    {}
    Protected void land()
    {}
}
```

Where ever arrow is pointing that is parent
Solid line is child

Inherited , Overridden , Specialized method

Build a project of loans , :personalloans, educational loans, home loans , car loans.
Rate of interest is different for all(overridden)
(specialized methods)

Principle , Time

RULE 1 :

=we can't reduce the visibility of overridden method,but we can increase it.

```
class Loan
{
    void disp()
    {
        System.out.println("Welcome to INEURON BANK");
    }
}

class PersonalLoan extends Loan
{
    void disp()
    {
        System.out.println("Personal loan app");
    }
}
```

```
}
```



```
public class LaunchLoan
{
    public static void main(String[] args)
    {
        PersonalLoan pl=new PersonalLoan();
        pl.disp();
    }
}
```

Major inbuilt methods are public

If you want to overridden `toString()` then you can't reduce visibility.

RULE 2 :

Return type cannot be changed.

It is possible that it change change

That is called as co-varient return type.

Rule 3:

Return type of overridden method in child class can be different as that of parent if it is **co-varient return type**(return type is-a relation).

(Watch the return types carefully and try to understand)

```
class Interest
{
}

class InterestPersonalLoan extends Interest
{
}

class Loans
{
    public Interest interest()
```

```

    {
        Interest it=new Interest();
        return it;
    }
}

class PersonalLoan extends Loans
{
    public InterestPersonalLoan interest()
    {

        InterestPersonalLoan ipl=new InterestPersonalLoan();
        return ipl;
    }
}

public class LaunchLoans {

    public static void main(String[] args) {
        // TODO Auto-generated method stub

    }
}

```

Rule 4:

Parameters of overridden method must be same as that of parent else it will be considered as specialized method.

Considering method overloading

super()

=it is present inside constructor at first line.

=it calls the parent class constructor

super keyword

= super keyword calls the instance variables of parent class.

Eg: super.age

Class Demo

```
{
```

```
    Int age = 25;
```

```
}
```

class Demo2 extends Demo

```
{
```

```
    Int age = 28;
```

```
    sop(super.age);
```

```
}
```

(developers believe on try and check)

final Keyword

Final you can apply to class, method and variable.

Final applied to class

=final class does not participate in inheritance

If a class is final, then it can't be inherited.

Final applied to method

=final method will gets inherited.

But it can't be overridden

Final applied to variable

=final var acts as constant, it can't be changed later.

Eg: final double PI = 3.14;

```
//final class Vehicle
//{
//    void disp()
//    {
//        System.out.println("vehicle");
//    }
//}
```

```

//class Car extends Vehicle //final class we cannot inherit
//{
//    // final class doesn't participate in inheritance
//}
class Vehicle
{
    int i=10;
    final void disp()
    {
        i=20;
        System.out.println(i);
        System.out.println("vehicle");
    }
}
class Car extends Vehicle
{
//    void disp() final will get inherited but we cannot override
//    {
//        System.out.println("disp");
//    }
}

public class LaunchFinalK {

    public static void main(String[] args) {

        Car c=new Car();
        c.disp();

    }
}

```

```

class Demo1
{
    void disp()
    {
        System.out.println("Demo in Parent");
    }
}

```

```
public void disp2()
{
}

public int add()
{
    return 10+2;
}

public int add(int a, int b)
{
    return a+b;
}

}

class Demo2 extends Demo1
{
    public void disp()// we can increase visibility
    {

    }

//    void disp2() we cannot reduce visibility
//    {
//    }

//    public void add() return type cannot be changed
//    {
//        System.out.println("Child");
//    }

//    public int add(int a, int b)
//    {
//        return a+b;
//    }

    public int add(int a)
    {
        return a;
    }
}
```

```
public class LaunchRulesOR {

    public static void main(String[] args) {
        // TODO Auto-generated method stub

    }

}
```

Inside interface all variables are by default \Rightarrow public final static

Private variables and methods will not participate in inheritance.

CONCEPT OF METHOD HIDING (STATIC METHODS)

=will it participate in inheritance ?

=if it is participating can we over ride ?

18)CodeSnippets

(2:37:00–

37)15-11-22

(Polymorphism and Abstract Keyword)

Polymorphism

Compile time polymorphism it refer to method overloading

=by considering no; of parameters and their order.

Run time polymorphism (true polymorphism)

Method overriding

- flexibility
- code size reduces

We have two child classes

Type of ref are Childs are Child1 and Child2

Whatever is the obj then same type of that obj.

=above type of doing is called the **tight coupling**.

```
Child1 c1 = new Child1();
//Parent ineuron = new Child(); //no errors
//through above code we can achieve the polymorphism
```

Once case type can be changed. If parent type ref.

Parent ref;

=ref is used to obj type value;

ref = c1;

```
ref.cry(); //same line doing multiple tasks // 1:M
```

```
ref = c2;
ref.cry(); //1:M
```

Is this actual way of achieving polymorphism

Whenever an obj is created then that ref type has to be parent type then only we can achieve the polymorphism this is only called as loose coupling

Cp and pp are ref that holds the address of Cargo and Passenger

That cp and pp are passed to Plane

As

```
Plane plane;
plane = cp;
plane.takeOff();//1:M
plane.fly();
plane.landing();
```

```
plane = pp;
plane.takeOff();//1:M
plane.fly();
plane.landing();
```

Here polymorphism is not happening well

Polymorphism has to reduce the code size.

Eg: polymorphic code

```
class Plane
{
    public void takeOff()
    {
        System.out.println("Plane is taking off");
    }
    public void fly()
    {
        System.out.println("Plane is flying");
    }
    public void landing()
    {
        System.out.println("Plane is landing");
    }

}

class CargoPlane extends Plane
{

    public void fly()
    {
        System.out.println("CargoPlane flies at lower height");
    }

}

class PassengerPlane extends Plane
{
    public void fly()
    {
        System.out.println("PassengerPlane flies at medium height");
    }

}

class FighterPlane extends Plane
{
    public void fly()
    {
        System.out.println("Fighter plane flies at high height");
    }
}
```

```
}

class Airport
{
    //any method name so give as permit
    public void permit(Plane plane)
    {
        plane.takeOff();
        plane.landing();
        plane.fly();
    }
}

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

        CargoPlane cp=new CargoPlane();

        PassengerPlane pp=new PassengerPlane();

        FighterPlane fp=new FighterPlane();

        Airport a=new Airport();
        a.permit(cp);
        a.permit(pp);
        a.permit(fp);

        //    Plane plane;
        //
        //    plane=cp;
        //    plane.takeOff(); //1:M
        //    plane.fly();
        //    plane.landing();
        //
        //    plane=pp;
        //
        //    plane.takeOff(); //1:M
        //    plane.fly();
        //    plane.landing();
    }
}
```

```
    }  
  
}
```

//In the above code size is reduced
//after jvm creates the obj it happens then only it is run time polymorphism and the above code
is also flexible

Loose coupling or upcasting is called run time polymorphism

```
class Parents  
{  
    public void cry()  
    {  
        System.out.println("Parents crying");  
    }  
}  
  
class Child11 extends Parents  
{  
    public void cry()  
    {  
        System.out.println("Child1 crying");  
    }  
    public void eat()  
    {  
        System.out.println("Child 1 eats less");  
    }  
}  
  
class Child12 extends Parents  
{  
    public void cry()  
    {  
        System.out.println("Child2 crying");  
    }  
    public void eat()
```

```

{
    System.out.println("Child 2 eats more");
}
}

public class LaunchParent2 {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Parents p=new Child11();
        p.cry();
        //p.eat(); // directly using parent type ref you cannot call
        child class specialized methods
        ((Child11) p).eat();//downcasting

        Parents p2=new Child12();
        p2.cry();
        ((Child12) p2).eat();

    }
}

```

Using parent type ref we can call only inherited and overridden methods

But we can't call the specialized methods.

By downcast parent type ref can access the specialized methods.

Abstraction

Abstraction in java is achieved using abstract keyword(0% 10% 50% 90% 100%) and also with interface(100%)

//definition //implementation //body all are same

1.abstract keyword is used for methods

In java we can have methods without implementation but those methods has to be declared as abstract.

Abstract methods are such methods body and implementations are not there.

=hiding actual implementation and only showing features.

2.

In a class if one method is also abstract then the class must be declared as abstract.

3.if one method is abstract then class is abstract

In a abstract class there can be

Abstract methods

And

Concrete methods

//abstract keyword can't be applied to variables

Which ever class extends the abstract class

Then the extended class has to give implementation for all the abstract methods or declare class as abstract

//we can create reference of abstract class

1.We can't create obj of abstract class.

Because there is not implementations for the methods.

But ref is possible to achieve the polymorphism

2.abstract class can have all methods as abstract

3.abstract class can have both abstract and concrete method

4.abstract class can have all methods as concrete. {they just allowed developers}

5.The sub class/child class if extending abstract class then either have to implement abstract method or declare the class as abstract.

6.constructor can't be made abstract and it won't participate in inheritance.(bcz constructor has super() method in first line)

Abstract class is incomplete class

Obj can't be created.

Can we make abstract class as final?

NO , it is illegal final and abstract is illegal combination.

Can we have constructor in abstract class ?

YES,

Concept of polymorphism and abstraction are dependent on inheritance.

MINI project

Req: to calculate the area of rec,circle and square ,
(pictorial representation in UML diagrams)

Rec	circle	square
Length : float		
Breadth : float		
input() : void		
compute() : void		
disp() : void		

With and without oops we can write the above project code

Eg: Write for oops

Inheritance,polymorphism,abstraction,encapsulation.

```
import java.util.Scanner;
//below code is abstraction
abstract class Shapes
{
    float area;
    abstract public void input();
    abstract public void compute();
    public void disp()
    {
        System.out.println("The area is "+ area);
    }
}

//extends is inheritance
class Rectangle extends Shapes
{
    float length;
    float breadth;

    public void input()
    {
        Scanner scan=new Scanner(System.in);
        System.out.println("please enter the length of rectangle");
        length=scan.nextFloat();
        System.out.println("please enter the breadth of rectangle");
        breadth=scan.nextFloat();
    }
    public void compute()
    {
        area=length*breadth;
    }
}
```

```
    }

}

//extends is inheritance
class Square extends Shapes
{
    float length;

    public void input()
    {
        Scanner scan=new Scanner(System.in);
        System.out.println("please enter the length of square");
        length=scan.nextFloat();
    }
    public void compute()
    {
        area=length*length;
    }
}

//extends is inheritance
class Circle extends Shapes
{
    float radius;
    final float pi=3.14f;
    public void input()
    {
        Scanner scan=new Scanner(System.in);
        System.out.println("please enter the radius of circle");
        radius=scan.nextFloat();
    }
    public void compute()
    {
        area=pi*radius*radius;
    }
}

//below code is polymorphism
class Geometry
```

```

{
    void permit(Shapes s)
    {
        s.input();
        s.compute();
        s.disp();
    }
}

public class LaunchProject {

    public static void main(String[] args) {

        Rectangle r=new Rectangle();
        Square s=new Square();
        Circle c=new Circle();

        Geometry g = new Geometry();
        g.permit(r);
        g.permit(s);
        g.permit(c);
    }
}

```

{ Class,methods ,variables,objects } without oops

But not recommended

```

import java.util.Scanner;

class Rectangle
{
    float area;
    float length;
    float breadth;

    void input()
    {
        Scanner scan = new Scanner(System.in);

```

```
System.out.println("Enter the length of rectangle : ");
length = scan.nextFloat();
System.out.println("Enter the breadth of rectangle : ");
breadth = scan.nextFloat();
}

void compute()
{
    area = length*breadth;
}

void disp()
{
    System.out.println("The area of the rectangle is "+area);
}

class Square
{
    float area;
    float length;

    void input()
    {
        Scanner scan = new Scanner(System.in);
        System.out.println("Enter the length of rectangle : ");
        length = scan.nextFloat();
    }

    void compute()
    {
        area = length*length;
    }

    void disp()
    {
        System.out.println("The area of the square is "+area);
    }
}
```

```
class Circle
{
    float area;
    float radius;

    void input()
    {
        Scanner scan = new Scanner(System.in);
        System.out.println("Enter the radius of circle : ");
        radius = scan.nextFloat();
    }

    void compute()
    {
        area = 3.14f*radius*radius;
    }

    void disp()
    {
        System.out.println("The area of circle is "+area);
    }
}

class withoutOOPs
{
    public static void main(String arg[])
    {
        Rectangle r = new Rectangle();
        r.input();
        r.compute();
        r.disp();

        Square s = new Square();
        s.input();
        s.compute();
        s.disp();

        Circle c = new Circle();
        c.input();
        c.compute();
        c.disp();
    }
}
```

```
}
```

19)CodeSnippet

(2:47:00 –

38)16-11-22 dbt

39)16-11-22

(HAS-Relationship and Dependency Injection)

This topic is extensively used in frameworks, especially for spring.

Relationship between two classes

1. Has - A relationship
2. Is - A relationship(using extends keyword)

```
class Car
{
    }           IS-A //does this make sense
class Engine extends Car{
```

Address - student , Account - employee

Car has an engine makes sense

After coding car and engine class , later they are binded.

Dependency Injection

```
Class Engine{//Dependant Object
```

```
}
```

```
Class Car{//Target Object
```

```
    //HAS-A relationship
```

```
    //Engine engine = new Engine();  
    Engine engine;//instance variable
```

```
}
```

=the process of injecting Dependent Object into Target Object is technically known as Dependency Injection

W.k.t Engine engine is the instance variable then it can be initialized in two ways that are through constructor and setter

Dependency Injection

The process of injecting dependant object into target object is called as "Dependancy injection".

We can achieve dependancy injection in 2 ways

- a. Constructor dependancy injection
- b. Setter dependancy injection

a. Constructor dependancy Injection

Injecting dependant object into target object through a constructor is called as "Constructor Dependancy Injection".

b. Setter dependancy Injection

Injecting dependant object into target object through a setter is called as "Setter Dependancy Injection".

eg#1

```
class Engine { //Dependant Object
```

```
}
```

```
class Car{//Target Object
```

```
    //HAS-A relationship
```

```
    Engine engine ; //instance variable
```

```
}
```

eg#2

```
class Address{//Dependant Object
}
class Student {//Target Object
//HAS-A relationship
Address address; //instance variable
}
eg#3
class Account{//Dependant Object
}
class Employee {//Target Object
//HAS-A relationship
Account account;//instance variable
}
```

Relationships in JAVA

As part of Java application development, we have to use entities(class) as per the application requirements.

In Java application development, if we want to provide optimizations over memory utilization, code Reusability, Execution Time, Sharability then we have to define relationships between entities.

There are three types of relationships between entities.

1. Has-A relationship (extensively used in projects)
2. IS-A relationship (achieved using extends keyword)
3. USE-A relationship (not popular)

Q) What is the difference between HAS-A Relationship and IS-A relationship?

Ans:

Has-A relationship will define associations between entities(class) in Java applications, here associations between entities will improve communication between entities and data navigation between entities.

IS-A Relationship is able to define inheritance between entity classes, it will improve "Code Reusability" in java applications.

Associations in JAVA

=====

There are four types of associations between entities

1. One-To-One Association (1:1)
2. One-To-Many Association (1:M)
3. Many-To-One Association (M:1)
4. Many-To-Many Association (M:M)

To achieve associations between entities, we have to declare either a single reference or array of reference variables of an entity class in another entity class.

```
class Address{  
    ---  
}  
class Account{  
    ---  
}  
class Employee{  
    ---  
    Address[] addr; // It will establish One-To-Many Association (1:M)  
    Account account; // One-To-One Association(1:1)  
}
```

Association = relating two classes, in such a way that communication becomes simple

Entities means classes

Code Reusability

```
class Object{  
    public boolean equals(Object obj){  
        ---  
    }  
    public String toString(){  
        ---  
    }  
}  
class Student{  
}
```

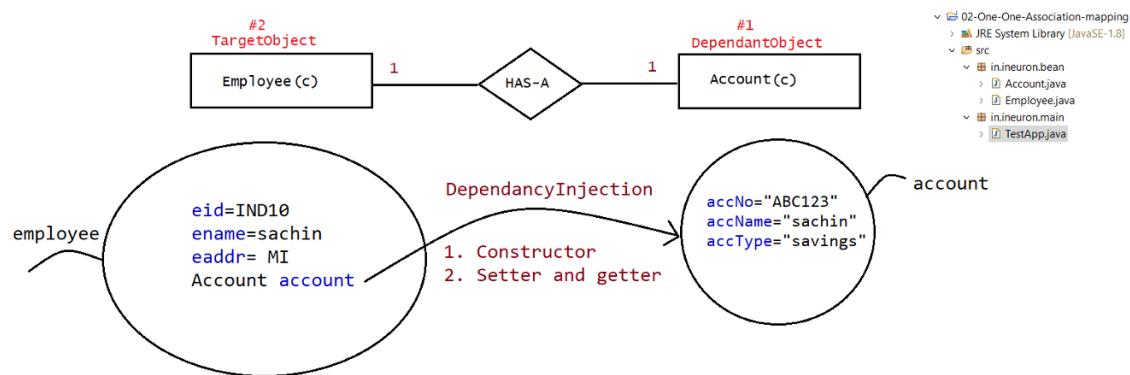
```
System.out.println(new Student());
```

1. One-To-One Association:

It is a relation between entities, where one instance of an entity should be mapped with exactly one instance of another entity.

eg: Every employee should have exactly one Account.

eg: refer:: 02-One-One-Association-mapping



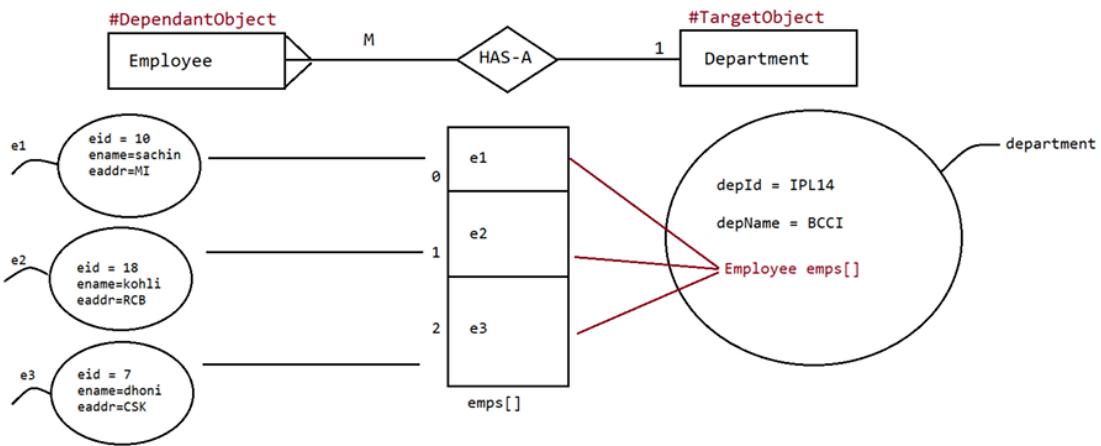
Data navigation is the employee obj can get the data of account.

2. One-To-Many Association:

It is a relationship between entity classes, where one instance of an entity should be mapped with multiple instances of another entity.

Example: Single department has multiple employees.

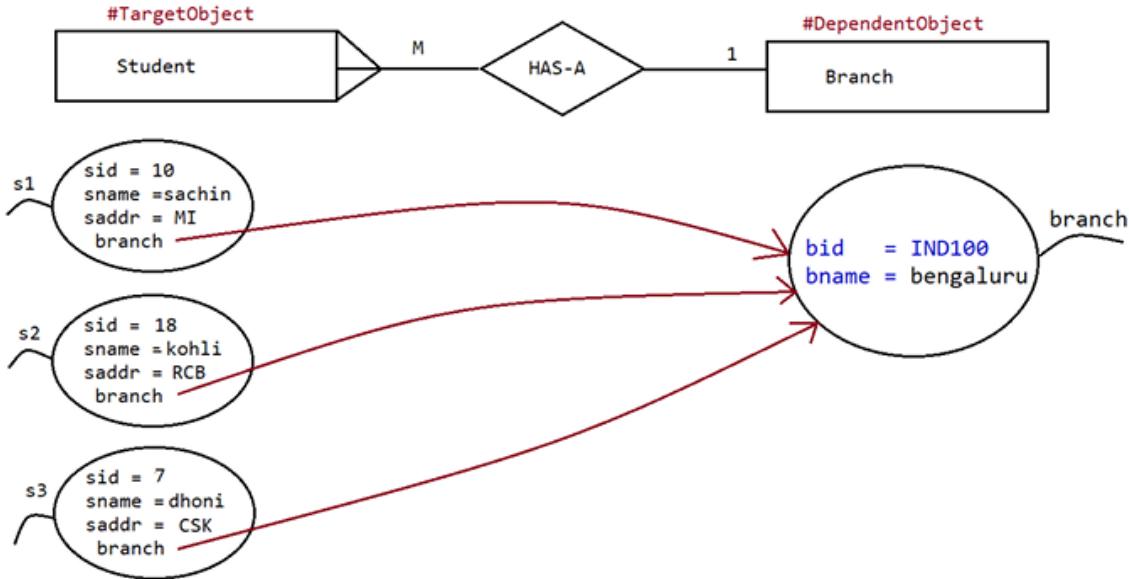
eg: refer:: 03-One-MANY-Association-mapping



3. Many-To-One Association:

It is a relationship between entities, where multiple instances of an entity should be mapped with exactly one instance of another entity.

Example: Multiple Students have joined with a single branch.
 eg: refer:: 04-MANY-ONE-Association-mapping



Completed in setters and getters style you do in constructor injection style

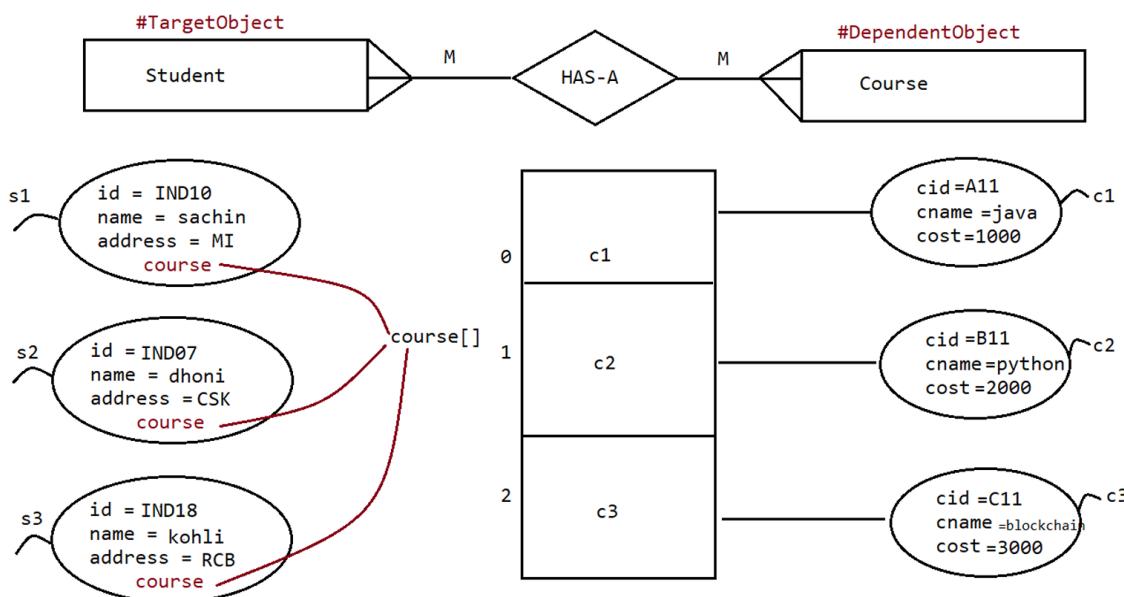
4. Many-To-Many Associations

(2:16:00–

It is a relationship between entities, Where multiple instances of an entity should be mapped with multiple instances of another entity.

EX: Multiple Students Have Joined with Multiple Courses

eg: refer:: 05-MANY-MANY-Association-mapping



21) Code Snippet

40) 17-11-22

(inner class and Introduction to Interface)

⇒ Java is for B2B

⇒ JavaScript and other are for B2C

Inner Class

Class human

```
{  
    Static String planet = "earth";  
    String name;  
    int age;  
}
```

Creating class inside a class

⇒ Design on paper how many class and methods

class A

```
{  
    Var;  
    Methods;  
    Class;  
}
```

Println belongs to which class PrintStream.class

Ctrl + hover and click

```
System.out.println("hi");  
PrintStream out = System.out;  
out.println("hello");
```

System is class

In and out are the objects

```
public class FirstCode
```

```
{
```

```
    private int num;
```

```
A obj = new A(); //class FirstCode knows about the class A  
//calling a function is a method so it can't be called here directly  
//a method can be called inside a method
```

```
public void show()  
{  
    System.out.println("in show"+num); //private var's can be  
    //accessible inside inner class
```

```
    obj.config();
```

```
}
```

```
class A      //inner class  
{  
    public void config()  
    {
```

```

        System.out.println("in config");
    }
}//how do you create the obj of A and call it

Public static void main(String[] arg){
//you can't call non static method inside static method
//with obj creation you can call it.
FirstCode obj = new FirstCode();
obj.show();

}

```

Class file is created for every class
 Inner class is like fileName\$innerClassName.class

```

class A
{
    public void show()
    {
        sop("hello, I am show method");
    }

    class B//you can make only inner class static,not outer classes
    {
        public void config()
        {
            sop(" I am config method");
        }
    }
}

public class SecondCode
{
    public static void main(String arg[])
    {
        A obj = new A();
        obj.show();

        // A.B obj1;
        // obj1 = new A.B();

        A.B obj1;
    }
}
```

```
    obj1 = obj.new B();  
  
    obj1.config();  
  
}  
  
//why do we don't make outer class static ?  
//why inner class exists?
```

Creating inner class is for designing
Create a different classes
break down into different class and modules

There are lot of design flow diagrams.
How many class we need and interfaces.

Its free to makes class
Its about maintenance

It is rarely used.

You can create a class inside a method

Documentation

```
/*  
*Build by Sri  
*  
*  
*/
```

Cmd for documentation
>>javadoc FirstCode.java
Automatically documentation is generated.

SecondCode.java
class Computer{
 public void config()
 {
 sop("in computer config");
 }
}

```

class AdvComputer extends Computer
{
    public void config()
    {
        sop("in AdvComputer config");
    }
}//the main purpose of advComputer class is to override the config method
//in entire the project this class is used only once.
//instead of it you can use the inner class concept
//providing the design details at the time of manufacturing or creation
//that is only the inner class

public class SecondCode
{
    public static void main(String arg[]){
        Computer obj = new AdvComputer();
        obj.config();
    }
}

```

Anonymous Inner Class

=it has to be b/w the semicolon and obj (constructor)

SecondCode.java

```

class Computer{
    public void config()
    {
        sop("in computer config");
    }
}

public class SecondCode
{
    public static void main(String arg[]){
        Computer obj = new AdvComputer()
        {//anonymous inner class
            public void config()
            {
                System.out.println("something new");
            }
        }
    }
}

```

```
        }
    };
    obj.config();
}
-----  
1.  
abstract class Computer{
    public abstract void config();

}

class Laptop extends Computer
{
    public void config()
    {
        System.out.println("its working");
    }

}
//intension : is just to implement the class
```

```
public class SecondCode
{
    public static void main(String arg[]){
        Computer obj = new Laptop();
        obj.config();
    }
}
```

```
2.  
abstract class Computer{
    public abstract void config();

}
public class SecondCode
{
    public static void main(String arg[]){
        Computer obj = new Laptop()
        {
            public void config()
            {
                System.out.println("its working very fine");
            }
        };
    }
}
```

```
    obj.config();  
}  
-----
```

Normal inner class is less used.
But anonymous inner class is widely used.

Interface

(1:37:00)

Way of communication, medium, way of representation.

From the perspective of java

IT IS SRS(service requirement specification)

Any srs is called as interface

Any db vendors

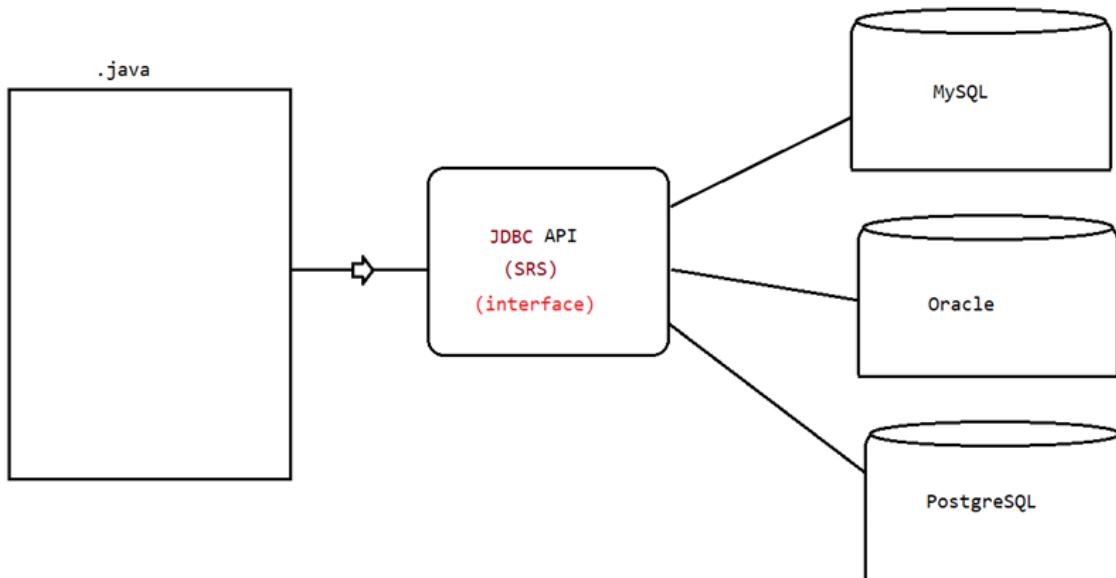
Oracle

MySQL

Mongo DB

PostgreSQL

Etc..



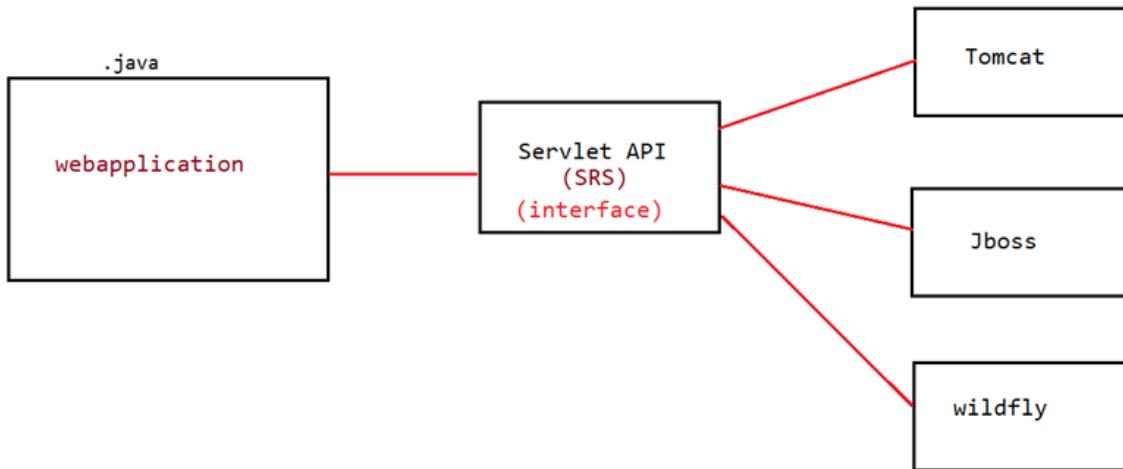
For three db write three code to connect.

But java writes only one code.

API

Sun Microsystem gave JDBC API as an interface

Java IS WORA so , only one code to connect to all db's



Web application are not run by JVM , that are run on servers

Server softwares :- Tomcat , Jboss , wildfly ,etc..

Multiple platforms are the servers.

THEY GAVE THE SERVLET API

That code can be run on tomcat , Jboss and wildfly.

=====

Def1::Any service requirement specification(SRS) is called interface.

eg:: SUN people responsible to define JDBC API and database vendor will provide implementation for that

SUN

|

JDBC API

|

Oracle MySQL PostgreSQL

=====

SUN

|

Servlet API

|

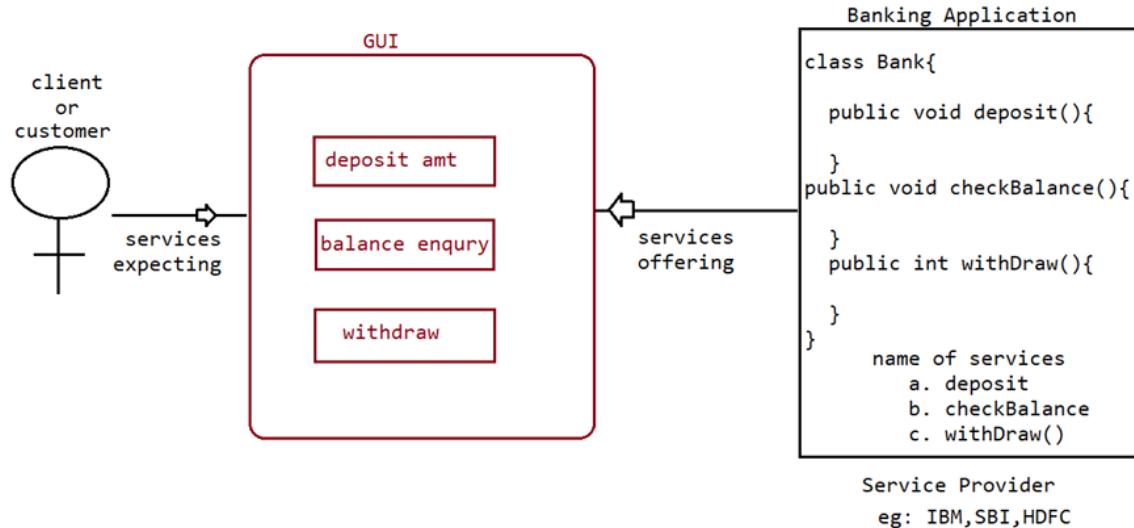
tomcat weblogic jboss

=====

.class ⇒ JVM (PART OF JDK SOFTWARE)

.class ⇒ JVM (part of special software called servers like tomcat , glassfish,jboss).

eg: Banking application through the JAVA



Services are deposit , checkbalance , withdraw

GUI is an interface between client and bank application.

Def2:: From client point of view an interface define the set of services what is expecting.

From service provider point of view an interface define the set of services what is "offering".

So interface acts a contract b/w client and serviceprovider.

eg:: GUI screen of ATM defines the set of services what the customer is expecting,

Bank people offered the same set of services what the customer is expecting.

Cusomter => GUI => Bank

Service provider

Eg : IBM , SBI ,HDFC

```

/*
class Account
{
    //abstract method = (it is not 100% abstract).
    public abstract void withdraw();
    public abstract void deposit();
    public abstract void checkBalance();
    public abstract void printPassbook();
    {
        //logic
  
```

```

    }
}

interface Account (100% abstraction to promote in java)
//by default all inside methods are public and abstract.
    void withdraw();
    void deposit();
    void checkBalance();

}

class Savings extends Account
{
}

class Current extends Account
{
}

class Salary extends Account
{
}

public class TestApp
{
    public static void main(String arg[])
    {

    }
}

```

Def3:: Inside interface every method is always abstract whether we are declaring or not hence interface is considered as 100% pure abstract class.

eg:

```

interface Account
{
//It is 100% abstract class
//The methods are by default "abstract and public"
void withdraw();
void deposit();
void checkBalance();
}

```

Summary::

Interface corresponds to Service Requirement Specification(SRS) or contract b/w client and service provider or **100% pure abstract class**

Conventions for interface

Class name or interface name to identify ?

1.

```
interface ISample
{
    void m1();
    void m2();
}
abstract class SampleImpl implements ISample
{
    void m1(){
        System.out.println("hey implementation given");
    }
    public abstract void m2();
}
public class TestApp
{
    public static void main(String arg[])
    {
        System.out.println("Implementation given");
    }
}
```

2.

```
interface ISample
{
    void m1();
    void m2();
}
class SampleImpl implements ISample
{
    @Override //indication to compiler that these methods are overridden methods
    public void m1(){
        System.out.println("hey implementation given for m1");
    }
    @Override
    public void m2(){
        System.out.println("hey implementation given for m2");
    }
}
public class TestApp
{
    public static void main(String arg[])
    {
        System.out.println("Implementation given");
    }
}
```

```

{
    ISample sample = new SampleImpl();//loose coupling
    sample.m1();
    sample.m2();
}

```

//accessibility can never be reduced, so in class methods are declared as public

Declaration and implementation of Interface

a. Whenever we are implementing an interface compulsorily for every method of that interface

we should provide implementation otherwise we have to declare class as abstract class in that case child class is responsible to provide implementation for remaining methods.

b. Whenever we are implementing an interface method compulsorily it should be declared as public otherwise it would result in

CompileTime Error.

```

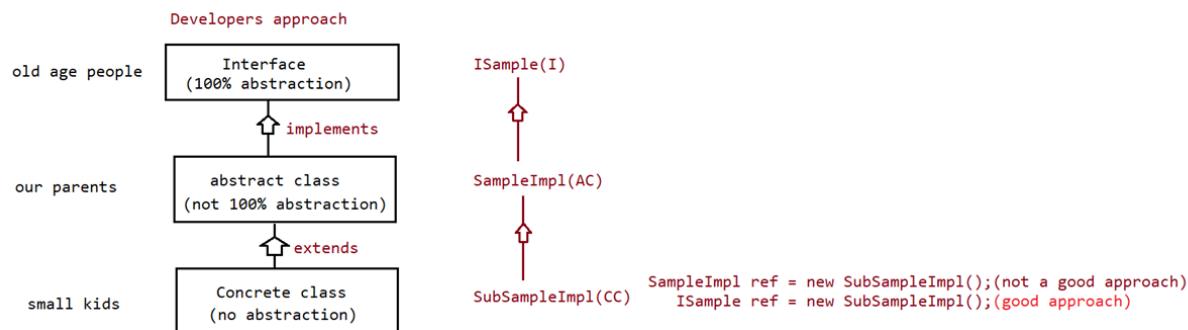
eg:: interface ISample{
void methodOne();
void methodTwo();
}

abstract class Sample implements ISample{
public void methodOne(){...}

}

class SubServiceProvider extends Sample{
    @Override
public void methodTwo(){...}
}

```



22)codeSnippet

41)18-11-22

(Interface continuation)

Difference b/w extends vs implements

case1:

extends :: One class can extends only class at a time

eg:: class One{

 class Two extends One{}

implements:: One class can implement any no of interface at a time.

```
eg:: interface One{
    void methodOne();
}
interface Two{
    void methodTwo();
}
class Demo implements One,Two{
    public void methodOne(){}
    public void methodTwo(){}
}
```

case2:

A class can extend a class and can implements any no of interfaces simultaneously.

```
eg: interface One{
    public void methodOne();
}
class Two{
    public void methodTwo(){}
}
class Three extends Two implements One{
    @Override
    public void methodOne(){
    ...
    }
    @Override
    public void methodTwo(){}
```

```
    }  
}
```

case3:

An interface can extend any no; of interfaces at a time.

```
eg:: interface IOne{  
    public void methodOne();  
}  
interface ITwo{  
    public void methodTwo();  
}  
interface IThree extends IOne,ITwo{  
    public void methodOne();  
    public void methodTwo();  
    public void methodThree();  
}
```

Can a class can simultaneously extend and implements

- 1.reusability \Rightarrow extends
- 2.implementation \Rightarrow implements
(interface)

Class & class extends

Class & interface implements

Class extends class and implements interface

Interface and interface extends

Interface IDemo3 extends IDemo1, IDemo2

Answer the following

=====

Which of the following is true?

- a. A class can extend any no of class at a time.
- b. An interface can extend only one interface at at time.
- c. A class can implement only one interface at at a time.
- d. A class can extend a class and can implement an interface but not both simultaneously.
- e. An interface can implements any no of Interfaces at a time.
- f. None of the above

answer: f

Consider the expression X extends Y which of the possibility of X and Y expression is true?

1. Both x and y should be classes.
2. Both x and y should be interfaces.
3. Both x and y can be classes or can be interfaces.
4. No restriction.

Equation : X extends Y == > true

class extends class => true

interface extends interface => true

Answer: 3

Q>

Predict X,Y,Z

- a. X extends Y,Z?
- b. X extends Y implements Z?
- c. X implements Y,Z?
- d. X implements Y extends Z?

a. X extends Y,Z?

X => interface

Y => interface

Z => interface

b. X extends Y implements Z?

X => class

Y => class

Z => interface

c. X implements Y,Z?

X => class

Y => interface

Z => interface

d. X implements Y extends Z

combination is illegal

Interface Methods

Every method present inside the interface is public.

Every method present inside the interface is abstract.

How many declaration are valid?

- a. void methodOne();
- b. public void methodOne();
- c. abstract void methodOne();
- d. public abstract void methodOne();

answer: All are valid

public => To make the method available for every implementation class.
abstract => Implementation class is responsible for providing the implementation.

java.sql.*

- 1.Statement
- 2.PreparedStatement
- 3.CallableStatement
- 4.Connection
- 5.ResultSet

(respective db vendors has to give implementation for all the above interfaces)

eg: jdbc api (java.sql.*)

|

|

implementation should be given by

- a. mysql
- b. oracle
- c. postgresql

How many access modifiers are there in java?

Generally 11

That are

- a.public
- b.protected
- c.private
- d.static
- e.synchronized
- f.strictfp
- G.final
- h.abstract
- i.native
- j.transient
- k.volatle

Since the methods present inside the interface is

=> public,abstract methods illegal combination of modifiers are

Static,private,protected,strictfp,synchronized,native,final.

Method in interface

- a.public and abstract

(we can't mark the above as private,protected,static,synchronized,strictfp,final,native,transient
are illegal combination for methods in interface.)

/* Variables in interface
transient and volatile are applicable */ doubt

Static => associated with implementation

Abstract → can't have implementation

Interface variables

=====

- => Inside the interface we can define variables.
- => Inside the interface variables define requirement level constants.
- => Every variable present inside the interface is by default public static final.

eg:: interface ISample{
int x=10;
}
public :: To make it available for implementation class Object.
static :: To access it without using implementation class Name.
final :: Implementation class can access the value without any
Modification.

```
interface IRemote
{
    //SRS
    public static final int MIN_VOLUME = 0;
    public static final int MAX_VOLUME = 100;
}
class LGImpl implements IRemote
{
    System.out.println(IRemote.MIN_VOLUME);
    System.out.println(IRemote.MAX_VOLUME);
}
public class TestApp{
    public static void main(String arg[]){
        }
}
```

//if it is final then value has to be initialized.

Static
a.complete information should be provided

variable declaration inside interface

- a. int x=10;
- b. public int x=10;
- c. static int x=10;
- d. final int x=10;
- e. public static int x=10;
- f. public final int x=10;
- g. static final int x=10;
- h. public static final int x=10;

Answer: All are legal

since the variable defined in interface is public static final,we cannot use modifiers like private,protected,transient,volatile.

since the variable is static and final, compulsorily it should be initialized at the time of declaration otherwise it would result in compile time error.

eg:: interface IRemote{ int x;}// compile time error.

=> interface variables can be accessed from implementation class, but cannot modify if we try to modify

it would result in compile time error.

```
eg:: interface Remote{
    int VOLUME = 100;
}
class Lg implements Remote{
public static void main(String... args){
    VOLUME=0;//CE:: cannot assign a value to final variable VOLUME
    System.out.println("value of volume is ::"+VOLUME);
}
}

eg:: interface Remote{
    int VOLUME = 100;
}
class Lg implements Remote{
public static void main(String... args){
    int VOLUME=0;//local variable
    System.out.println("value of volume is ::"+VOLUME);//0
}
}
```

Local variable and interface variable

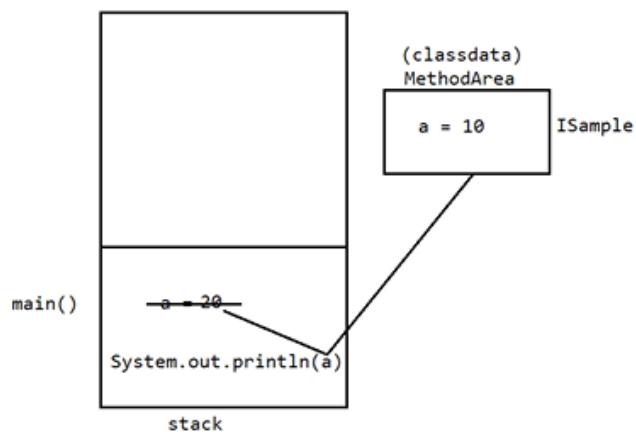
Local variable win's.

```
interface ISample
{
    int a=10;//public static final
}

public class TestApp implements ISample{

    public static void main(String[] args){

        int a = 20;//local variable
        System.out.println(a);
    }
}
```



System.out.println(ISample.a);

```

interface ISample
{
    void m1();
}
interface IDemo
{
    void m1();
}
class CommonImpl implements ISample, IDemo
{
    public void m1() //valid
    {
        sop("m1");
    }
}

```

Super method comes when there is relationship between the class and class

Interface Naming Conflicts

Case 1::

If 2 interfaces contain a method with same signature and same return type in the implementation class only
one method implementation is enough.

eg::

```

interface Left{ public void methodOne();}
interface Right{public void methodOne();}
class Test implements Left,Right{
@Override
    public void methodOne(){
...
    }
}

```

Case2:

If 2 interfaces contain a method with same name but different arguments in the implementation class we have to provide implementation for both methods and these methods acts as Overload methods.
eg::

```

interface Left{ public void methodOne();}
interface Right{public void methodOne(int i);}
class Test implements Left,Right{
@Override

```

```
public void methodOne(){
    ...
}
@Override
public void methodOne(int i){
...
}
}
```

case3::

If two interfaces contain a method with the same signature but different return types then it is not possible to implement both interfaces simultaneously.

```
eg:: interface Left { public void methodOne(); }
        interface Right{ public int methodOne(); }
        class Test implements Left,Right{
            @Override
            public void methodOne(){
                ...
            }
        @Override
            public int methodOne(){
...
            }
        }
//invalid
```

Note:

Q> Can a java class implements 2 interfaces simultaneously?

yes possible, except if two interfaces contains a method with same signature but different return types.

Variable naming conflicts::

Two variables can contain a variable with same name and there may be a chance variable naming

conflicts but we can resolve variable naming conflicts by using interface names.

example1:

```
interface Left{ int x=888;}
        interface Right{ int x=999;}
        public class Test implements Left,Right{
            public static void main(String... args){
                System.out.println(Left.x);
                System.out.println(Right.x);
            }
        }
```

```
>>javap java.lang.Runnable
```

```
>>javap java.io.Serializable
```

```
>>javap java.lang.Cloneable
```

Note:

inside the interface the methods are by default "public and abstract".

inside the interface the variables are by default " public static and final".

We can also write an interface without any variable or abstract methods.

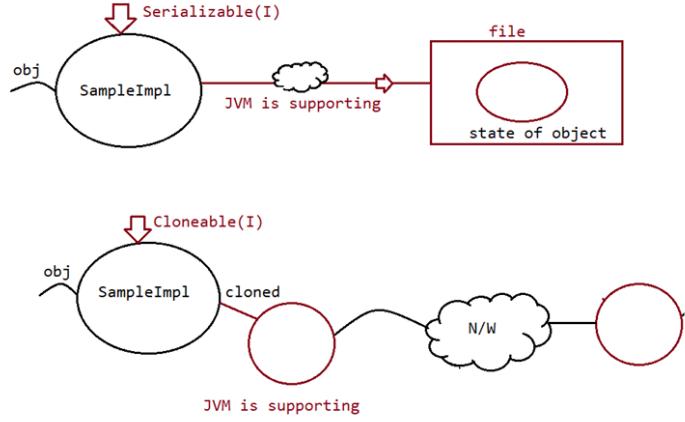
```
interface Serializable{  
}  
class SampleImpl implements Serializable{  
}
```

```
interface Cloneable{  
}  
class SampleImpl implements Cloneable{  
}
```

"Marker Interfaces"

```
interface Serializable{  
}  
class SampleImpl implements Serializable{  
}
```

```
interface Cloneable{  
}  
class SampleImpl implements Cloneable{  
}
```



What is the benefit if obj implements the serializable
Storing the data onto a file

Jvm will support functionality for some interfaces

`Cloneable(I) ⇒ obj` can be duplicated and that can be send it over network.

We won't write the code(for `cloneable(I)`), it is supported by jvm.

MarkerInterface

=====

=> If an interface does not contain any methods and by implementing that interface if our Object will get some ability such type of interface are called "Marker Interface"/"Tag Interface"/"Ability Interface".

=> example

 Serializable,Cloneable,SingleThreadModel.

example1

 By implementing a Serializable interface we can send that object across the network and we can save the state of an object into the file.

example2

 By implementing a SingleThreadModel interface servlet can process only one client request at a time so that we can get "Thread Safety".

example3

 By implementing Cloneable Interface our object is in a position to provide exactly duplicate cloned objects.

Without having any methods in the marker interface how objects will get ability?

Ans.JVM is responsible to provide required ability.

Why is JVM providing the required ability to Marker Interfaces?

Ans. To reduce the complexity of the programming.

Can we create our own marker interface?

Yes, it is possible but we need to customize JVM.(hard for beginner)

```
interface IDemo
```

```
{
```

```
}
```

```
class TestApp implements IDemp
```

```
{
```

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

```
{
```

```
}
```

```
}
```

//JVM → customize the jvm by writing lines of code for ur marker interface.

//customization is at architect level

//you can build your own api's

=====

=====

=====

Adapter class(It is a design pattern allowed to solve the problem of direct implementation of interface methods)

```
=====
=====
=====
```

It is a simple java class that implements an interface only with empty implememtation for every method.

If we implement an interface compulsorily we should give the body for all the methods whether it

is required or not. This approach increases the length of the code and reduces Readability.

```
eg:: interface X{
    void m1();
    void m2();
    void m3();
    void m4();
    void m5();
}
class Test implements X{
    public void m3(){
        System.out.println("I am from m3()");
    }
    public void m2(){}
    public void m3(){}
    public void m4(){}
    public void m5(){}
}
```

In the above approach, even though we want only m3(), still we need to give body for all the abstract methods, which increase the length of the code, to reduce this we need to use "Adapater class".

Instead of implementing the interface directly we opt for "Adapter class".

Adapter class are such classes which implements the interface and gives dummy implementation for all the abstract methods of interface.

So if we extends Adapter classes then we can easily give body only for those methods which are interested in giving the body.

```
eg::
interface X{
```

```

        void m1();
        void m2();
        void m3();
        void m4();
        void m5();
    }
abstract class AdapterX implements X{
    public void m1(){}
    public void m2(){}
    public void m3(){}
    public void m4(){}
    public void m5(){}
}
class TestApp extends AdapterX{
    public void m3(){
        System.out.println("I am from m3()");
    }
}

```

eg:

```

Servlet(I)
| implements
GenericServlet(Abstract class)
| extends
HttpServlet(Abstract class)
| extends
MyServlet(class)

```

//more number of abstract classes then less code at the bottom

23)CodeSnippets

(2:47:00 ----

42)19-11-22 dbt

43)21-11-22

(interface conclusion and wrapper classes)

3 def for interface

Srs

Contract b/w client and service provider

100% Pure abstraction

Methods inside the interface are by default public and abstract.

Variables inside the interface are by default public static and final.

Using interfaceName we can resolve the naming conflict of variables in interfaces.

Difference between concrete, abstract classes and interface and when to use them ?

Nothing i know about implementation => interface

Partial i know about the implementation => abstract class

Complete implementation and ready to provide service => concrete class

When to go interface, abstract class and concrete class?

interface:: It is prefered when we speak only about specification(no implementation).

abstract class:: It is prefered when we speak about partial implementation.

concrete class:: It is prefered when we speak about complete implementation and ready to provide service then we go for concrete class.

Difference b/w interface and abstract class?

Interface:: If we dont know anything about implementation just we have requirement specification then we should go for an interface.

Abstract class: If we are talking about implementation but not completely then we should go for an abstract class.

Interface:: Every method present inside the interface is always public and abstract whether we are declaring or not.

Abstract :: Every method present inside abstract class need not be public and abstract.

Interface:: We can't declare interface methods with the modifiers like private,protected,final,static,synchronized,native,strictfp.

Abstract :: There are not restrictions on abstract class method modifiers.

Interface:: Every interface variable is always public static final whether we are declaring or not.

Abstract:: Every abstract class variable need not be public static final.

Interface:: Every interface variable is always public static final we can't declare with the following modifiers like private,protected,transient,volatile.

Abstract:: No restriction on access modifiers

Interface:: For every interface variable compulsorily we should perform

initialisation at the time of declaration, otherwise we get a compile time error.

Abstract:: Not required to perform initialisation for abstract class variables at the time of declaration.

Interface:: Inside interface we can't write static and instance block.

Abstract :: Inside abstract class we can write static and instance block.

Interface:: Inside interface we can't write constructor.

Abstract :: Inside abstract class we can write constructor.

Note:

static block => .class file loading happens and used to initialize static variables.

instance block => during the creation of an object, just before the constructor call used for initialization instance variable.

constructor => during the creation of an object, used for initialization instance variable.

What is the need of abstract class? can we create an object of abstract class,

Does it contains constructor?

=> abstract class object cannot be created becoz it is abstract.

=> But constructor is needed for constructor to initialize the object.

eg::

```
class Parent {  
    Parent() {  
        System.out.println(this.hashCode()); //this method comes from objClass  
    }  
}  
  
class Child extends Parent {  
    public Child() {  
        System.out.println(this.hashCode());  
    }  
}  
  
public class TestApp {  
    public static void main(String[] args) {  
        Child c = new Child();  
        System.out.println(c.hashCode());  
    }  
}
```

/*
during the child class obj creation, only child class obj will be created but no the parent class object (still constructor of parent is called to bring the properties of parent to child).

*/

Can abstract class be instantiated/object be created ? ans.No

Can abstract class contains constructor ? ans.Yes

Why abstract class can contain constructor whereas interface doesnot contain constructor?

abstract class =>constructor it is used to perform initialization of the object.

it is used to provide the value for the instance variable.

it is used to contain instance variable which are required for child

object to perform intialisation for those instance variables.

interface => every variable is always static,public and final their is no chance of existing instance variable inside the class.

so we should perform initialisation at the time of declaration.

so constructor is not required for interface.

eg#1.=>real time person won't exists but type exist of it like , student , teacher,....

```
abstract class Person{
```

```
    String name;
```

```
    int age;
```

```
    int height;
```

```
    int weight;
```

```
    Person(String name,int age,int height,int weight){
```

```
        super();
```

```
        this.name=name;
```

```
        this.age=age;
```

```
        this.height=height;
```

```
        this.weight=weight;
```

```
}
```

```
}
```

```
class Student extends Person{
```

```
    int rollno;
```

```
    int marks;
```

```
    Student(String name,int age,int height,int weight,int rollno,int marks){
```

```
        super(name,age,height,weight,rollno);
```

```
        this.rollno=rollno;
```

```
        this.marks=marks;
```

```
}
```

```
}
```

//setter and getters are not used, generally constructor is preferred.
//can interface obj be instantiated ? ans.no
//can interface contains constructor ? ans. No instance var's,so constructor not required

Question1:

Can reference be created for abstract class? yes
Person p =new Student("sachin",49,5.6f,71,10,100);

Can reference be created for interface?

```
ISample sample = null;
```

Note::Every method present inside the interface is abstract, but in abstract class also we take only abstract methods then

what is the need of interface concept?

```
abstract class Sample{  
    public abstract void m1();  
    public abstract void m2();  
}  
interface ISample{  
    void m1();  
    void m2();  
}
```

=> we can replace interface concepts with abstract classes, but it is not a good programming practice.

```
eg#1  
interface X{  
    ...  
    ...  
}  
class Test implements X{  
    ...  
    ...  
}
```

Test t=new Test();
i. performance is high.
ii.While implementing X we can extends
one more class,through which we can bring reusability.

```
eg#2.  
abstract X{
```

```
...
...
}
class Test extends X{
...
...
}
Test t=new Test();
i.performance is low.
ii.While extending X we can't extends
any other classes so reusability is not brought.
```

Note: If everything is abstract then it is recommended to go for an interface.

Wrapper class

(1:52:00–

Purpose

1. To wrap primitives into object form so that we can handle primitives also just like objects.
2. To define several utility functions which are required for the primitives.

Wrapper class are also helper classes
FROM JDK 1.5 wrapper classes introduced.

Constructors (level)

Almost all the Wrapper class have 2 constructors

- a. one taking a primitive type.
- b. one taking a String type.

eg: Integer i=new Integer(10);
Integer i=new Integer("10");
Double d=new Double(10.5);
Double d=new Double("10.5");

Note: If String argument is not properly defined then it would result in RunTimeException called "NumberFormatException".

```
eg:: Integer i=new Integer("ten");//RE:NumberFormatException
```

Wrapper class and its associated constructor

Byte => byte and String
Short => short and String
Integer => int and String
Long => long and String
**Float => float ,String and double
Double => double and String
**Character=> character
***Boolean => boolean and String

eg::

- 1) Float f=new Float (10.5f);
- 2) Float f=new Float ("10.5f");
- 3) Float f=new Float(10.5);
- 4) Float f=new Float ("10.5");

eg::

- 1) Character c=new Character('a');
- sop(c)
- 2) Character c=new Character("a"); //invalid
- sop(c)

eg::

Boolean b=new Boolean(true);
Boolean b=new Boolean(false);
Boolean b1=new Boolean(True);//C.E
Boolean b=new Boolean(False);//C.E
Boolean b=new Boolean(TRUE);//C.E

//String input = case is not imp , content is not imp

//String input = case insensitive of true is treated as true other cases treated as false.

eg::

Boolean b1=new Boolean("true");
Boolean b2=new Boolean("True");//false
Boolean b3=new Boolean("false");
Boolean b4=new Boolean("False");

Boolean b5=new Boolean("nitin");
Boolean b6=new Boolean("TRUE");
System.out.println(b1);//true
System.out.println(b2);//true

```
System.out.println(b3);//false  
System.out.println(b4);//false  
System.out.println(b5);//false  
System.out.println(b6);//true
```

eg::

```
class Test  
{  
public static void main(String[] args)  
{  
Boolean b1 =new Boolean("yes");//false  
Boolean b2 =new Boolean("no");//false  
System.out.println(b1);  
System.out.println(b2);  
System.out.println(b1.equals(b2));//false.equals(false)-> true  
System.out.println(b1 == b2);//false //pointing to diff obj  
Integer i1 = new Integer(10);  
Integer i2 = new Integer(10);  
System.out.println(i1);//10  
System.out.println(i2);//10  
System.out.println(i1.equals(i2));//true //compares the content  
}  
}
```

Note: In case of Boolean constructor, boolean value be treated as true w.r.t to case insensitive part of "true", for all others it would be treated as "false".

Note:

If we are passing String argument then case is not important and content is not important.

If the content is case insensitive String of true then it is treated as true in all other cases it is treated as false.

Note: In case of Wrapper class, `toString()` is overridden to print the data.

In the case of Wrapper class, `equals()` is overridden to check the content.

Just like String class, Wrapper classes are also treated as "Immutable Class".

(2:41:00–

Immutable class

=====

If we create an Object and if we try to make a change, with that change new object will be created and those changes will not be reflected in the old copy.

Can we make our user defined class as Immutable?

ans. yes possible as shown below

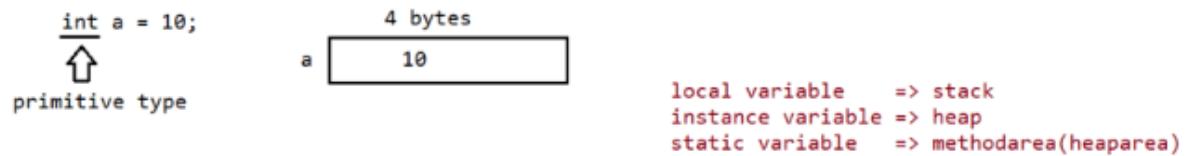
final class Test

```
{  
    int i;  
    Test(int i){  
        this.i = i;  
    }  
    public Test modify(int i){  
        if (this.i == i)  
            return this;  
        else  
            return new Test(i);  
  
    }  
    public static void main(String[] args)  
    {  
        Test t1 = new Test(10);  
        Test t2 = t1.modify(10);  
        Test t3 = t1.modify(100);  
        Test t4 = t3.modify(100);  
        System.out.println(t1==t2);//true  
        System.out.println(t1==t3);//false  
        System.out.println(t2==t3);//false  
        System.out.println(t3==t4);//true  
    }  
}
```

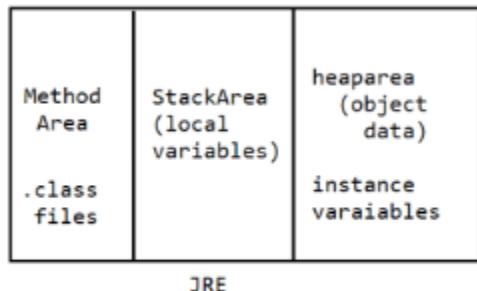
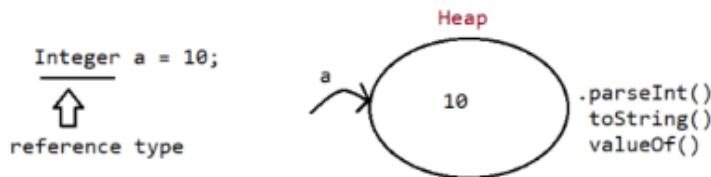
```
>>javap java.util.Collection  
Javap java.util.List  
Javap java.util.ArrayList  
Javap java.util.AbstractList
```

Primitive type 4 byte and 10 stored
int a = 10;

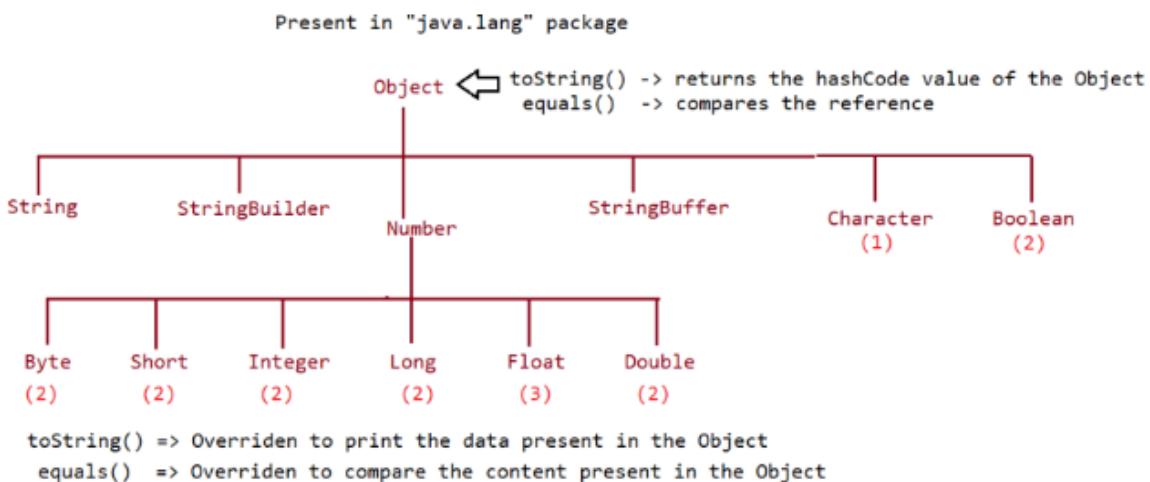
Reference type //object created and 10 stored , you can call Integer a = 10;
//methods of the obj



JDK1.5V Wrapper classes are introduced



Down are number of constructors

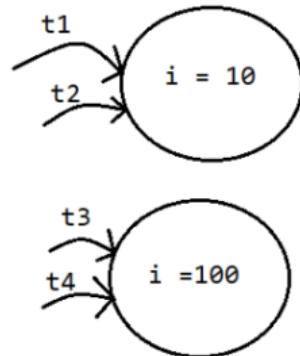


```

class Test
{
    int i;
    Test(int i){
        this.i = i;
    }
    public Test modify(int i){
        if (this.i == i)
            return this;
        else
            return new Test(i);
    }
    public static void main(String[] args)
    {
        Test t1 = new Test(10);
        Test t2 = t1.modify(10);
        Test t3 = t1.modify(100);
        Test t4 = t3.modify(100);

        System.out.println(t1==t2); //true
        System.out.println(t1==t3); //false
        System.out.println(t2==t3); //false
        System.out.println(t3==t4); //true
    }
}

```



`Integer i1 = new Integer(10);`

`sop(i1); //toString()`

`Integer i2 = new Integer("10");`

`sop(i2);`

`Integer i3 = new Integer("ten");`

`sop(i3); //RE`

`>>java.lang.Integer(java.lang.String) error is java.lang.NumberFormatException`

Implement of `toString` is to return hashcode

24)CodeSnippet

44)22-11-22

(wrapper class)(starts from 22 min)(no snippet)

What was the need of wrapper classes ?

To store primitive data in the form of obj so that we can make use of utility methods.

String ⇒ case insensitive of true is true remaining all false

Object

toString() => returns the hashcode of the obj

equals() => compares the reference

Wrapper classes

toString() ⇒ compares the content

Equals() ⇒ prints the string

Wrapper class utiltiy methods

=====

1. valueOf() method.

2. XXXValue() method.

3. parseXxx() method.

4. toString() method.

```
public static wrapper valueOf(String data, int radix) throws  
java.lang.NumberFormatException;  
public static wrapper valueOf(String data) throws  
java.lang.NumberFormatException;  
public static wrapper valueOf(int data);
```

valueOf() method

=====

To create a wrapper object from primitive type or String we use valueOf().

It is alternative to constructor of Wrapper class, not suggestable to use.

Every Wrapper class,except character class contain static valueOf() to create a Wrapper Object.

eg#1.

```
Integer i=Integer.valueOf("10");
```

```
Double d=Double.valueOf("10.5");
```

```

Boolean b=Boolean.valueOf("nitin");
System.out.println(i);
System.out.println(d);
System.out.println(b);

eg#2.
public static valueOf(String s,int radix)
|=> binary : 2(0,1)
|=> octal : 8(0-7)
|=> decimal : 10(0-9)
|=> hexadecimal : 16(0-9,a,b,c,d,e,f)
|=> base : 36(0-9,a-z)

Integer i1=Integer.valueOf("1111");
System.out.println(i1);//1111
Integer i2=Integer.valueOf("1111",2);
System.out.println(i2);//15
Integer i3=Integer.valueOf("ten");
System.out.println(i3);//RE:NumberFormatException
Integer i4=Integer.valueOf("1111",37);
System.out.println(i4);//RE:NumberFormatException

eg#3.
public static valueOf(primitivetype x)
Integer i1=Integer.valueOf(10);
Double d1=Double.valueOf(10.5);
Character c=Character.valueOf('a');
Boolean b=Boolean.valueOf(true);
Primitive/String =>valueOf() => WrapperObject

```

xxxValue()

We can use xxxValue() to get primitive type for the given Wrapper Object.
These methods are a part of every Number type Object.
(Byte,Short,Integer,Long,Float,Double) all these classes have these 6 methods which is

Written as shown below.

Methods

=====

```

public byte byteValue();
public short shortValue();
public int intValue();
public long longValue();
public float floatValue();
public double doubleValue();

```

eg#1.

```

Integer i=new Integer(130);

```

```
// result = minrange +(total -maxrange -1)
System.out.println(i.byteValue());//-126
System.out.println(i.shortValue());//130
System.out.println(i.intValue());//130
System.out.println(i.longValue());//130
System.out.println(i.floatValue());//130.0
System.out.println(i.doubleValue());//130.0
```

charValue()

Character class contains charValue() to get Char primitive for the given Character Object.

```
public char charValue()
eg#1.
Character c=new Character('c');
char ch= c.charValue();
System.out.println(ch);
```

booleanValue()

Boolean class contains booleanValue() to get boolean primitive for the given boolean

Object.
public boolean booleanValue()
eg#1.
Boolean b=new Boolean("nitin");
boolean b1=b.booleanValue();
System.out.println(b1);//false
In total xxxValue() are 36 in number.
=> xxxValue() => convert the Wrapper Object => primitive.

parseXXXX()

=====

We use parseXXXX() to convert String object into primitive type.
form-1

=====

```
public static primitive parseXXX(String s)
```

Every wrapper class,except Character class has parseXXX() to convert String into primitive type.

```
eg: int i=Integer.parseInt("10");
    double d =Double.parseDouble("10.5");
```

```

boolean b=Boolean.parseBoolean("true");
usage of Wrapper class in realtime coding
=====
//WAP to take inputs from the command line and perform arithmetic operations
class Test
{
}
public static void main(String[] args)
{
    //valueOf()  => Converts String/Primitive to Wrapper type
    //xxxValue()  => Converts Wrapper type to Primitive type
    //parseXXX()  => converts String to primitive type
    //commandline arguments => String inputs = args[0],args[1]
    int i1 = Integer.parseInt(args[0]);
    int i2 = Integer.parseInt(args[1]);
    System.out.println(i1+i2);
    System.out.println(i1-i2);
    System.out.println(i1*i2);
    System.out.println(i1/i2);
    //args -> String, convert into primitive type and process
}
form-2
=====
public static primitive parseXXXX(String s, int radix)
    |=> range is from 2 to 36
Every Integral type Wrapper class(Byte,Short,Integer,Long) contains the following
parseXXXX()
to convert Specified radix String to primitive type.
eg: int i=Integer.parseInt("1111",2);
    System.out.println(i);//15
Note: String => parseXXX() => primitive type
toString()
=====
To convert the Wrapper Object or primitive to String.
Every Wrapper class contain toString()
form1
=====
public String toString()
1. Every wrapper class (including Character class) contains the above toString()
method to convert wrapper object to String.
2. It is the overriding version of Object class toString() method.
3. Whenever we are trying to print wrapper object reference internally this
    toString() method only executed
eg: Integer i=Integer.valueOf("10");

```

```
System.out.println(i); //internally it calls toString() and prints the Data.  
form2  
=====  
public static String toString(primitivetype)  
1. Every wrapper class contains a static toString() method to convert primitive to  
String.  
String s=Integer.toString(10);  
      |=> primitive type int.  
eg:  
String s=Integer.toString(10);  
String s=Boolean.toString(true);  
String s=Character.toString('a');  
form3  
=====  
Integer and Long classes contains the following static toString() method to convert  
the  
primitive to specified radix String form.  
public static String toString(primitive p,int radix)  
|=> 2 to 36  
eg: String s=Integer.toString(15,2)  
     System.out.println(s); // 1111  
form4  
=====  
Integer and Long classes contains the following toXxxString() methods.  
public static String toBinaryString(primitive p);  
public static String toOctalString(primitive p);  
public static String toHexString(primitive p);  
Example:  
class WrapperClassDemo {  
    public static void main(String[] args) {  
        String s1=Integer.toBinaryString(7);  
        String s2=Integer.toOctalString(10);  
        String s3=Integer.toHexString(20);  
        String s4=Integer.toHexString(10);  
        System.out.println(s1);//11  
        System.out.println(s2);//12  
        System.out.println(s3);//14  
        System.out.println(s4);//a  
    }  
}  
Note:  
String class  
public static String valueOf(boolean);  
public static String valueOf(char);
```

```
public static String valueOf(int);
public static String valueOf(long);
public static String valueOf(float);
public static String valueOf(double);
String data = String.valueOf('a');//static factory methods
String data = "sachin".toUpperCase();//instance factory methods
AutoBoxing and AutoUnBoxing
=====
untill 1.4Version, we can't provide wrapper class objects in place of primitive and
primitive in place of wrapper object all
the required conversions should be done by the programmer.
But from jdk1.5 Version onwards,we can provide primitive in place of wrapper and in
place of wrapper we can keep primitive
also.All the requird conversion will be done by the compiler automatically, this
mechanism is called as "AutoBoxing" and
"AutoUnBoxing".
eg#1.
Boolean b1 = Boolean.valueOf(true);
if (b1)
    System.out.println("hello");
eg#2.
ArrayList al = new ArrayList();
al.add(10);
```

Autoboxing

```
=====
Automatic conversion of primitive type to wrapper object by the compiler is called
"AutoBoxing".
Integer i1 = 10;
|
|After compilation the code would be
|
|
Integer i1 = Integer.valueOf(10);
Note: Autoboxing is done by the compiler using a method called "valueOf()".
```

AutoUnBoxing

```
=====
Automatic conversion of wrapper object to primitive type by compiler is called
"AutoUnBoxing".
Integer i1 = new Integer(10);
int i2 = i1;
```

```

|
|compiler converts Integer to int type using intValue()
|
int i2 = i1.intValue();
Note: AutoUnboxing is done by the compiler using a method called "xxxValue()"
Case1:
=====
class Test
{
    static Integer i1 = 10;// AutoBoxing
    public static void main(String[] args)
    {
        int i2 = i1;//AutoUnBoxing
        m1(i2);
    }
    public static void m1(Integer i2){//AutoBoxing
        int k = i2;//AutoUnBoxing
        System.out.println(k);//10
    }
}
Compiler is responsible for conversion of primitive to wrapper and wrapper to
primitive using the concept of
"AutoBoxing and AutoUnBoxing".
case2:
class Test
{
    static Integer i1;//i1 = null
    public static void main(String[] args)
    {
        int i2 = i1;// int i2 = i1.intValue() :: NullPointerException
        System.out.println(i2);
    }
}
Case3 :
Integer i1 = 10;//AutoBoxing
Integer i2 = i1;
i1++; => i1 = i1+1
System.out.println(i1);
System.out.println(i2);
System.out.println(i1==i2);
Case4:
Integer x = new Integer(10);
Integer y = new Integer(10);
System.out.println(x == y);//false

```

Case5:

```
Integer x = new Integer(10); //memory from heap area  
Integer y = 10; //AutoBoxing ==> Integer y = Integer.valueOf(10);  
System.out.println(x == y); //false
```

Case6:

```
Integer x = new Integer(10);  
Integer y = x; ==> reference is reused so pointing to same object  
System.out.println(x == y); //true
```

Case7:

```
Integer x = 10;  
Integer y = 10;  
System.out.println(x == y);  
Integer a = 100;  
Integer b = 100;  
System.out.println(a == b);  
Integer i = 1000;  
Integer j = 1000;  
System.out.println(i == j);
```

Note:

1. To implement autoboxing concept in wrapper class a buffer of object will be created at the time of class loading.
2. During AutoBoxing, if an object has to be created first JVM will check whether the object is already available inside buffer or not.
3. If it is available, then JVM will reuse the buffered object instead of creating a new Object.
4. If the Object is not available inside buffer, then JVM will create a new object in the heap area, this approach improves the performance and memory utilization

But this buffer concept is applicable only for few cases

1. Byte => -128 to +127
2. Short => -128 to +127
3. Integer => -128 to +127
4. Long => -128 to +127
5. Character => 0 to 127
6. Boolean => true, false

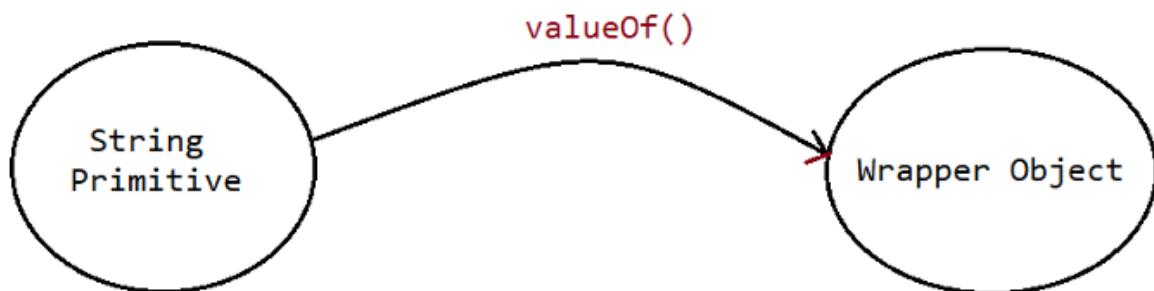
In the remaining cases new object will be created.

```
// String/primitive to wrapper => valueOf()  
// Wrapper type to primitive => xxxValue()  
class Test  
{  
    public static void main(String[] args)  
    {  
        Integer x = 128;
```

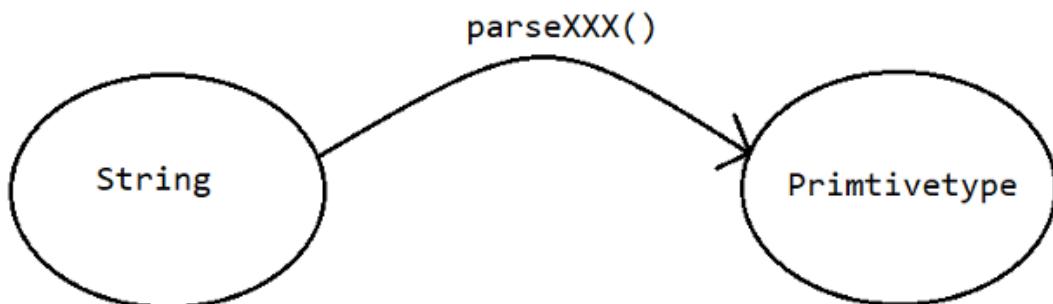
```

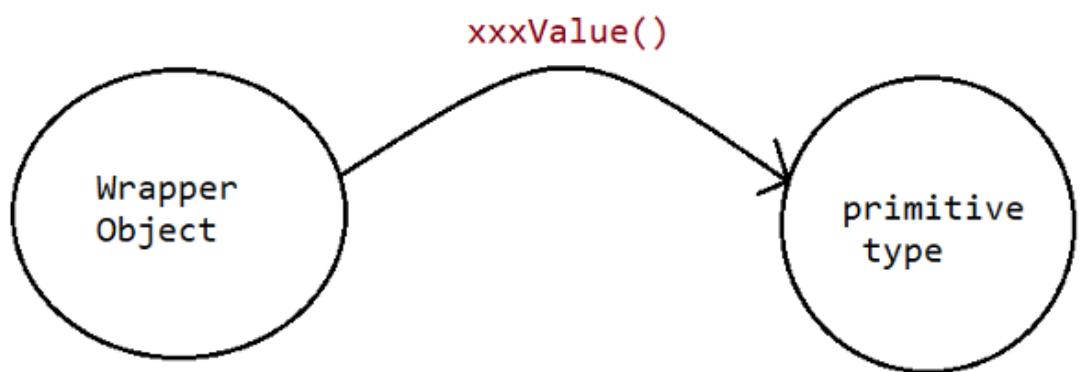
Integer y = 128;
System.out.println(x == y);//false
Integer a = 127;
Integer b = 127;
System.out.println(a == b);//true
Boolean b1 = true;
Boolean b2 = true;
System.out.println(b1==b2);//true
Double d1 = 10.0;
Double d2 = 10.0;
System.out.println(d1==d2);//false
}
}

```



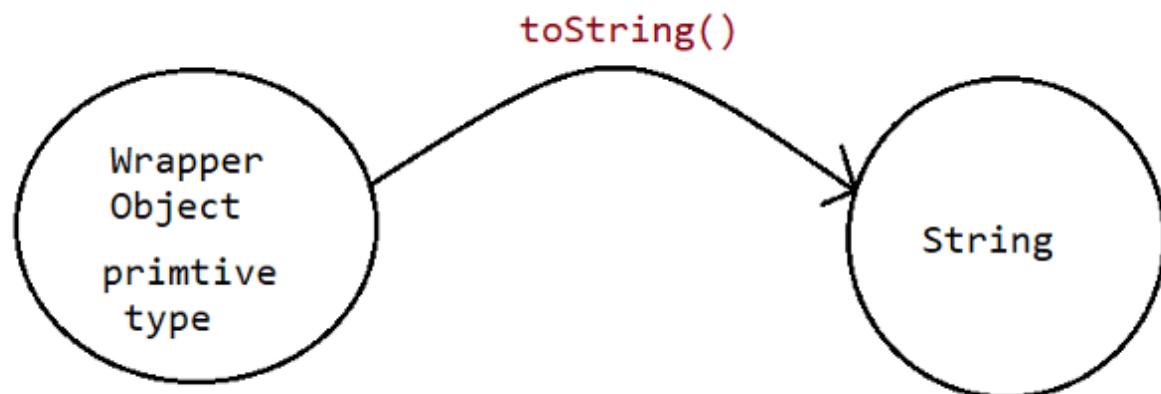
```
public static Character valueOf(char);
```

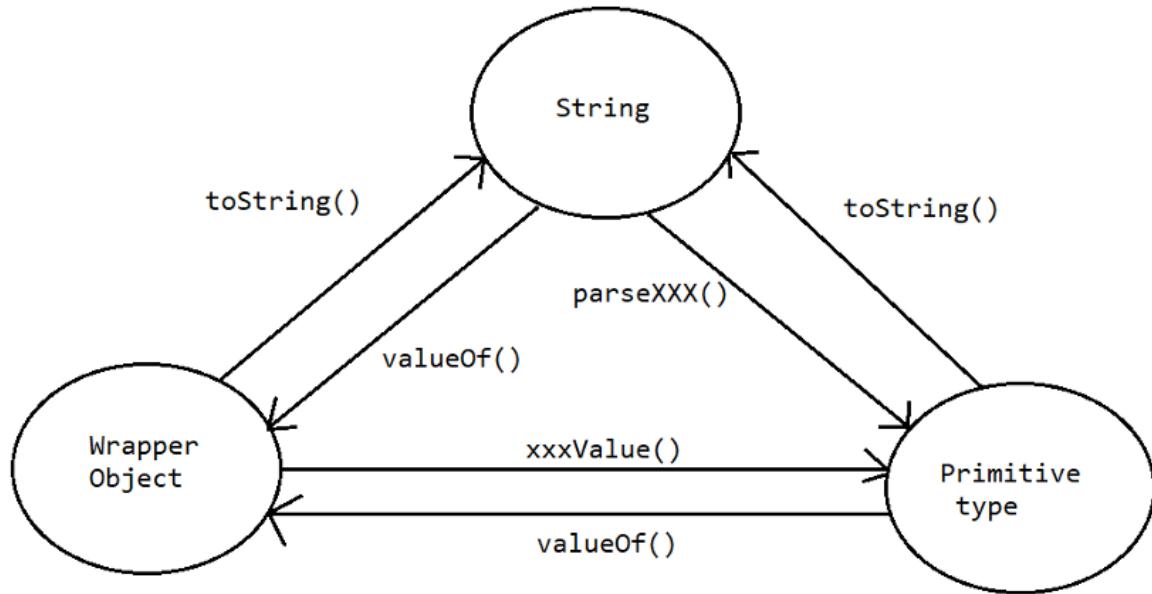




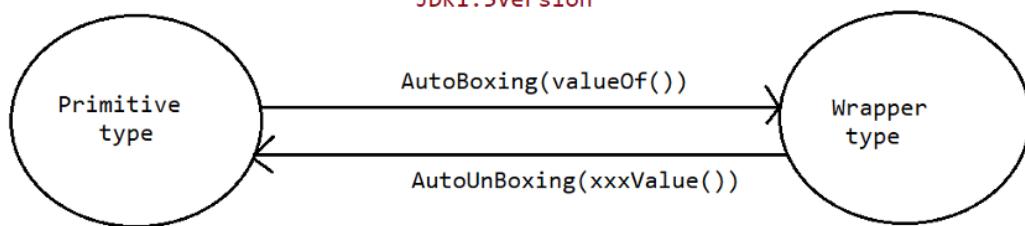
Integer
Byte
Short =>6 xxxValue()
Long
Float
Double

booleanValue()
charValue()





Compiler will do the conversions automatically from
JDK1.5 Version



Integer(Immutable)

```

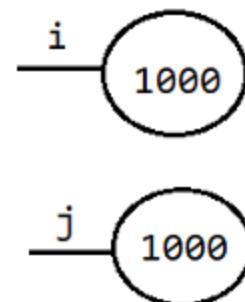
Integer i1 = 10;
Integer i2 = i1;
i1++;
System.out.println(i1); //11
System.out.println(i2); //10
System.out.println(i1==i2); //false
  
```



```
Integer x = 10;  
Integer y = 10;  
System.out.println(x == y); //true
```

```
Integer a = 100;  
Integer b = 100;  
System.out.println(a == b); //true
```

```
Integer i = 1000;  
Integer j = 1000;  
System.out.println(i == j); //false
```



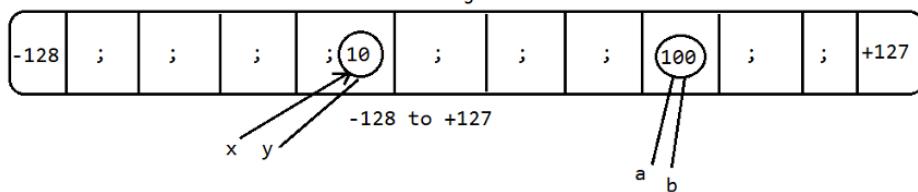
Compiler uses "valueOf()" for AutoBoxing.



Implemented in intelligent way in Wrapper classes

Buffer of Objects

At the time of loading the .class file jvm
will create buffer of object to be used during
AutoBoxing(range : -128 to +127)



45)23-09-22

(no snippets class)

Varargs and New vs NewInstance()

This concepts are used at higher end

Aware about constructors and wrapper class

Wrapper Class

- 1.AutoBoxing
- 2.Widening(implicit typecasting)
- 3.Var-arg approach

Wrapper class is immutable

User defined class can be made as immutable

Snippets

```
Integer i1= new Integer(10);
Integer i2= new Integer(10);
System.out.println(i1==i2);//false
Integer i1 = 10;
Integer i2 = 10;
System.out.println(i1==i2);//true
Integer i1 =Integer.valueOf(10);
Integer i2 =Integer.valueOf(10);
System.out.println(i1==i2);//true
Integer i1 =10;
Integer i2 =Integer.valueOf(10);
System.out.println(i1==i2);//true
```

Note:

When compared with constructors it is recommended to use valueOf() method to create wrapper object.

Method overloading is also called as compile time polymorphism because it binds the method calls at compile time

From JDK 1.0 v method overloading is available

Form jdk 1.4v => change in the no of arguments new method should be written

Form jdk 1.5v => single method can handle any no of arguments (all should be of same type)

Var arg method

It stands for variable argument methods.

In java language, if we have variable no of arguments, then compulsorily new method has to be written till jdk1.4.

But jdk1.5 version, we can write single method which can handle variable no of arguments(but all of them should be of same type).

Syntax:: methodOne(dataType... varaiableName)
... => It stands for ellipse

eg#1.

```
class Demo
{
    //JDK1.5V called Var-args(also ellipse approach)
    public void add(int... x){
        System.out.println("var-arg approach");
    }
}

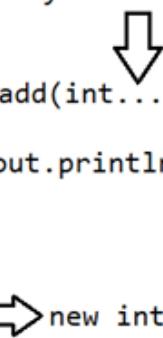
class Test
{
    public static void main(String[] args)
    {
        Demo d =new Demo();
        d.add();
        d.add(10);
        d.add(10,20);
        d.add(10,20,30);//no limit any no; of arguments can be passed.
    }
}
```

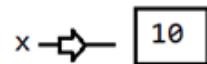
Output

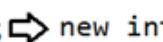
```
var-arg approach
var-arg approach
var-arg approach
var-arg approach
```

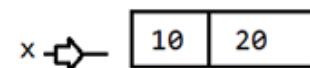
JVM internally uses Array representation to hold the values of x

```
public void add(int... x)
{
    System.out.println("var-arg approach");
}
```

d.add(); → new int[]{}


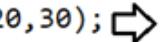


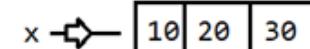
d.add(10); → new int[]{10}




d.add(10,20); → new int[]{10,20}




d.add(10,20,30); → new int[]{10,20,30}




Each time when method is called obj is created

Note:: internally the var arg method will converted to SingleDimension Array, so we can access the var arg method arguments using index.

```
class Demo
{
    public void add(int... x)
    {
        for(int ele : x)
        {
            sop(ele);
        }
    }
}
```

eg2::

```
class Demo
{
    public void methodOne(int... x)
    {
        int total=0;
        for(int i=0;i<x.length;i++){
            total+=x[i];
        }
        System.out.println("The sum is "+total);
    }
}
```

```
}

}

eg3::

    public static void main(String[] args){
        Demo d= new Demo();
        d.methodOne();//The sum is 0
        d.methodOne(10);//The sum is 10
        d.methodOne(10,20,30);// The sum is 60
        class Demo
        {
            public void methodOne(int... x){
                int total =0;
                for(int data:x){total+=data;}
                System.out.println("The sum is "+total);
            }
        }
    }

case1
=====

    public static void main(String[] args){
        Demo d= new Demo();
        d.methodOne();//The sum is 0
        d.methodOne(10);//The sum is 10
        d.methodOne(10,20,30);// The sum is 60
```

Valid Signatures

- 1.public void methodOne(int... x)
- 2.public void methodOne(int...x)
- 3.public void methodOne(int ...x)

case2

=====

We can mix normal argument with var argument

```
public void methodOne(int x,int... y)
public void methodOne(String s,int... x)
```

What if ?

Two var args? in a method call only one var arg is supported

First one has to be normal argument and second or last can be var args

Var args came into picture to overcome problems of method overloading

case3

=====

While mixing var arg with normal argument var arg should be always last.

public void methodOne(int... x,int y); (invalid)

case4

=====

In an argument list there should be only one var argument

public void methodOne(int... x,int ...y); (invalid)

case5

=====

We can overload var arg method, but var arg method will get a call only if none of matches are found.

(just like default statement of switch case)

eg::

class Test

{

 public void methodOne(int ...i) //methodOne(int[] i) =>default stmt

{

 System.out.println("Var arg method");

}

 public void methodOne(int i)

{

 System.out.println("Int arg method");

}

 public static void main(String[] args)

{

 Test t= new Test();

 t.methodOne(10);//Int arg method

 t.methodOne();//Var arg method

 t.methodOne(10,20,30);//Int arg method

}

}

case6

=====

public void methodOne(int... x) => it can be replace as int[] x

case7

=====

public void methodOne(int... x)

public void methodOne(int[] x)

output:: CE because we cannot have two methods with the same signature.

For array replacing ... with [] works
But for all cases it won't work

SingleDimension Array to VarArg method

1. Wherever Singledimesion array is present we can replace it with VarArg.
eg:: public static void main(String[] args) => String... args
2. Wherever Var arg is present we cannot replace it with a SingleDimension Array.
eg:: public void methodOne(String... args) => String[] args (invalid)

Note:

m1(int... x)
=> we can call to this method by passing group of int values and x will
become 1D array(int[] x)
m1(int[] x)
=> we can call this method by passing a 1D array only.

Note::

eg1::
class Test
{
 public void methodOne(int... x){
 for(int data: x){
 System.out.println(data);
 }
 }
 public static void main(String... args){
 Test t= new Test();
 t.methodOne(10,20,30);
 }
}

In the above pgm x is treated as a One-D array.

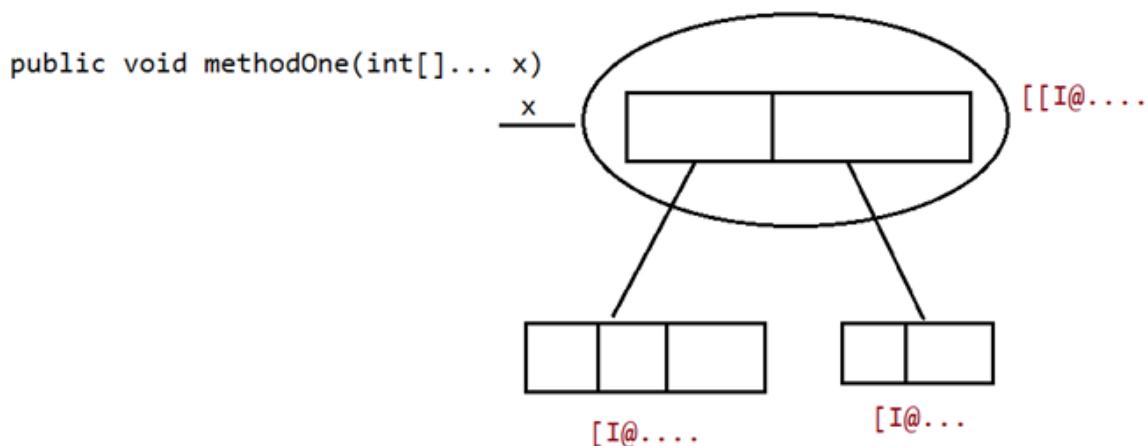
eg2::
class Test
{
 public void methodOne(int[]... x)//array of 1D
 {
 for(int[] OneD: x){
 for(int element:oneD){
 System.out.println(data);
 }
 }
 }
}

```

    }
    public static void main(String... args){
        Test t= new Test();
        int[] a= {10,20,30};
        int[] b= {30,40};
        t.methodOne(a,b);
    }
}

```

In the above program x is treated as 2D array



Note::

methodOne(int...x)

=> we can call this method by passing a group of int values, so it becomes a 1-D array.

methodOne(int[]... x)

=> we can call this method by passing a group of 1D int[], so it becomes a 2-D array.

-----Var Args completed-----

Wrapper class

=====

1. AutoBoxing
2. Widening(Implicit TypeCasting done by the Compiler (applicable for both primitive and Wrapper type))
3. Var-Args

case1::

Widening vs Autoboxing(value of)

class AutoBoxingAndUnboxingDemo

{

 public static void methodOne(long l)

 {

 System.out.println("widening");

```

    }
    public static void methodOne(Integer i)
    {
        System.out.println("autoboxing");
    }
    public static void main(String[] args) {
        int x=10; //implicit type casting => long
        methodOne(x); // primitive =====> do typecasting=====> found ==>
        long(binding happens by compiler)
    }
}
output: widening

```

case2::Widening vs var-arg method

```

class AutoBoxingAndUnboxingDemo
{
    public static void methodOne(long l)
    {
        System.out.println("widening");
    }
    public static void methodOne(int... i) //varargs will be given last chance
    {
        System.out.println("var-arg method");
    }
    public static void main(String[] args) {
        int x=10;
        methodOne(x); // primitive =====> do type casting=====>
        found ==> long(binding happens by compiler)
    }
}
output: widening

```

Case 3: Autoboxing vs var-arg method :

```

class AutoBoxingAndUnboxingDemo {
    public static void methodOne(Integer i) {
        System.out.println("AutoBoxing");
    }
    public static void methodOne(int... i) {
        System.out.println("var-arg method");
    }
    public static void main(String[] args) {
        int x=10;
        methodOne(x); // int =====> implicit type casting====>
        long,float,double
    }
}

```

```
//int => autoboxing => Integer  
}  
}  
}  
Output: AutoBoxing
```

Case4:

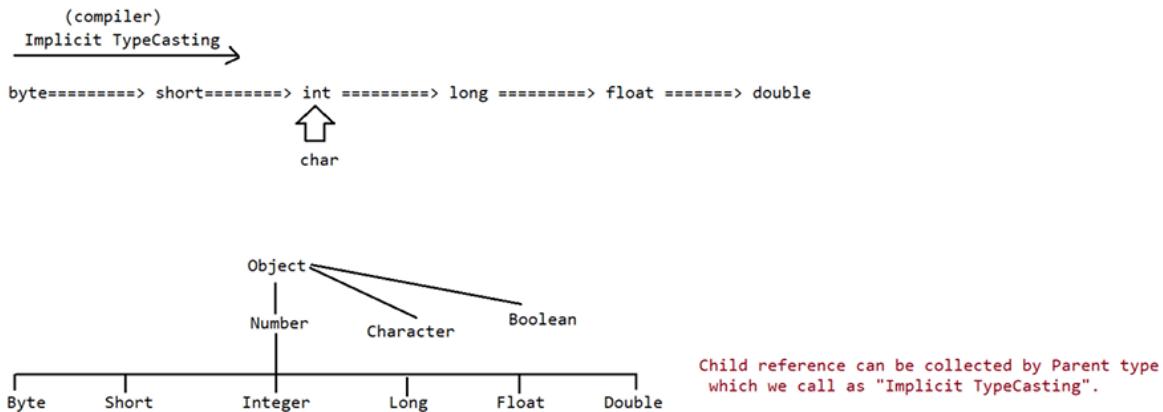
```
// int =====> Autoboxing =====> Integer  
class AutoBoxingAndUnboxingDemo {  
public static void methodOne(Long l) {  
System.out.println("Long");  
}  
public static void main(String[] args) {  
int x=10;  
methodOne(x); //CE: can't find the method  
}  
}
```

Note:

Widening followed by Autoboxing is not allowed in java, but Autoboxing followed by widening is allowed.

Case 5:

```
class AutoBoxingAndUnboxingDemo {  
public static void methodOne(Object o) {  
System.out.println("Object");  
}  
public static void main(String[] args) {  
int x=10;  
methodOne(x); // AutoBoxing =====> int =====> Integer  
// Widening(typecasting) =====> Integer =====> Number, Object  
}  
}  
Output: Object
```



In java, whenever binding happens, first autoboxing is preference later widening

Which of the following declarations are valid ?

1. int i=10 ;//valid
 2. Integer I=10 ; //AutoBoxing(valueof())
 3. int i=10L ; // invalid(long==> int)
 4. Long l = 10L ; //AutoBoxing(valueOf())
 5. Long l = 10 ; //AutoBoxing=> Integer ==> Number, Object , so invalid
 6. long l = 10 ; // valid(int==> long)
 7. Object o=10 ; //AutoBoxing==> Integer ==> Number, Object so valid
 8. double d=10 ; // valid(int==> double)
 9. Double d=10 ; // AutoBoxing=> Integer==> Number, Object , so invalid
 10. Number n=10; //AutoBoxing==> Integer ==> Number, Object so valid
-
- wrapper classes completed-----

new Vs newInstance()

-
1. new is an operator to create an objects , if we know class name at the beginning then we can create an object by using new operator .
 2. newInstance() is a method presenting class " Class " , which can be used to create object.
 3. If we don't know the class name at the beginning and its available dynamically Runtime then we should go for

 newInstance() method

eg#1.

```
public class Test {
    public static void main(String[] args) throws Exception {
        //Take the input of the classname for which object has to be created at
```

```

the runtimeonly
}
}
String className = args[0];

//Load the class file explicitly
Class c = Class.forName(className);

//for the loaded class object is created using zero param constructor
Object obj=c.newInstance();

//Perform typecasting to get Student Object
Student std = (Student)obj;
System.out.println(std);

```

If dynamically provide class name is not available then we will get the RuntimeException saying ClassNotFoundException
To use newInstance() method compulsory corresponding class should contains no argument constructor , otherwise we will get the RuntimeException saying "InstantiationException".
if the argument constructor is private then it would result in "IllegalAccessException".

Note: During typecasting, if there is no relationship b/w 2 classes as parent to child then it would result in "ClassCastException".

```

class Student
{
    Static{
        sop("student .class file is loading");
    }
    public Student(){
        sop("Student constructor is called");
    }
}
public class Test
{
    public static void main(String[] args) throws ClassNotFoundException,
    InstantiationException, IllegalAccessException
    {
        //      //create an object of student class
        //      Student std1 = new Student();
    }
}

//Take the input of the class for which obj has to be created at the runtime

```

```

String className = args[0];

//load the class file explicitly
Class c = Class.forName(className);

//if loaded class doesn't have it leads to exception InstantiationException
//if constructor is private then IllegalAccessException

//for loaded class create an instance
//for the loaded class object is created using zero param constructor only
Object obj = c.newInstance();

//newInstance() method calls the zero param constructor
/*
//not possible
args[0] obj1 = new args[0];
className obj2 = new className();
*/

//perform type casting to get Student Object
Student std = (Student)obj; //Student ⊑ String(no relation learns to runtime error)
System.out.println(std);

/*
new will create a memory on the heap area
Student ⇒ JVM will search for student.class file in current working directory if found load the
.class file data into MethodArea

During the loading of .class file
A. static variables will get memory set with default value
B. static block gets executed.

In the heap area , for the required obj memory for instance var's is given jvm will set the default
values to it
A. Execute the instance block if available
B. Call the constructor to set the meaningful values to the instance variables.

JVM will give the address of the obj to hashing algorithm which generates the hashCode for the
obj and the hashCode will be returned as the reference to the programmer.

*/
}

>>javap java.lang.Class
>>java Test java.lang.String
Error : classCastException

```

```
/*
new ⇒ required class details known to compiler but not available at jvm then it would result in
"NoClassDefFoundError"

newInstance() ⇒ required class details not available at jvm then it would result in
"ClassNotFoundException"
*/
```

Difference between new and newInstance() :

new

====

new is an operator , which can be used to create an object.

We can use new operator if we know the class name at the beginning.

Test t= new Test();

If the corresponding .class file not available at Runtime then we will get
RuntimeException saying NoClassDefFoundError,It is unchecked.

To used new operator the corresponding class not required to contain no argument Constructor.

newInstance()

=====

newInstance() is a method , present in class Class , which can be used to create
an object .

We can use the newInstance() method , If we don't class name at the beginning and
available dynamically Runtime.

Object of=Class.forName(arg[0]).newInstance();

If the corresponding .class file not available at Runtime then we will get
RuntimeException saying ClassNotFoundException , It is checked.

To used newInstance() method the corresponding class should compulsory contain no
argument constructor , Otherwise we will get
RuntimeException saying InstantiationException.

Difference between ClassNotFoundException & NoClassDefFoundError :

1. For hard coded class names at Runtime in the corresponding .class files not
available we will get NoClassDefFoundError ,
which is unchecked

Test t = new Test();

In Runtime Test.class file is not available then we will get

NoClassDefFoundError

2. For Dynamically provided class names at Runtime , If the corresponding .class files is not
available then we will get the

RuntimeException saying "ClassNotFoundException".

Ex : Object of=Class.forName("Test").newInstance();

At Runtime if Test.class file not available then we will get the

`ClassNotFoundException` , which is a checked exception.

Note:

`new` will create a memory on the heap area

`Student => JVM will search for Student.class file in Current Working Directory`

`if found load the .class file data into MethodArea`

`During the loading of .class file`

a. static variables will get memory set with default value

b. static block gets executed

In the heap area, for the required object memory for instance variables is given

jvm will set the default values to it

a. Execute the instance block if available

b. call the constructor to set the meaningful values to the instance

variables.

JVM will give the address of the object to hashing algorithm which generates the `hashCode` for the object and that `hashCode` will be returned as the reference to the programmer

`new => required class details known to compiler but not available at jvm then it would result in "NoClassDefFoundError"`

`newInstance() => required class details not available at jvm then it would result in "ClassNotFoundException"`

```
class A extends Thread
{
    public void m1(){
        A a = new A();
        sysout(a.getClass());//A
    }
}
public class Test
{
    static {
        sop("Test.class file is loading by jvm");
    }
    public static void main(String[] arg) throws Exception{
        new A().m1();
    }
}
```

46)23-11-22 dbt

47)24-11-22

(Topic's = import stmt , static import , packages and its usage , instanceof vs isInstance())

Import statements

```
ArrayList al = new ArrayList(); //error  
//because it is not available in current working directory
```

```
/*  
Javac ⇒ search for the required class information in  
    a. Cwd  
    b. Dig the programmer specified where the class is available  
*/
```

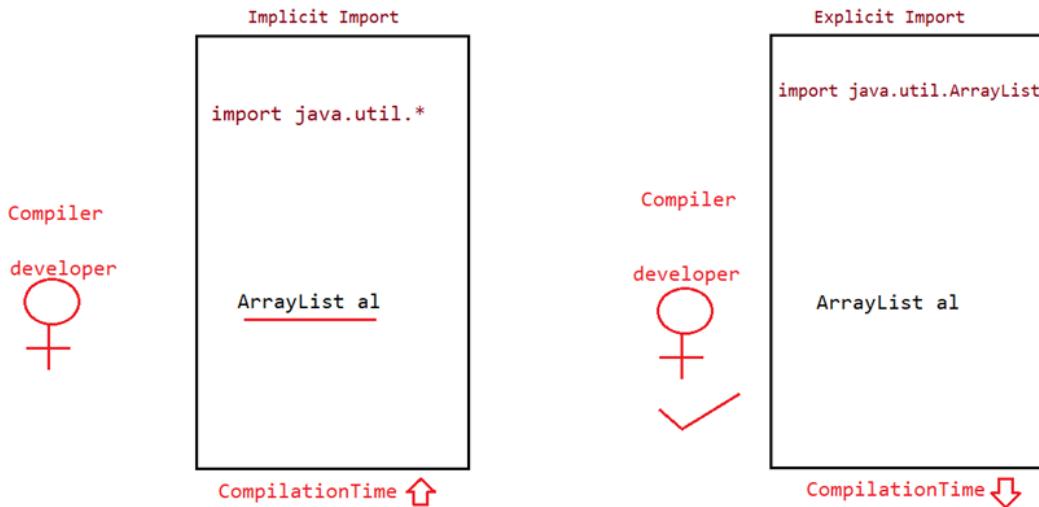
```
java.util.ArrayList al = new java.util.ArrayList(); //fully qualified path
```

Fully qualified paths disturb the readability

So,

Import it at first

```
import java.util.ArrayList; //Explicit import = compilation time is low  
//informing the compiler to search for ArrayList class in "java.util"  
import java.util.*; //Implicit import = compilation time is more
```



At JVM no difference, performance side.
For developers explicit import is good.

```
class Test{
```

```
public static void main(String args[]){
ArrayList l=new ArrayList();
}
}
```

Output:

Compile time error.

```
D:\Java>javac Test.java
Test.java:3: cannot find symbol
symbol : class ArrayList
location: class Test
```

ArrayList l=new ArrayList();

=> We can resolve this problem by using fully qualified name "java.util.ArrayList l=new java.util.ArrayList();". But problem with using fully qualified name every time is it increases length of the code and reduces readability.

=> We can resolve this problem by using import statements.

Example:

```
import java.util.ArrayList;
class Test{
    public static void main(String args[])
    {
        ArrayList l=new ArrayList();
    }
}
```

Output:

```
D:\Java>javac Test.java
```

Hence whenever we are using import statement it is not require to use fully qualified names we can use short names directly.

This approach decreases length of the code and improves readability.

Case 1: Types of Import Statements:

There are 2 types of import statements.

- 1) Explicit class import
- 2) Implicit class import.

Explicit class import:

Example: Import java.util.ArrayList ;

=> This type of import is highly recommended to use because it improves readability of the code.

=> Best suitable for developers where readability is important.

Implicit class import:

Example: import java.util.*;
=> It is never recommended to use because it reduces readability of the code.
=> Best suitable for students where typing is important.

Case 2:

Which of the following import statements are meaningful ?

- a. import java.util;
- b. import java.util.ArrayList.*;
- c. import java.util.*;
- d. import java.util.ArrayList;

Answer: c and d

Case3:

consider the following code.

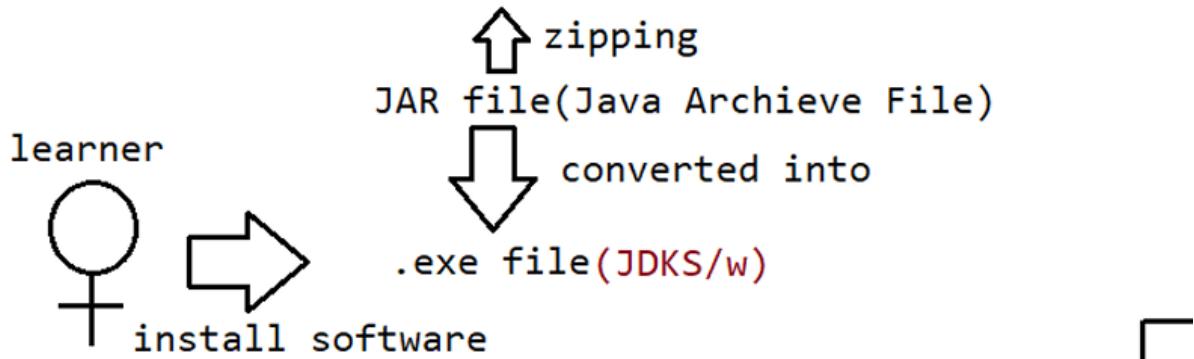
```
class MyArrayList extends java.util.ArrayList  
{  
}
```

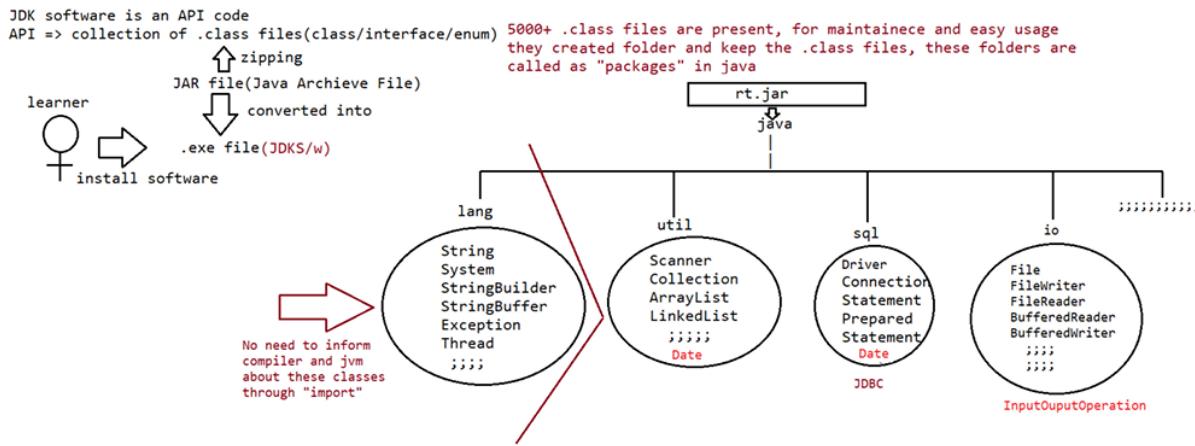
=> The code compiles fine even though we are not using import statements because we used a fully qualified name.

=> Whenever we are using a fully qualified name it is not required to use an import statement.

Similarly whenever we are using import statements it is not required to use a fully qualified name.

JDK software is an API code
API => collection of .class files(class/interface/enum)





All day to day using common and basic classes are present in lang package.(no need to import)(that are present by default)

Case4:

```
import java.util.*;
import java.sql.*;
class Test{
public static void main(String args[]) {
Date d=new Date();
}
}
Output:
Compile time error.
```

D:\Java>javac Test.java

Test.java:7: reference to Date is ambiguous,
both class java.sql.Date in java.sql and class java.util.Date in java.util match
Date d=new Date();

Note: Even in the List case also we may get the same ambiguity problem because it is available in both util and awt packages.

Then explicit import is best and fully qualified paths are used.

Case5:

While resolving class names compiler will always gives the importance in the following order.

1. Explicit class import
2. Classes present in current working directory.
3. Implicit class import.

Example:

```
import java.util.Date;
import java.sql.*;
class Test {
public static void main(String args[]){}
```

```
Date d=new Date();  
}  
}
```

The code compiles fine and in this case util package Date will be considered.

Case 6:

Whenever we are importing a package all classes and interfaces present in that package are by default available but not sub package classes.

```
java  
|=> util  
    |=> Scanner.class,ArrayList.class,LinkedList.class  
    |=> regex  
        |=> Pattern.class
```

To use Pattern class in our Program directly which import statement is required ?

- a. import java.*;
- b. import java.util.*;
- c. import java.util.regex.*; //valid(implicit import)
- d. import java.util.regex.Pattern;//valid(explicit import)

Note:

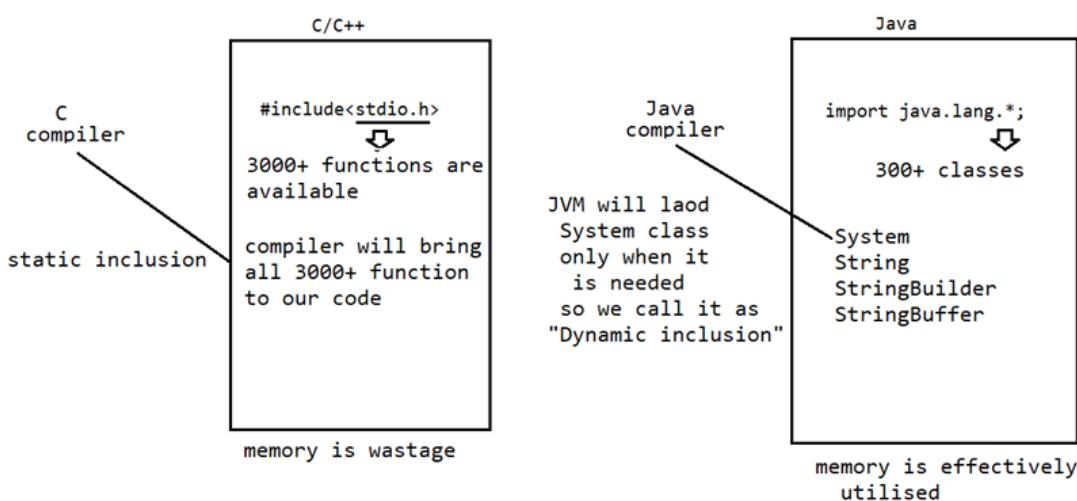
* => it refers to only .class files not subpackages .class files

Case7:

In any java Program the following 2 packages are not require to import because these are available by default to every java Program.

1. java.lang package
2. default package(current working directory)

Case 8:



"Import statement is totally compile time concept" if more no of imports are there then more will be the compile time
but there is "no change in execution time".

Difference between C language #include and java language import ?

```
#include
=====
1. It can be used in C & C++
2. At compile time only compiler copy the code from standard library and placed in current program.
3. It is static inclusion
4. wastage of memory
Ex : <jsp:@ file="">
```

```
import
=====
1. It can be used in Java
2. At runtime JVM will execute the corresponding standard library and use it's result in current program.
3. It is dynamic inclusion
4. No wastage of memory
Ex : <jsp:include >
```

Note:

In the case of C language #include all the header files will be loaded at the time of include statement hence it follows static loading.

But in java import statement no ".class" will be loaded at the time of import statements in the next lines of the code whenever we are

using a particular class then only corresponding ".class" file will be loaded.

Hence it follows "dynamic loading" or "load-on -demand" or "load-on-fly".

JDK 1.5 versions new features :

1. For-Each
2. Var-arg
3. Queue
4. Generics
5. Auto boxing and Auto unboxing
6. Co-varient return types
7. Annotations
8. Enum
9. Static import
10. String builder

Static import:

Eg: `javap java.lang.Math` all methods are static.if a class contains all methods are static then that class is called as helper class and methods also called as helper methods or utility methods.

This concept was introduced in 1.5 versions. According to sun static import improves readability of the code but according to

worldwide Programming exports (like us) static imports creates confusion and reduces readability of the code. Hence if there is no specific requirement never recommended to use a static import.

Usually we can access static members by using class name but whenever we are using static import it is not required to use class name we can access directly.

Without static import:

```
class Test
{
    public static void main(String args[]){
        System.out.println(Math.sqrt(4));
        System.out.println(Math.max(10,20));
        System.out.println(Math.random());
    }
}
```

Output:

```
D:\Java>javac Test.java
D:\Java>java Test
2.0
20
0.841306154315576
```

With static import:

```
import static java.lang.Math.sqrt;
import static java.lang.Math.*;
class Test{
    public static void main(String args[]){
        System.out.println(sqrt(4));
        System.out.println(max(10,20));
        System.out.println(random());
    }
}
Output:
D:\Java>javac Test.java
```

```
D:\Java>java Test
2.0
20
0.4302853847363891
(static import is the flop concept of jdk 1.5v)
```

```
class Test{
static String name = "sachin";
}
output
=====
Test.name.length() =====> 6

import java.io.PrintStream;
class System{
static PrintStream out;
}
class PrintStream{
public void println(){
;;;;;
}
}
System.out.println()
|       |       |
|       |       |=> It is a method of PrintStream class
|       |
|       |=> It is a reference of PrintStream Class
|=> it is an class
```

Example 3:

```
import static java.lang.System.out;
class Test{
public static void main(String args[]){
out.println("hello");
out.println("hi");
}}
```

Output:

```
D:\Java>javac Test.java
D:\Java>java Test
hello
hi
```

Example 4:

```
import static java.lang.Integer.*;
import static java.lang.Byte.*;
class Test{
public static void main(String args[]){
System.out.println(MAX_VALUE);
}
}
Output:
Compile time error.
```

```
D:\Java>javac Test.java
Test.java:6: reference to MAX_VALUE is ambiguous,
both variable MAX_VALUE in java.lang.Integer and variable MAX_VALUE in
java.lang.Byte match
System.out.println(MAX_VALUE);
```

Note:

Two packages contain a class or interface with the same name is very rare hence ambiguity problems are very rare in normal imports.

But 2 classes or interfaces can contain a method or variable with the same name is very common hence ambiguity the problem is also very common in static imports.

While resolving static members compiler will give the precedence in the following order.

1. Current class static member
2. Explicit static import
3. implicit static import

eg:

```
import static java.lang.Integer.MAX_VALUE;
import static java.lang.Byte.*;
class Test{
static int MAX_VALUE = 999;
public static void main(String[] args){
System.out.println(MAX_VALUE);
}
}
```

Which of the following import statements is valid?

- import java.lang.Math.*; //invalid
- import static java.lang.Math.*;//valid
- import java.lang.Math;//valid
- import static java.lang.Math;//invalid
- import static java.lang.Math.sqrt.*;//invalid
- import java.lang.Math.sqrt;//invalid

```
import static java.lang.Math.sqrt();//invalid
import static java.lang.Math.sqrt; //valid
Usage of static import reduces readability and creates confusion hence if there is no specific
requirement never recommended to use
static import.
```

What is the difference between general import and static import ?

normal import

=====

=> We can use normal imports to import classes and interfaces of a package.
=> whenever we are using normal imports we can access classes and interfaces directly by
their short name. It is not required to use fully qualified names.

static import

=====

=> We can use static import to import static members of a particular class.
=> whenever we are using static import it is not required to use class name we can access static
members directly.

Functional Interface

=====
Lambda expression ⇐ functional interface ⇐ interface ⇐ abstract ⇐ inheritance

interface Demo

```
{//default methods are public and abstract
    void disp();
}
```

interface Demo

```
{//default methods are public and abstract
    void disp();
//    void disp1();
//    int length();
//    void disp(String name);
}
```

=if an interface contains only one method, then that interface is called as functional interface.
If there is more than one method then it is not functional interface

=if it is functional interface then lambda expression can be written using it.

Whenever an interface is functional interface then use an annotation

@Functional Interface

```
//functional interface
@interface Demo
{
    void disp();
}

//(lambda expr and functional interface works together)
//(we can't write lambda expr without functional interface)
//      arrow operator -> it is lambda operator from JAVA 8
//To write lambda expression operator is invented
//Lambda expression is an anonymous method(unnamed method)
```

Left of lambda operator = parameters

Right of lambda operator = body

```
()->{ sop("hello"); }
```

If lambda expression contains single stmt then curly braces are optional

```
()->sop("hey");
```

Lambda expressions can never be written alone

It is depended on functional interface.

```
@FunctionalInterface
interface Demo
{
    void disp();
}
public class Test
{
    public static void main(String[] args)
    {
        Demo d = ()->sop("hello");
        d.disp();
    }
}
```

o/p
hello

⇒java also supports the functional programming
One class can implement multiple interface.

Can multiple interfaces implementation possible in anonymous class.

```
@FunctionalInterface
interface Add{
    void add();
}
@FunctionalInterface
interface Sub{
    void sub();
}
public class LambdaExpr{
    public static void main(String[] args)
    {
        Add add = (a, b)->
        {
            int res = a+b;
            sop(res);
        };
        //if parameter is one then parenthesis and type of data are optional
        Sub s = num1 ->
        {
            int res = num1-5;
            return res;
        };
        sop(s.sub(10));
    }
}
```

48)25-11-23

(exception handling part 1)

Method hiding

```
class Parent
{
    public static void disp()
    {
        System.out.println("hello parent");
    }
}
class Child extends Parent
{
    public void disp()
```

```

    {
        System.out.println("hello child1");
    }
}

//static methods do participates in inheritance
//if static method is overridden then it calls parent type.
//this concept is called method hiding
public class LaunchMH {

    public static void main(String[] args) {
        Parent p = new Child();
        p.disp();
    }
}

```

Can we inherit static methods ?

Yes, but parent type is called.that is only called the method hiding.

Exception Handling

(important concept)

Internet is connecting devices

Web is a services built on the top of internet

```

void disp()
{
    disp();
}

```

On stack frames are created,util the capacity of stack, it least to overflow.

If there is Abnormal termination.

During early day , not much both

In 1970's 1 tier architecture

In 1980's 2 tier architecture

In 1990's 3's tier architecture (INTERNET)

Due to geo political scenario.

During runtime due to faulty inputs our application should not terminate abnormally so exception handling concept came into picture

```
import java.util.Scanner;

public class Ex1 {

    public static void main(String[] args) {
        System.out.println("Connection to CALC App is established");

        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the first num to divide : ");
        int num1 = sc.nextInt();
        System.out.println("Enter the second num to divide : ");
        int num2 = sc.nextInt();

        int res = num1/num2 ;
        System.out.println("the res after dividing is "+res);

        System.out.println("Connection is terminated");
    }
}
```

output

```
Connection to CALC App is established
Enter the first num to divide :
10
Enter the second num to divide :
0
Exception in thread "main" java.lang.ArithmaticException: / by zero
at Ex1.main(Ex1.java:14)
```

It is only called abnormal termination during the runtime.
Mistake occurs during the runtime.

⇒ As a developer we need to be careful so that abnormal termination will not occur

EXCEPTION

Exception is a mistake that occurs during runtime or an unwanted event or unusual event which will result in abnormal termination of application

EXCEPTION HANDLING

Exception Handling is the process of handling exceptions in such a manner that abnormal termination of application should not happen.

Methods are brought into the runtime stack and executed.

In whichever method exception occurs that method creates an exception object.(that particular stack frame will send to JVM and JVM will give the exception object if dev is not given)

If the developer won't handle the exception then JVM sends it to the default exception handler.

Exception Hierarchy ⇒ Inbuilt classes ⇒ using we can handle exception

How do we handle exceptions ?

Inbuilt class we use between try and catch.
(try catch)

Two types of Exceptions

=>Checked Exception
=such an exception where compiler only identifies the exception
=>Unchecked Exception
=compiler will not give any warning.As a developer we need to identify it.

Auto compiler is present in eclipse

```
try{  
    //risky code  
}  
catch(inbuilt classes)  
{  
    //Handler  
}
```

Think and find risky points in code

Anticipate it

Write entire risky codes inside the try block

```
//import java.util.Scanner;  
//  
//public class Ex1 {  
//  
//    public static void main(String[] args) {  
//        System.out.println("Connection to CALC App is established");  
//  
//        Scanner sc = new Scanner(System.in);  
//        System.out.println("Enter the first num to divide : ");  
//        int num1 = sc.nextInt();  
//        System.out.println("Enter the second num to divide : ");  
//        int num2 = sc.nextInt();  
//  
//        int res = num1/num2 ;  
//        System.out.println("the res after dividing is "+res);  
//    }  
//}
```

```

//                                     System.out.println("Connection is terminated");
//      }
//
//}
//lets handle the exception
import java.util.Scanner;

public class Ex1 {

    @SuppressWarnings("resource")
    public static void main(String[] args) {
        System.out.println("Connection to CALC App is established");
        try {
            Scanner sc = new Scanner(System.in);
            System.out.println("Enter the first num to divide : ");
            int num1 = sc.nextInt();
            System.out.println("Enter the second num to divide : ");
            int num2 = sc.nextInt();

            int res = num1/num2 ;
            System.out.println("the res after dividing is "+res);
        }
        catch(Exception e)
        {
            System.out.println("You're trying to divide by Zero.illogical");
        }
        System.out.println("Connection is terminated");
    }
}

```

Connection to CALC App is established

Enter the first num to divide :

10

Enter the second num to divide :

0

You're trying to divide by Zero.illogical

Connection is terminated

Only if there is exception then only catch block is executed

ArithmaticException

NegativeArraySizeException

ArrayIndexOutOfBoundsException

```
import java.util.Scanner;

public class Ex2 {

    @SuppressWarnings("resource")
    public static void main(String[] args) {
        System.out.println("Connection to CALC App is established");
        try {
            Scanner sc = new Scanner(System.in);
            System.out.println("Enter the first num to divide : ");
            int num1 = sc.nextInt();
            System.out.println("Enter the second num to divide : ");
            int num2 = sc.nextInt();

            int res = num1/num2 ;
            System.out.println("the res after dividing is "+res);
            System.out.println("Enter the size of an array");
            int size = sc.nextInt();
            int[] a = new int[size];

            System.out.println("Enter the element to be inserted in the array");
            int elem = sc.nextInt();

            System.out.println("Enter the position at which elem has to be inserted ");
            int pos = sc.nextInt();

            a[pos] = elem;
            System.out.println("element "+elem+"inserted at position "+pos+" sucessfully");

        }
        catch(Exception e)
        {
            System.out.println("Wrong input please give right one");
        }
        System.out.println("Connection is terminated");
    }
}
```

⇒for different type of mistake, it has to give different exception.
But in above program.common exception is used

Single try block and multiple catch blocks
And give generic catch block.

Ref below code

```
import java.util.Scanner;

public class Ex2 {

    @SuppressWarnings("resource")
    public static void main(String[] args) {
        System.out.println("Connection to CALC App is established");
        try {
            Scanner sc = new Scanner(System.in);
            System.out.println("Enter the first num to divide : ");
            int num1 = sc.nextInt();
            System.out.println("Enter the second num to divide : ");
            int num2 = sc.nextInt();

            int res = num1/num2 ;
            System.out.println("the res after dividing is "+res);
            System.out.println("Enter the size of an array");
            int size = sc.nextInt();
            int[] a = new int[size];

            System.out.println("Enter the element to be inserted in the array");
            int elem = sc.nextInt();

            System.out.println("Enter the position at which elem has to be inserted ");
            int pos = sc.nextInt();

            a[pos] = elem;
            System.out.println("element "+elem+"inserted at position "+pos+" sucessfully");

        }
        catch(ArithmaticException ae)
        {
            System.out.println("please provide non zero denominator");
        }
        catch(NegativeArraySizeException nae)
        {
            System.out.println("please be positive");
        }
        catch(ArrayIndexOutOfBoundsException aie)
        {
```

```

        System.out.println("Please give input in limit");
    }
    catch(Exception e)
    {
        System.out.println("wrong input!");
    }
    System.out.println("Connection is terminated");
}
}

```

Generic exception can't be provided at the beginning.

25) Code Snippets

Given:

```

1. public class Barn {
2.
public static void main(String[] args) {
3.
4.
5.
6.
7.
8.
9. }
new Barn().go("hi", 1);
new Barn().go("hi", "world", 2);
}
public void go(String... y, int x) {
System.out.print(y[y.length - 1] + " ");
}

```

What is the result?

- A. hi hi
- B. hi world
- C. world world
- D. Compilation fails.
- E. An exception is thrown at runtime.

Answer: D

Q>

What is the result?

```

11. public class Person {
12.
String name = "No name";

```

```
13.  
14. }  
15.  
public Person(String nm) { name = nm; }  
16. public class Employee extends Person {  
17.  
String empID = "0000";  
18.  
19. }  
20.  
public Employee(String id) { empID = id; }  
21. public class EmployeeTest {  
22.  
public static void main(String[] args){  
23.  
24.  
25.  
26. }  
}  
Choose the answer
```

- A. 4321
- B. 0000
- C. An exception is thrown at runtime.
- D. Compilation fails because of an error in line 18.

Question

```
1. public class Mud {  
2. //insert code here  
3. System.out.println("hi");  
4. }  
5. }
```

And the following five fragments:

```
public static void main(String...a) {  
public static void main(String.* a) {  
public static void main(String... a) {  
public static void main(String[]... a) {  
public static void main(String...[] a) {
```

How many of the code fragments, inserted independently at line 2, compile?

- A. 0
- B. 1
- C. 2
- D. 3
- E. 4

F. 5

Answer: D

QUESTION

Given:

```
1. class Atom {  
2.  
Atom() {super(); System.out.print("atom ");}  
3.  
4. class Rock extends Atom {  
5.  
Rock(String type) { super(); System.out.print(type);}  
6.  
7. public class Mountain extends Rock {  
8.  
Mountain() {  
9.  
10.  
11.  
12.  
13.}  
super("granite ");  
new Rock("granite ");  
}  
public static void main(String[] a) { new Mountain();}
```

What is the result?

- A. Compilation fails.
- B. atom granite
- C. granite granite
- D. atom granite granite
- E. An exception is thrown at runtime.
- F. atom granite atom granite

Answer:F(constructor chaining) atom granite atom granite

Given:

```
11.  
abstract class Vehicle { public int speed() { return 0; }  
12.  
13.  
21.  
22.  
23  
24  
class Car extends Vehicle { public int speed() { return 60; }  
class RaceCar extends Car { public int speed() { return 150; } ...  
RaceCar racer = new RaceCar();
```

```
Car car = new RaceCar();
Vehicle vehicle = new RaceCar();
System.out.println(racer.speed() + ", " + car.speed() + ", " +
vehicle.speed());
```

What is the result?

- A. 0, 0, 0
- B. 150, 60, 0
- C. Compilation fails.
- D. 150, 150, 150
- E. An exception is thrown at runtime.

Vehicle

```
|=> int speed() => 0
|
|
```

Car

```
|=> int speed() => 60
|
|
```

RaceCar (object)

```
|=> int speed() => 150
```

Answer: D(becoz of overriding jvm will use the runtime object)150,150,150

Question

5. class Building { }

6.

public class Barn extends Building {

7.

8.

9.

10.

11.

12.

13.

14.

15. }

public static void main(String[] args) {

}

Which is true?

Building build1 = new Building();

Barn barn1 = new Barn();

Barn barn2 = (Barn) build1;

Object obj1 = (Object) build1;

String str1 = (String) build1; //RE: ClassCastException

Building build2 = (Building) barn1;

A. If line 10 is removed, the compilation succeeds.

- B. If line 11 is removed, the compilation succeeds.
- C. If line 12 is removed, the compilation succeeds.
- D. If line 13 is removed, the compilation succeeds.
- E. More than one line must be removed for compilation to succeed.

Object =====> obj1

|

Building===== >build1

|

Barn=====> barn1

Answer: C

Given:

```
21. class Money {  
22.  
23.  
24. }  
private String country = "Canada";  
public String getC() { return country; }  
25. class Yen extends Money {  
26.  
public String getC() { return super.country; }  
27. }  
28. public class Euro extends Money {  
29.  
public String getC(int x) { return super.getC(); }  
30.  
31.  
public static void main(String[] args) {  
System.out.print(new Yen().getC() + " " + new  
Euro().getC());  
32.  
33. }  
}
```

What is the result?

- A. Canada
- B. null Canada
- C. Canada null
- D. Canada Canada
- E. Compilation fails due to an error on line 26.
- F. Compilation fails due to an error on line 29.

Answer: E

Q>

Given:

```
1. public class Boxer1{  
2.
```

```
Integer i;
3.
4.
5.
6.
7.
8.
9.
10.
11}
int x;
public Boxer1(int y) {
x = i+y;
System.out.println(x);
}
public static void main(String[] args) {
new Boxer1(new Integer(4));
}
```

What is the result?

- A. The value "4" is printed at the command line.
- B. Compilation fails because of an error in line 5.
- C. Compilation fails because of an error in line 9.
- D. A NullPointerException occurs at runtime.
- E. A NumberFormatException occurs at runtime.
- F. An IllegalStateException occurs at runtime.

y = 4

x = null + 4(jvm do operation on null => NullPointerException)

answer: D

Note:

```
Integer i1= new Integer("ten");//NumberFormatException
```

50)28-11-22

(exception handling part 2)

Whenever there is an Exception

- 1.Handle the Exception
- 2.Duck the Exception
- 3.Rethrowing an Exception

UNCHECKED EXCEPTION (not checked by compiler)

```
import java.util.Scanner;
```

```
class Alpha
{
    void alpha() throws ArithmeticException
    {
        System.out.println("Connection to Calc app is established");

        Scanner scan=new Scanner(System.in);
        System.out.println("Enter the first num to divide");
        int num1=scan.nextInt();
        System.out.println("Enter the 2nd num to divide");
        int num2=scan.nextInt();

        int res=num1/num2;

        System.out.println("The res is "+ res);
        System.out.println("Connection is terminated");
    }
}

class Beta
{
    void beta() throws ArithmeticException
    {
        Alpha a=new Alpha();
        a.alpha();
    }
}

public class LaunchException4 {

    public static void main(String[] args)
    {
        try
        {
            Beta b=new Beta();
            b.beta();
        }
        catch(ArithmeticException e)
        {
            System.out.println(e.getMessage());
            System.out.println("Exception finally handled in main");
        }
    }
}
```

```
    }

}

}
```

CHECKED EXCEPTION (risky code is identified by compiler)

```
//UncheckedException --> Compiler will not force us to handle(It will not
be checked during compile time)
//CheckedException--> it will be checked by compiler
public class LaunchException5 {

    public static void main(String[] args) throws Exception
    {
        // TODO Auto-generated method stub
        System.out.println("Before sleepn");
        Thread.sleep(4000);
        System.out.println("after the nap");

    }
}
```

Checked exception are ducked.

But recommended to handle it.

Ducking means ignoring the exception

Ducking is suitable for checked exceptions

Ducking and handling both are different.

100% confident that exception won't occurs then we need to duck the exception.

Keyword used in java to duck an exception “throws”

Exception Handling

Skeleton

```
try
{
    Risky / Suspicious code
}
catch (Exception e)
{
    Handling / Alternate / Pre Cautionary code
}
finally
{
    Resource deallocation / Clean up code
}
```

Exception handled in alpha

After handling the exception also propagate the exception to the caller

=This concept is called Rethrowing an exception

Throwing or propagating the already handled exception

Throw keyword is used to rethrow and exception

```
import java.util.Scanner;
//Re-throwing an exception
class Alpha1
{
    void alpha() throws ArithmeticException
    {
        System.out.println("Connection to Calc app is established");
        try
```

```
{  
Scanner scan=new Scanner(System.in);  
System.out.println("Enter the first num to divide");  
int num1=scan.nextInt();  
System.out.println("Enter the 2nd num to divide");  
int num2=scan.nextInt();  
  
int res=num1/num2;  
  
System.out.println("The res is "+ res);  
}  
catch(ArithmeticException e)  
{  
System.out.println("Exception handled in alpha only");  
throw e;  
}  
finally {  
System.out.println("Connection is terminated");  
}  
}  
}  
  
public class LaunchException6 {  
  
public static void main(String[] args)  
{  
try  
{  
System.out.println("Main method connection");  
Alpha1 a=new Alpha1();  
a.alpha();  
}  
catch(ArithmeticException e)  
{  
System.out.println("Exception handled in main");  
}  
}  
}  
}
```

Eg : ATM block after multiple wrong input pin

Throw keyword is used to rethrow an exception

Stmts below the throw keyword will never gets executed.

If it is necessary that the below stmts has to execute then write that stmts inside the finally block.

While throw keyword rethrows the exception before that finally block will gets executed.

Finally block will execute even if there is no exception.

If throw is used the throws must has to be used in the signature.

Try and catch = to handle the exception

Throws = duck the exception

Throw = inside the catch block , to rethrow

Finally = close the resource

What does the Exception Object contain ?

Ans. name of the exception :

Description of the Exception :

Stack Trace of the Exception : which frame it occurred and caller

Methods to print Exception Information

Throwable

- getMessage()
 - toString()
 - printStackTrace()
- 1) e.getMessage() : prints the description of the exception
Eg: / by zero
 - 2) e.toString() : prints the name and the description of the Exception
Eg: ArithmeticException : / by zero
 - 3) e.printStackTrace() : prints the name and the description of the Exception along with the stack trace.
Eg: ArithmeticException: / by zero of Demo.alpha()

Finally

- 1) If no exception then executes
- 2) If exception matched then executes

- 3) If exception catch will not match then executes
 At any cost finally block will gets executed because resources has to be closed.

Class Demo

```
{
    int disp()
    {
        try{
            sop("method started");
            return 10;
        Finally{
            sop("method ended");
        }
    }
}
```

Exception Raised	Exception Handled	finally block Executed
NO	—	YES
YES	YES	YES
YES	NO	YES

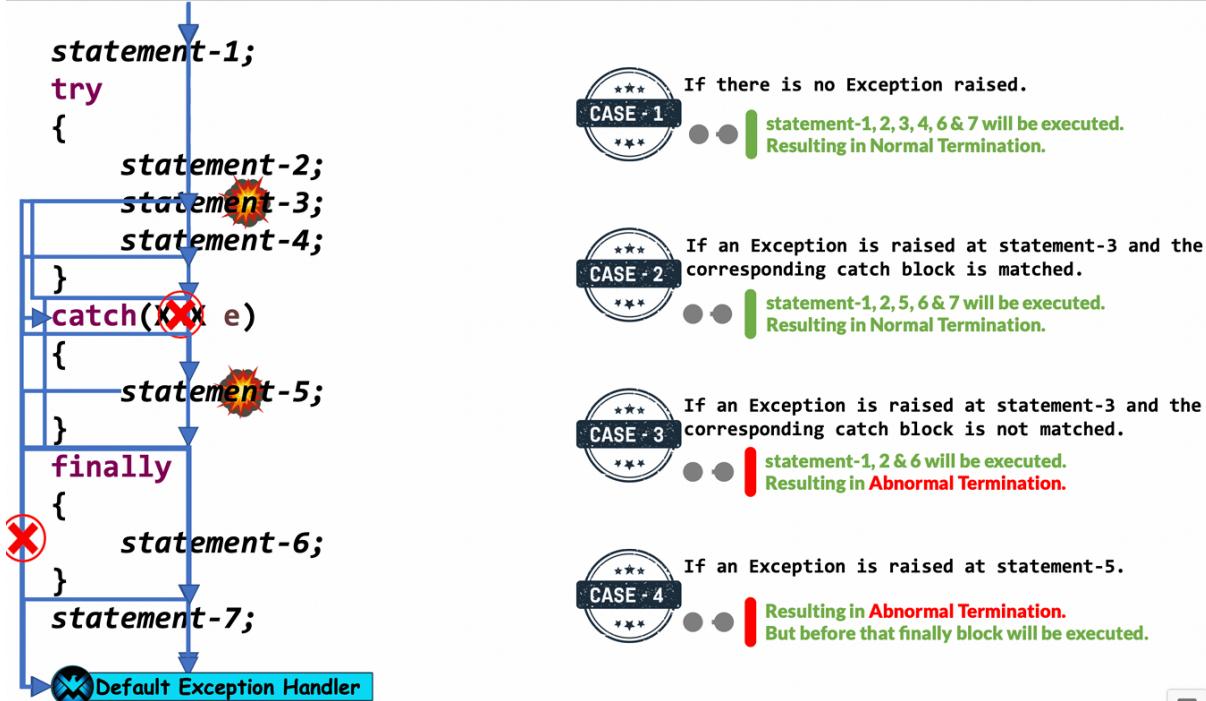
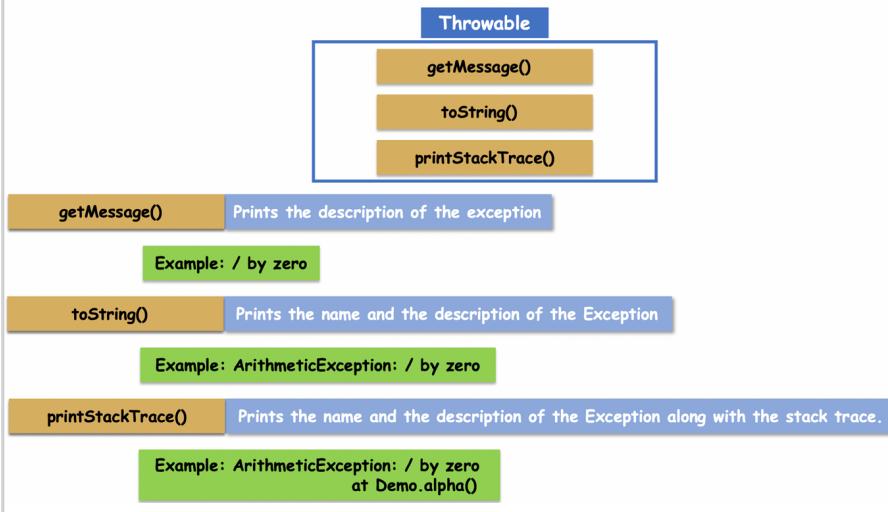
system.exit(0) will dominate the finally block

Class Demo

```
{
    void disp()
    {
        try{
            sop("method started");
            System.exit(0);
        }
        Finally{
            sop("method ended");
        }
    }
}
```

Finally block can't be written alone

Methods to Print Exception Information



Exception Hierarchy

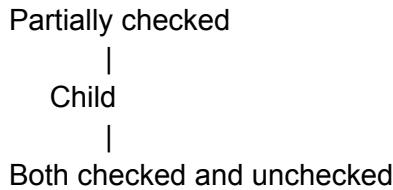
Unchecked Exception and checked Exception

`catch(-----)`

We use classNames

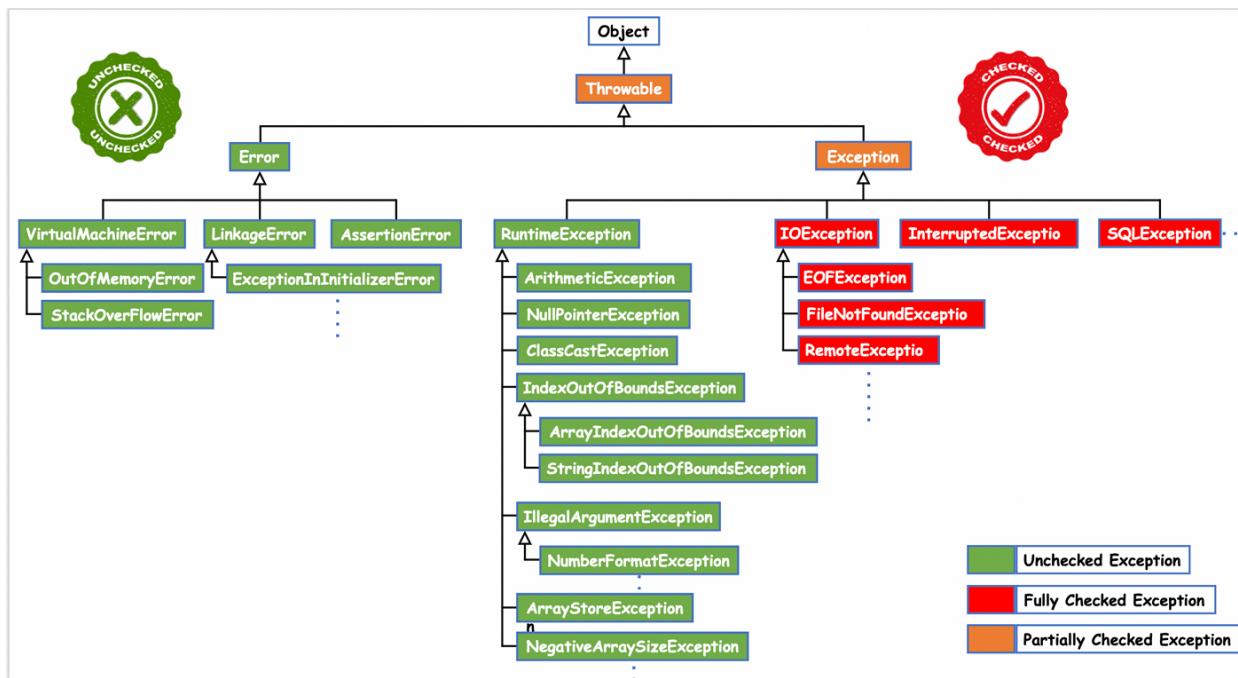
|
Inbuilt class names

|
No import statement so all are present in `java.lang`



InterruptedException, SQL exception, Null pointerException all are checked exceptions

If there is a parent class whose childs are checked and unchecked then that are called as partially checked



Parent of all class is OBJECT

RuntimeExceptions are unchecked exceptions

Compiler will not give any warnings

ioException InterruptedException SQLExceptions are checked exceptions

Compiler only gives the warnings for checked exceptions

Userdefined exception or custom exceptions

We can able to create our own exceptions

Try with resources

Differences between throw & throws

throw	throws
throw keyword is used to explicitly throw an exception to the JVM.	throws keyword is used to declare an exception to delegate exception handling responsibility to the caller.
throw keyword is followed by an instance of Throwable or a subclass of Throwable.	throws keyword is followed by exception class name.
throw keyword is used within the method body.	throws keyword is used with the method signature.
Multiple exceptions can't be thrown with a single throw clause.	Multiple exceptions can be declared with a single throws clause.
Used in rethrowing an exception.	Used in both rethrowing and ducking an exception.
<pre>try { } catch (Exception e) { throw e; }</pre>	<pre>void test() throws Exception { }</pre>

26) Code Snippets

Q.

```
interface Foo {}  
class Alpha implements Foo {}  
class Beta extends Alpha {}  
class Delta extends Beta {  
public static void main( String[] args ) {  
Beta x = new Beta();  
16. //insert code here 16  
}  
}
```

Which code, inserted at line 16, will cause a java.lang.ClassCastException?

- A. Alpha a = x;
- B. Foo f = (Delta)x;
- C. Foo f = (Alpha)x;
- D. Beta b = (Beta)(Alpha)x;

Answer: B

Foo()

```
|  
| implements  
|  
Alpa(C)  
|  
| extends  
|  
Beta(C) =====> x -> Foo f = (Delta)x; // invalid becoz the collecting type is of  
Foo( which is not related)
```

```
|  
| extends  
|  
Delta(C)
```

Q>

Given:

```
public class Batman {  
    int squares = 81;  
    public static void main(String[] args) {  
        new Batman().go();  
    }  
    void go() {  
        incr(++squares);  
        System.out.println(squares);  
    }  
    void incr(int squares) { squares += 10; }  
}
```

What is the result?

- A. 81
- B. 82
- C. 91
- D. 92
- E. Compilation fails.
- F. An exception is thrown at runtime.

Answer: B

Given:

1. public class Pass {
2. public static void main(String [] args) {
3.
int x = 5;
4.
5.
6.
7. }
- 8.

```
9.  
10.  
11.  
12.}  
Pass p = new Pass();  
p.doStuff(x);  
System.out.print(" main x = " + x);  
void doStuff(int x) {  
    System.out.print(" doStuff x = " + x++);  
}
```

What is the result?

- A. Compilation fails.
- B. An exception is thrown at runtime.
- C. doStuff x = 6 main x = 6
- D. doStuff x = 5 main x = 5
- E. doStuff x = 5 main x = 6
- F. doStuff x = 6 main x = 5

Answer: D

Q>

Given:

```
String[] elements = { "for", "tea", "too" };  
String first = (elements.length > 0) ? elements[0] : null;
```

What is the result?

- A. Compilation fails.
- B. An exception is thrown at runtime.
- C. The variable first is set to null.
- D. The variable first is set to elements[0].

Answer: D

Note:

```
int[] data = {10,20,30};  
System.out.println(data.length);  
String[] names = {"Navin","Haider","Nitin"};  
System.out.println(names.length);  
System.out.println(names[0].length());
```

Arrays => To find the length of the array we use length property or variable or field

String => To find the no of characters present in String we use length().

Q> Given:

```
10. public class SuperCalc {  
11.  
protected static int multiply(int a, int b) { return a * b;}  
12. }
```

and:

```
20. public class SubCalc extends SuperCalc{
```

21.
public static int multiply(int a, int b) {
22.
int c = super.multiply(a, b); // super => it always refers to
object creation process.
23.
24.
25. }
and:
return c;
}
30. SubCalc sc = new SubCalc ();
31. System.out.println(sc.multiply(3,4));
32. System.out.println(SubCalc.multiply(2,2));

What is the result?

- A. 12,4
- B. The code runs with no output.
- C. An exception is thrown at runtime.
- D. Compilation fails because of an error in line 21.
- E. Compilation fails because of an error in line 22.
- F. Compilation fails because of an error in line 31.

Answer: E

Question

```
class Foo {  
    public int a = 3; //instance variable  
    public void addFive() { a += 5; System.out.print("f "); }  
}  
class Bar extends Foo {  
    public int a = 8;//instance variable  
    public void addFive() { this.a += 5; System.out.print("b " ); }//overriding  
}
```

Invoked with:

```
Foo f = new Bar();//loose coupling  
f.addFive();//call will be decided by jvm based on runtime object becoz it is  
overriding method  
System.out.println(f.a);//since a is present in both parent and child compiler only  
will bind based on the type
```

What is the result?

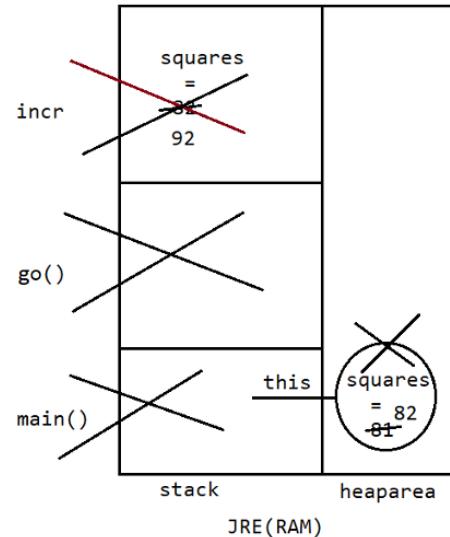
- A. b 3
- B. b 8
- C. b 13
- D. f 3
- E. f 8
- F. f 13

G. Compilation fails.

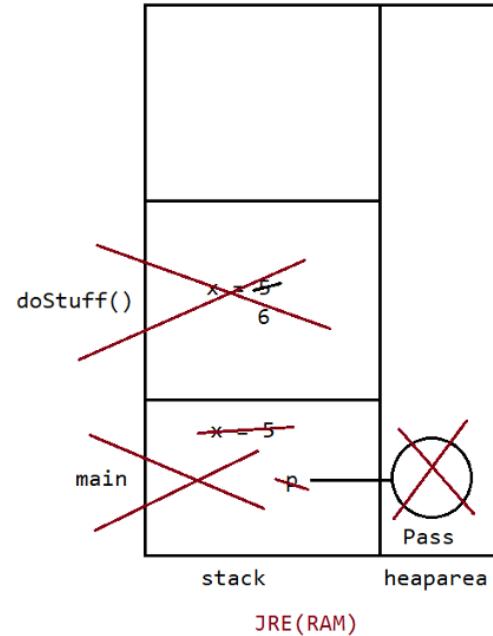
H. An exception is thrown at runtime.

Answer: A

```
public class Batman {  
    int squares = 81; // instance variable  
  
    public static void main(String[] args) {  
        new Batman().go();  
    }  
  
    void go() {  
        incr(++squares);  
        System.out.println(squares); // 82  
    }  
  
    void incr(int squares) { squares += 10; }  
}
```



```
1. public class Pass {  
2. public static void main(String [] args) {  
3.     int x = 5;  
4.     Pass p = new Pass();  
5.     p.doStuff(x);  
6.     System.out.print(" main x = " + x);  
7. } // main x = 5  
8.  
9. void doStuff(int x) {  
10.    System.out.print(" doStuff x = " + x++);  
11. } // doStuff x = 5  
12. }
```



51)29-11-22

(exception handling 3)

Custom Exception |
Valid and invalid ways (try-catch-finally)

Nesting try-catch

Control

(next day class)

Try with resources

Packages

(later day)

Threads|Multithreading

Collections

```
class Launch
{
    public static void main(String[] args)  JVM
    {
        try
        {
            System.out.println("Inside outer try");
            try
            {
                System.out.println("Inside inner try");
                System.out.println(10/0);
            }
            catch(ArithmeticException e)
            {
                System.out.println("Inside inner catch");
            }
            System.out.println("Outside inner try-catch");
        }
        catch(Exception e)
        {
            System.out.println("Inside outer catch");
        }
        finally
        {
            System.out.println("Inside outer finally");
        }
    }
}
```

OUTPUT

```
Inside outer try
Inside inner try
Inside inner catch
Outside inner try-catch
Inside outer finally
```

```

class Launch
{
    public static void main(String[] args) JVM
    {
        try
        {
            System.out.println("Inside outer try");
            try
            {
                System.out.println("Inside inner try");
                System.out.println(10/0);
            }
            catch(NullPointerException e)
            {
                System.out.println("Inside inner catch");
            }
            System.out.println("Outside inner try-catch");
        }
        catch(Exception e)
        {
            System.out.println("Inside outer catch");
        }
        finally
        {
            System.out.println("Inside outer finally");
        }
    }
}

```

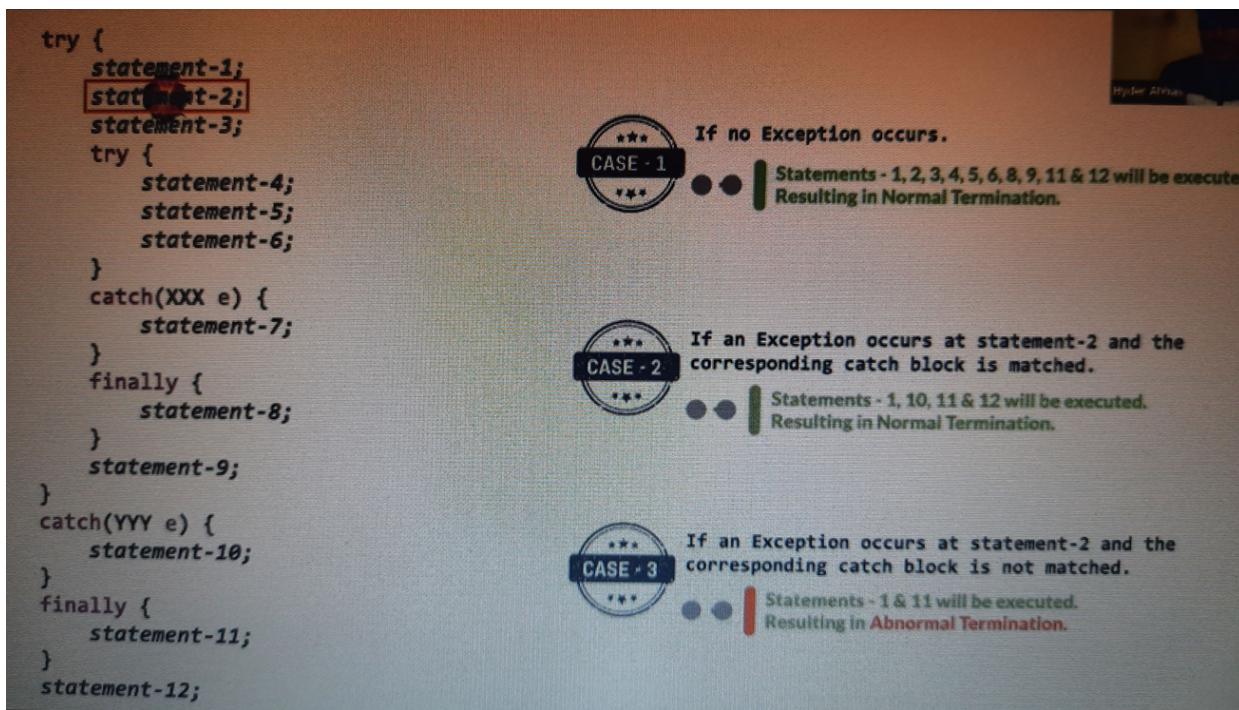
OUTPUT

```

Inside outer try
Inside inner try
Inside outer catch
Inside outer finally

```

Any number of inside nesting is possible even inside try - catch - finally



```

try {
    statement-1;
    statement-2;
    statement-3;
    try {
        statement-4;
        statement-5; 
        statement-6;
    }
    catch(XXX e) {
        statement-7;
    }
    finally {
        statement-8;
    }
    statement-9;
}
catch(YYY e) {
    statement-10;
}
finally {
    statement-11;
}
statement-12;

```



If an Exception occurs at statement-5 and the corresponding inner catch block is matched.

Statements - 1, 2, 3, 4, 7, 8, 9, 11 & 12 will be executed.
Resulting in Normal Termination.



If an Exception occurs at statement-5 and the corresponding inner catch block is not matched, but outer catch block is matched.

Statements - 1, 2, 3, 4, 8, 10, 11 & 12 will be executed.
Resulting in Normal Termination.



If an Exception occurs at statement-5 and both inner and outer catch blocks are not matched.

Statements - 1, 2, 3, 4, 8 & 11 will be executed.
Resulting in Abnormal Termination.

```

try {
    statement-1;
    statement-2;
    statement-3;
    try {
        statement-4;
        statement-5;
        statement-6;
    }
    catch(XXX e) {
        statement-7;
    }
    finally {
        statement-8; 
    }
    statement-9;
}
catch(YYY e) {
    statement-10;
}
finally {
    statement-11;
}
statement-12;

```



If an Exception occurs at Statement-8 and the corresponding catch block is matched.

Statements - 1, 2, 3, X, X, X (4, 5, 6, 7 optional), 10, 11, 12 will be Executed with Normal termination.



If an Exception occurs at Statement-8 and the corresponding catch block is not matched.

Statements - 1, 2, 3, X, X, X (4, 5, 6, 7 optional), 11 will be Executed with Abnormal termination.

CASE - 1	CASE - 2	CASE - 3	CASE - 4	CASE - 5
Only try <pre>try {}</pre>	Only catch <pre>catch(XXX e) {}</pre>	Only finally <pre>finally {}</pre>	try - catch <pre>try { } catch(XXX e) {}</pre>	Reversed <pre>catch(XXX e) { } try {}</pre>

CASE - 6	CASE - 7	CASE - 8	CASE - 9	CASE - 10	CASE - 11
Multiple try <pre>try { } try { } catch(XXX e) {}</pre>	Multiple try <pre>try { } catch(XXX e) {}</pre>	Multiple try - catch <pre>try { } catch(XXX e) {}</pre>	Multiple catch <pre>try { } catch(XXX e) {}</pre>	Multiple catch <pre>try { } catch(XXX e) {}</pre>	Multiple catch <pre>try { } catch(XXX YYY e) {}</pre>

CASE - 12

try-catch-finally

```
try  
{  
}  
catch(XXX e)  
{  
}  
finally  
{  
}
```



CASE - 13

try - finally

```
try  
{  
}  
} finally  
{  
}
```



CASE - 14

catch - finally

```
catch(XXX e)  
{  
}  
} finally  
{  
}
```



CASE

Unordered

```
try  
{  
}  
} finally  
{  
}  
catch(XXX e)  
{  
}
```



CASE - 24

Nested try-catch-finally



```
try
{
    try
    {
    }
    catch(XXX e)
    {
    }
    finally
    {
    }
}
catch(XXX e)
{
    try
    {
    }
    catch(XXX e)
    {
    }
    finally
    {
    }
}
finally
{
    try
    {
    }
    catch(XXX e)
    {
    }
    finally
    {
    }
}
```

String a = null

When we try to do upper and lower case methods for the above
It leads to nullPointerException.

Synchronous VS Asynchronous Exceptions
(54:55)

Synchronous Exceptions

- Occurs at a specific program statement.

```
class Launch
{
    public static void main(String[] args)
    {
        String str = null;
        str.toUpperCase(); // 7
    }
}
```

OUTPUT
java.lang.NullPointerException
at Launch.main(Launch.java:6)

```
class Launch
{
    public static void main(String[] args)
    {
        int[] a = new int[5];
        a[5] = 10; // 8
    }
}
```

OUTPUT
java.lang.ArrayIndexOutOfBoundsException:
at Launch.main(Launch.java:6)

Synchronous Exception means = occurs at a specific program stmt

Eg: ArrayIndexOutOfBoundsException

Asynchronous Exceptions

- Occurs anywhere in the program.

```
class Launch
{
    public static void main(String[] args)
    {
        Scanner scan = new Scanner(System.in);
        System.out.println("Enter your name:");
        String name = scan.next();
        System.out.println("Enter your grades:");
        String grade = scan.next();
    }
}
```

OUTPUT

Enter your name:
Sachin
Enter your grades:
CTRL + C Keyboard Interrupt
Exception in thread "main" Terminate batch job (Y/N)?

Examples:-

Keyboard interrupts,

Asynchronous Exception means = Occurs anywhere in the program

Eg: Keyboard Interrupt

RUNTIME ERROR

Which occurs during the runtime

Eg: stackOverflow

Demonstration of Runtime Errors

RECURSION

```
class Launch
{
    public static void alpha()
    {
        alpha();
    }

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

OUTPUT

```
Exception in thread "main" java.lang.StackOverflowError
at Launch.alpha(Launch.java:5)
at Launch.alpha(Launch.java:5)
at Launch.alpha(Launch.java:5)
at Launch.alpha(Launch.java:5)
at Launch.alpha(Launch.java:5)
.
.
```

AR for
alpha()
AR for
alpha()

alpha()	_____
main()	_____

Run-Time Stack

RECURSION

Demonstration of Runtime Errors

The diagram illustrates a run-time stack with multiple frames. At the top is a frame for `beta()`. Below it are several frames for `alpha()`, each preceded by "AR for". The stack continues with frames for `beta()` and then ends with a frame for `main()`.

Run-Time Stack

```

class Launch
{
    public static void alpha()
    {
        beta();
    }
    public static void beta()
    {
        alpha();
    }
    public static void main(String[] args)
    {
        alpha(); ^*
    }
}

```

OUTPUT

```

Exception in thread "main" java.lang.StackOverflowError
at Launch.beta(Launch.java:9)
at Launch.alpha(Launch.java:5)
at Launch.beta(Launch.java:9)
at Launch.alpha(Launch.java:5)
...

```

Demonstration of Runtime Errors

The diagram shows a code snippet where a large integer array is created. A callout box highlights the creation of a 1GB array and states it is impossible to do so contiguously on the heap.

Impossible to **1000000000 * 4 = 4 GB**
contiguously on the heap

OUTPUT

```

Exception in thread "main" java.lang.OutOfMemoryError: Java heap space
at Launch.main(Launch.java:5)

```

Array uses the contiguous memory locations.
You can't manage errors like the exceptions

User Defined Exception

```
import java.util.Scanner;
```

```
class InvalidCustomerException extends Exception
{
    public InvalidCustomerException(String msg)
    {
        super(msg);
    }
}

class Atm
{
    int userid=1212;
    int password=1111;
    int pw;
    int uid;

    public void input()
    {
        Scanner scan=new Scanner(System.in);

        System.out.println("Kindly enter the user id");
        uid=scan.nextInt();

        System.out.println("Kindly enter the password");
        pw=scan.nextInt();
    }

    public void verify() throws InvalidCustomerException
    {
        if((userid== uid) && (password == pw) )
        {
            System.out.println("Take your cash");
        }
        else
        {
//System.out.println("wrong data");
            InvalidCustomerException ice=new InvalidCustomerException("Are
you sure? try again bcz wrong input");
            //System.out.println(ice);
            System.out.println(ice.getMessage());
            throw ice;
        }
    }
}
```

```
        }
    }
}

class Bank
{
    public void initiate()
    {
        Atm a=new Atm();
        try
        {
            a.input();
            a.verify();

        }
        catch (InvalidCustomerException e1)
        {
            try
            {
                a.input();
                a.verify();

            }
            catch (InvalidCustomerException e2)
            {
                try
                {
                    a.input();
                    a.verify();

                }
                catch (InvalidCustomerException e3)
                {

                    System.out.println("Oh Choor dude we caught you card
is blocked! !");
                    System.exit(0);
                }
            }
        }
    }
}
```

```

        }
    }

}

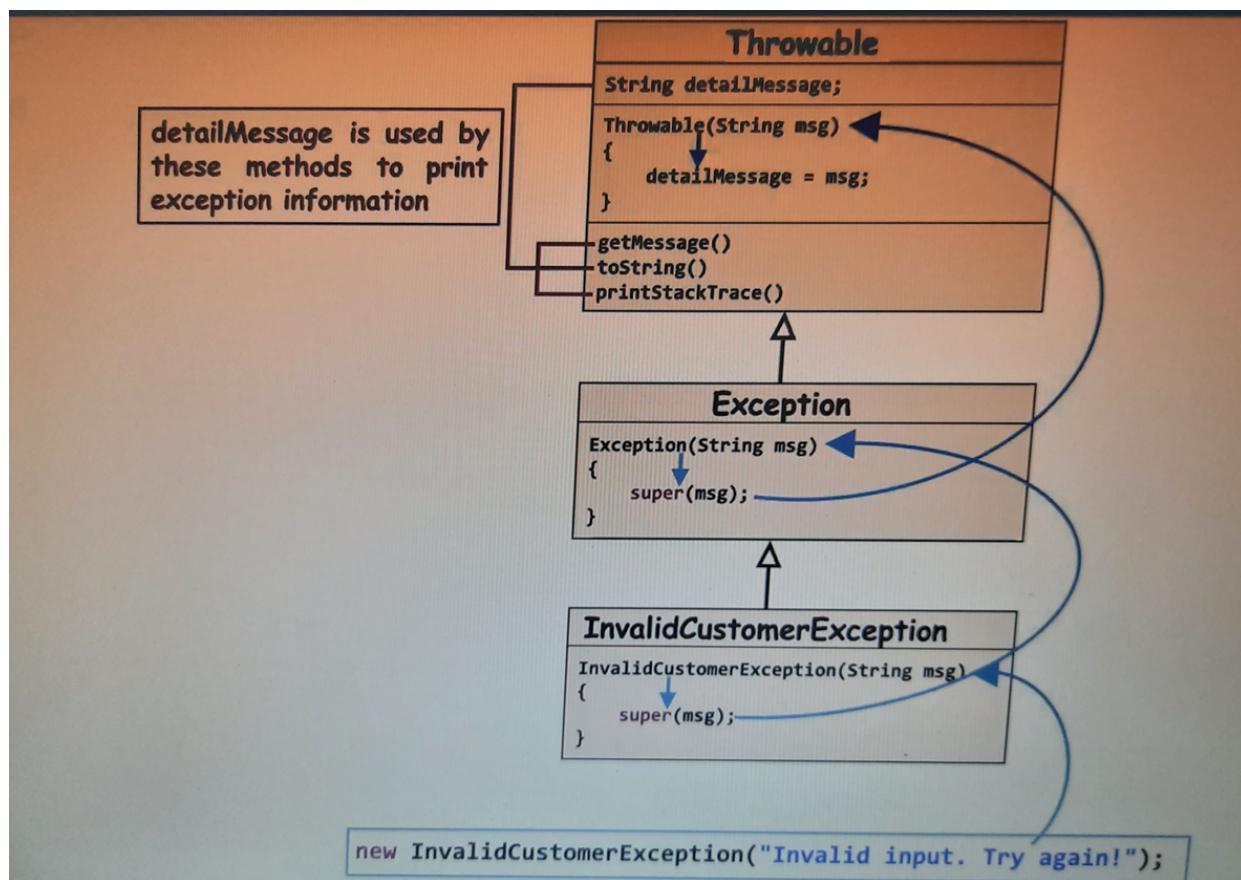
public class LaunchCE {

    public static void main(String[] args) {

        Bank b=new Bank();
        b.initiate();

    }
}

```



RTO license

```

import java.util.Scanner;//5
class UnderAgeException extends Exception//13

```

```
{  
    public UnderAgeException(String message)//15  
    {  
        super(message); //16  
    }  
}  
  
class OverAgeException extends Exception//14  
{  
    public OverAgeException(String message)//17  
    {  
        super(message); //17  
    }  
  
}  
  
}  
  
class Applicant//1  
{  
    int age;//2  
    public void input()//3  
    {  
        Scanner scan=new Scanner(System.in);//4  
        System.out.println("Please enter your age");//6  
        age=scan.nextInt();//7  
    }  
    //void verify() throws Exception  
    void verify()/*8*/ throws UnderAgeException,OverAgeException //26 //30  
    {  
        if(age <18)//9  
        {  
            UnderAgeException uae=new UnderAgeException("Ohh dude? calm  
down your time will come");//23  
            System.out.println(uae.getMessage());//24  
            throw uae;//25  
        }  
        else if(age > 60)//10  
        {  
            OverAgeException oae=new OverAgeException("your time is near  
calm down and pray");//27  
            System.out.println(oae.getMessage());//28  
        }  
    }  
}
```

```
        throw oae;//29
    }

else//11
{
    System.out.println("You're eligible");//12
}
}

class Rto //18
{
    public void initiate()//19
    {
        Applicant a=new Applicant(); //20
        Try //31
        {
            a.input(); //21
            a.verify(); //22
        }
        catch(UnderAgeException| OverAgeException e) //32
        {
            try//33
            {
                a.input();
                a.verify();
            }
            catch(UnderAgeException |OverAgeException e1)//33
            {
                System.out.println("Don't ever try again read rules");//34
                System.exit(0);//35
            }
        }
    }
    // try
    //{
    //     a.input();
    //     a.verify();
    // }
}
```

```

//      catch(OverAgeException e)
//      {
//          try
//          {
//              a.input();
//              a.verify();
//          }
//          catch(OverAgeException e1)
//          {
//              System.out.println("Don't ever try again read rules");
//              System.exit(0);
//          }
//      }
}

public class LaunchRto {

    public static void main(String[] args) {
        Rto r=new Rto(); //36
        r.initiate(); //37

    }
}

```

We won't work with Applets and Graphics that are deprecated api

27)Code Snippets

(02:34:11)

Q>

```

class Thingy { Meter m = new Meter(); }
class Component { void go() { System.out.print("c"); } }
class Meter extends Component { void go() { System.out.print("m"); } }
class DeluxeThingy extends Thingy {
    //meter object would come and meter would extend component so navigation is
possible
    public static void main(String[] args) {

```

```
DeluxeThingy dt = new DeluxeThingy();
dt.m.go();
Thingy t = new DeluxeThingy();
t.m.go();
}
}
```

Which two are true? (Choose two.)

- A. The output is mm.
- B. The output is mc.
- C. Component is-a Meter.
- D. Component has-a Meter.
- E. DeluxeThingy is-a Component.
- F. DeluxeThingy has-a Component.

relationship b/w 2 classes as inheritance => IS-A relation

relationship b/w 2 classes which are used for navigation => HAS-A relationship

Answer: A F

Q>

```
public class Test{
public static void main(String[] args){
boolean a =new Boolean(Boolean.valueOf(args[0]));//new Boolean(false)
System.out.println(a);//false
boolean b =new Boolean(args[1]);//new Boolean("null")
System.out.println(b);//false
}
}
```

And the commands are

javac Test.java

java Test 1 null

commandline arguments

=====

args[0] = "1"

args[1] = "null"

What is the result?

- A. 1 null
- B. true false
- C. false false
- D. true true
- E. Class cast Exception is thrown at the runtime

Answer: C

Q>

```
public class Test {
public static void doSum(int x, int y){
}
System.out.println("int sum is:: "+(x+y));
```

```
public static void doSum(Integer x, Integer y){  
    System.out.println("Integer sum is:: "+(x+y));  
}  
public static void doSum(double x, double y){  
    System.out.println("double sum is:: "+(x+y));  
}  
public static void doSum(float x, float y){  
    System.out.println("float sum is:: "+(x+y));  
}  
public static void main(String[] args) {  
    doSum(10,20);  
    doSum(10.0,20.0);  
}  
}
```

What is the result?

- A. int sum is :: 30
 float sum is :: 30.0
- B. int sum is :: 30
 double sum is :: 30.0
- C. Integer sum is :: 30
 double sum is :: 30.0
- D. Integer sum is:: 30
 float sum is :: 30.0

Answer: B

Q>

Given:

```
1. public class BuildStuff {  
2.  
3.  
4.  
5.  
6.  
7.  
8.  
9.  
10.  
11.  
12.}  
public static void main(String[] args) {  
    Boolean test = new Boolean(true);//test = true  
    Integer x = 343;  
    Integer y = new BuildStuff().go(test, x);// true,343  
    System.out.println(y);//49  
}
```

```
int go(Boolean b, int i) {  
if(b) return (i/7); // return 343/7 => 49  
return (i/49);  
}
```

What is the result?

- A. 7
- B. 49
- C. 343
- D. Compilation fails.
- E. An exception is thrown at runtime.

Answer: B

Q>

```
class Vehicle{  
int x;  
Vehicle(){  
}  
this(10);  
Vehicle(int x){  
this.x = x;  
}  
}  
class Car extends Vehicle{  
int y;  
Car(){  
super(); // line-n1  
this(20); // line-n2  
}  
Car(int y){  
this.y = y;  
}  
public String toString(){  
return super.x + " " + this.y;  
}  
}
```

```
public class Test {  
public static void main(String[] args) {  
Vehicle y= new Car();  
System.out.println(y);  
}  
}
```

Predict the answer

- A. 10:20
- B. 0:20
- C. Compilation fails at line n1

D. Compilation fails at line n2

Answer: D

Q>

```
public static void main(String[] args) {  
    Integer i = new Integer(1) + new Integer(2); //line-12  
    switch(i) { //line-13  
        case 3: System.out.println("three"); break;  
        default: System.out.println("other"); break; //line-15  
    }  
}
```

What is the result?

- A. three
- B. other
- C. An exception is thrown at runtime.
- D. Compilation fails because of an error on line 12.
- E. Compilation fails because of an error on line 13.
- F. Compilation fails because of an error on line 15.

Answer: A

Q>

Given:

```
public class Foo {  
    static int[] a;  
    static { a[0]=2; }  
}
```

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

Which exception or error will be thrown when a programmer attempts to run this code?

- A. java.lang.StackOverflowError
- B. java.lang.IllegalStateException
- C. java.lang.ExceptionInInitializerError
- D. java.lang.ArrayIndexOutOfBoundsException

static{

//if some exception occurs during initialization then it is called as

"ExceptionInInitializerError"

}

Q>

Given:

10. interface Jumper { public void jump(); }

...

20. class Animal {

...

30. class Dog extends Animal {//Dog IS-A Animal , Dog Has-A tail

31.

Tail tail;

```
32. }
...
40. class Beagle extends Dog implements Jumper{//Beagle is a Dog,Beagle is a
Animal,Beagle has-a tail, Beagle is-A jumper
41.
public void jump() {}
42. }
...
50. class Cat implements Jumper{//Cat is-a jumper
51.
public void jump() {}
52. }
Which three are true? (Choose three.)
A. Cat is-a Animal
B. Cat is-a Jumper
C. Dog is-a Animal
D. Dog is-a Jumper
E. Cat has-a Animal
F. Beagle has-a Tail
G. Beagle has-a Jumper
Answer: BCF
QUESTION
Given:
1. class X {
2.
X() { System.out.print(1); }
3.
4.
5.
6. }
X(int x) {
this(); System.out.print(2);
}
7. public class Y extends X {
8.
Y() { super(6); System.out.print(3); }
9.
10.
11.
Y(int y) {
this(); System.out.println(4);
}
12. public static void main(String[] a) { new Y(5); }
13.}
```

What is the result?

- A. 13
- B. 134
- C. 1234
- D. 2134
- E. 2143
- F. 4321

Answer: C(1234)

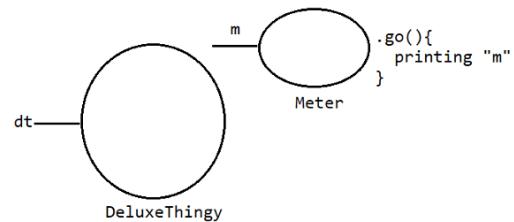
```
class Thingy { Meter m = new Meter(); }

class Component { void go() { System.out.print("c"); } }

class Meter extends Component {
    void go() { System.out.print("m"); } //overriding method
}

class DeluxeThingy extends Thingy {
    public static void main(String[] args) {
        DeluxeThingy dt = new DeluxeThingy();
        dt.m.go();
        Thingy t = new DeluxeThingy();
        t.m.go();
    }
}

output: "mm"(because of runtime object the method call will happen)
```



```
class Car extends Vehicle{
    int y;
    Car(){
        super(); //line-n1
        this(20); //line-n2
    }
    Car(int y){
        this.y= y;
    }
    public String toString(){
        return super.x + " " + super.y;
    }
}

public class Test {
    public static void main(String[] args) {
        Vehicle y= new Car();
        System.out.println(y);
    }
}
```

52)30-11-22 dbt

53)30-11-22

(exception handling 4)

Exception Handling

=====

1.try with resource

2.try with multi-catch block

3.Rules of overriding associated with Exception

1.Instanceof vs isInstanceOf(Object obj)

2.How to create a user defined package and in real time project how it is used?

1.7 version Enhancements

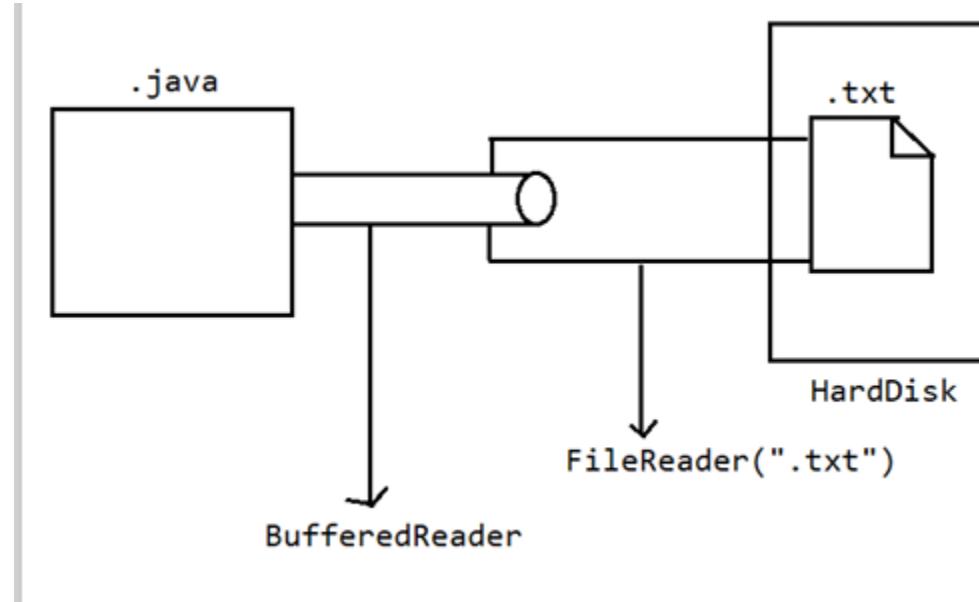
=====

1. try with resource

2. try with multicatch block

until jdk1.6, it is compulsorily required to write finally block to close all the resources which are open as a part of try block.

(Resource means an Object which you have used inside code to perform operation)



```
eg:: BufferedReader br=null; //Resource
try{
    br=new BufferedReader(new FileReader("abc.txt"));
}catch(IOException ie){
    ie.printStackTrace();
}finally{
```

```

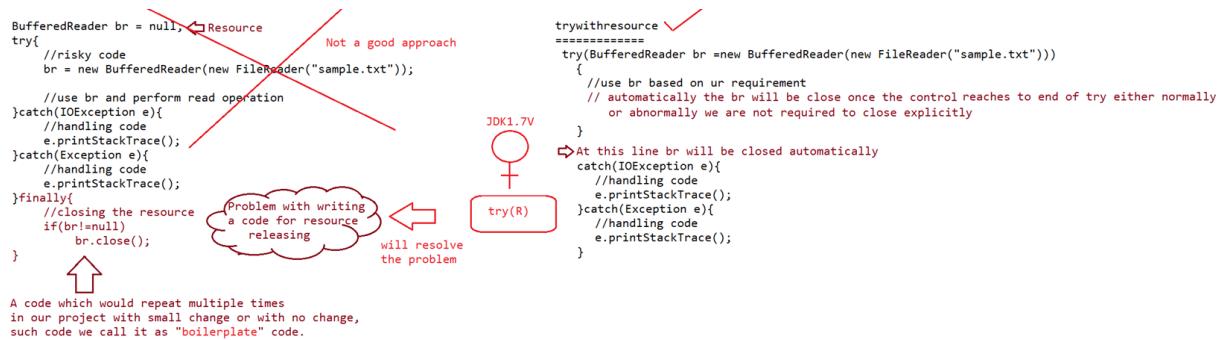
try{
    if(br!=null){
        br.close();
    }
} catch(IOException ie){
    ie.printStackTrace();
}
}

```

```

>>javap java.lang.AutoClosable
>>javap java.sql.Connection
>>javap java.lang.Statement
>>javap java.util.Scanner

```



Problems in the approach

1. Compulsorily the programmer is required to close all opened resources which increases the complexity of the program
2. Compulsorily we should write finally block explicitly, which increases the length of the code and reviews readability.

To Overcome this problem SUN MS introduced try with resources in "1.7" version of jdk.

try with resources

In this approach, the resources which are opened as a part of try block will be closed automatically once the control reaches to the end of try block normally or abnormally, so it is not required to close explicitly so the complexity of the program would be reduced.

It is not required to write finally block explicitly, so length of the code would be reduced and readability is improved.

```

try(BufferedReader br=new BufferedReader(new FileReader("abc.txt"))){
    //use br and perform the necessary operation
    //once the control reaches the end of try automatically br will be closed
}

```

```
}catch(IOException ie){  
    //handling code  
}
```

Rules of using try with resource

=====

1. we can declare any no of resources, but all these resources should be seperated with ;

eg#1.

```
try(R1;R2;R3;){  
    //use the resources  
}
```

2. All resources are said to be AutoCloseable resources iff the class implements an interface called "java.lang.AutoCloseable"

either directly or indirectly

eg:: java.io package classes, java.sql.package classes

```
public interface java.lang.AutoCloseable {  
    public abstract void close() throws java.lang.Exception;  
}
```

Note: which ever class has implemented this interface those classes objects are refered as "resources".

3. All resource reference by default are treated as implicitly final and hence we can't perform reassignment with in try block.

```
try(BufferedReader br=new BufferedReader(new FileWriter("abc.txt")){  
    br=new BufferedReader(new FileWriter("abc.txt"));  
}  
output::CE: can't reassign a value
```

4. until 1.6 version try should compulsorily be followed by either catch or finally, but from 1.7 version we can take only take try with resources without cath or finally.

```
try(R){  
//valid  
}
```

5. Advantage of try with resources concept is finally block will become dummy because we are not required to close resources explicitly.

6. try with resource nesting is also possible.

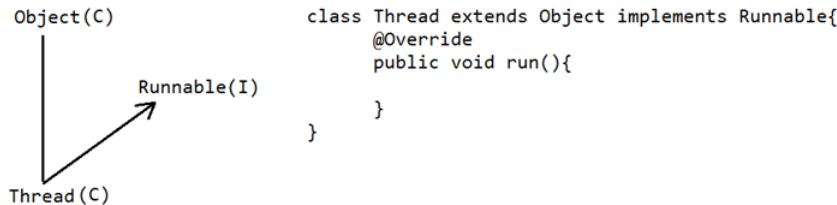
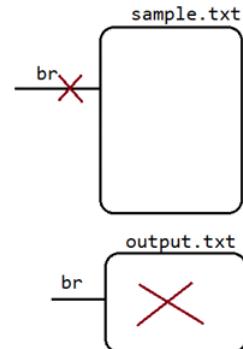
```
try(R1){  
    try(R2){  
        try(R3){  
        }  
    }  
}
```

```
}
```

Try with resource is only to avoid the final block

```
    br is final variable
    ↓
try(BufferedReader br =new BufferedReader(new FileReader("sample.txt"))){
    br =new BufferedReader(new FileReader("output.txt")); CE: can't reassign
}
//br.close() will execute automatically and resource will be closed
catch(Exception e){
    e.printStackTrace();
}

    1. both should be closed
    2. output.txt should be closed.
    3. confusion whom to close
    4. None of the above
```



MultiCatchBlock

=====

Till jdk1.6, even though we have multiple exception having same handling code we have to write a separate catch block for every exceptions, it increases the length of the code and reduces readability.

logic

====

```
try{
    ....
    ....
    ....
    ....
}catch(ArithmaticException ae){
ae.printStackTrace();
}catch(NullPointerException ne){
ne.printStackTrace();
}catch(ClassCastException ce){
System.out.println(ce.getMessage());
}catch(IOException ie){
System.out.println(ie.getMessage());
```

```
}
```

To overcome this problem SUNMS has introduced "Multi catch block" concept in 1.7 version

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

```
}catch(ArithmeticException |NullPointerException e){  
e.printStackTrace();  
}catch(ClassCastException |IOException e){  
e.printStackTrace();
```

In multicatch block, there should not be any relation b/w exception types(either child to parent or parent to child or same type) it would result in compile time error.

```
eg:: try{  
    }catch( ArithmeticException | Exception e){  
e.printStackTrace();  
}
```

Output: CompileTime Error

throw => handle the exception using catch block and throw it back the exception object to the caller.

throws => method signature and commonly used if the exception is "CheckedException".

CheckedException => compiler will check for the handling code only then compilation is successful.

eg: IOException, SQLException, are all checked exceptions.

UnCheckedException => compiler will not check for the handling code, but jvm will come into picture

and possibility of "successful" or "abnormal" Termination.

"UnCheckedException".

eg: RuntimeException and its child classes
Error and its child classes are all

Rules of Overriding when exception is involved

While Overriding if the child class method throws any checked exception compulsorily the parent class method should throw the same checked exception or its parent otherwise we will get Compile Time Error.
There are no restrictions on UncheckedException.
eg#1.

```
class Parent{
public void methodOne();
}
class Child extends Parent{
public void methodOne() throws Exception{}
}
error: methodOne() in Child cannot override methodOne() in Parent
    public void methodOne() throws Exception{}
        overridden method does not throw Exception
```

Rules w.r.t Overriding

=====

```
parent: public void methodOne() throws Exception{}
child : public void methodOne()
output: valid
```

```
parent: public void methodOne(){}
child : public void methodOne() throws Exception{}
output: invalid
```

```
parent: public void methodOne()throws Exception{}
child : public void methodOne()throws Exception{}
output: valid
```

```
parent: public void methodOne()throws IOException{}
child : public void methodOne()throws IOException{}
output: valid
```

```
parent: public void methodOne()throws IOException{}
child : public void methodOne()throws FileNotFoundException,EOFException{}
output: valid
```

```
parent: public void methodOne()throws IOException{}
child : public void methodOne()throws FileNotFoundException,InterruptedException{}
output: invalid
```

```
parent: public void methodOne()throws IOException{}
child : public void methodOne()throws FileNotFoundException,ArithmaticException{}
output: valid
```

```
parent: public void methodOne()
child : public void methodOne()throws
ArithmaticException,NullPointerException,RuntimeException{}
output: valid
```

```
parent: public void methodOne()throws IOException{}  
child : public void methodOne()throws Exception{}  
output: invalid
```

```
parent: public void methodOne()throws Throwable{}  
child : public void methodOne()throws IOException{}  
output: valid
```

instanceof

=====

1. We can use the instanceof operator to check whether the given object is a particular type or not.

r instanceof X

r => reference

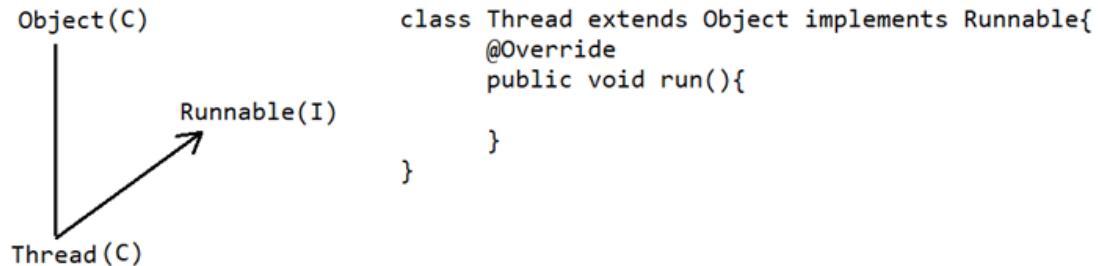
X => class/interfaceName

eg:

```
ArrayList al =new ArrayList(); //inbuilt object where we can keep any type of  
other objects  
al.add(new Student()); //0th position  
al.add(new Cricketer()); //1st position  
al.add(new Customer()); //2nd position  
Object o=l.get(0); // l is an arraylist object  
if(o instanceof Student) { //instanceof returns the boolean  
Student s=(Student)o;  
//perform student specific operation  
}  
elseif(o instanceof Customer) {  
Customer c=(Customer)o;  
//perform Customer specific operations  
}
```

eg#2.

Ex :



```
//Object t = new Thread();
//Thread t = new Object();
Thread t = new Thread( );
System.out.println(t instanceof Thread);//true
System.out.println(t instanceof Object);//true
System.out.println(t instanceof Runnable); //true
public class Thread extends Object implements Runnable {
}
```

check

```
sop(t instanceof String) //CE
//instanceof means JVM role is more
```

Is any relationship between thread and string

check

```
(null instanceof Object)
```

=> To use instanceof operator compulsory there should be some relation between argument types

(either child to parent Or parent to child Or same type) Otherwise we will get compile time error saying incompatible types.

```
eg: String s= new String("sachin");
    System.out.println(s instanceof Thread);//CE
```

```
Thread t=new Thread( );
System.out.println(t instanceof String);//CE
```

=> Whenever we are checking the parent object is child type or not by using instanceof operator that we get false.

```
Object o=new Object( );
System.out.println(o instanceof String ); //false
```

```
Object o=new String("ashok");
System.out.println(o instanceof String); //true
```

=> For any class or interface X null instanceof X is always returns false

```
System.out.println(null instanceof X); //false
```

```
public class Test {
    public static void main(String[] args) {
        Object t = new Thread();
        System.out.println(t instanceof Object);//true
        System.out.println(t instanceof Thread);//true
        System.out.println(t instanceof Runnable);//true
        System.out.println(t instanceof String);//false
        System.out.println(null instanceof Object);//false
    }
}
```

```
}
```

```
isInstance()
```

```
=====
```

```
Difference between instanceof and isInstance( ) :
```

```
instanceof
```

```
=====
```

```
instanceof an operator which can be used to check whether the given object is  
particular type or not We know at the type  
at the beginning it is available.
```

```
eg: String s = new String("sachin");  
System.out.println(s instanceof Object );//true  
//If we know the type at the beginning only.
```

```
isInstance( )
```

```
=====
```

```
isInstance( ) is a method , present in class Class , we can use isInstance() method to checked  
whether the given object is  
particular type or not We don't know at the type at beginning it is available Dynamically at  
Runtime.
```

```
class Test {  
    public static void main(String[] args) {  
        Test t = new Test();  
        System.out.println(Class.forName(args[0]).isInstance(t));///arg[0] --- //We don't know the type at  
beginning  
    }  
}  
java Test Test //true  
java Test String //false  
java Test Object //true
```

```

class Inner {
    private int x;
    public void setX( int x ){ this.x = x; }
    public int getX(){ return x; }
}

```

```

Outer o = new Outer();
Inner i = new Inner();
int n = 10;
i.setX(n);
o.setY(i);

answer: produce the output 100?
// insert code here 29
System.out.println(o.getY().getX());

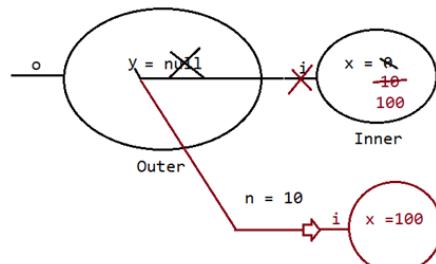
```

- A. n = 100;~~X~~
- B. i.setX(100);~~✓~~
- C. o.getY().setX(100);~~✓~~
- D. i = new Inner(); i.setX(100);~~X~~
- E. o.setY(i); i = new Inner(); i.setX(100);~~X~~
- F. i = new Inner(); i.setX(100); o.setY(i);~~✓~~

```

class Outer {
    private Inner y;
    public void setY( Inner y ){ this.y = y; }
    public Inner getY() { return y; }
}

```



```

public class Hello {
    String title;
    int value;
    public Hello() {
        title += " World";
    }
    public Hello(int value) {
        this.value = value;
        title = "Hello";
        Hello(); // constructor can't be called explicitly
        it results in "CE".
    }
}
Hello c = new Hello(5);
System.out.println(c.title);

```

28)Code Snippets

(2:55:00)

Q>

Given:

```

abstract public class Employee {
    protected abstract double getSalesAmount();
    public double getCommision() {
        return getSalesAmount() * 0.15;
    }
}
class Sales extends Employee {
    17. // insert method here
}

```

Which two methods, inserted independently at line 17, correctly complete the Sales class? (Choose two.)

- A. double getSalesAmount() { return 1230.45; }
- B. public double getSalesAmount() { return 1230.45; }
- C. private double getSalesAmount() { return 1230.45; }
- D. protected double getSalesAmount() { return 1230.45; }

Answer: BD

Q>

Given this code from Class B:

```
25. A a1 = new A();  
26. A a2 = new A();  
27. A a3 = new A();  
28. System.out.println(A.getInstanceCount());
```

What is the result?

```
1. public class A{  
2.  
3. private int counter = 0;  
4.  
5. public static int getInstanceCount() {  
6.  
return counter;  
7. }  
8.  
9.  
10.  
11. }  
12.  
13.}
```

- ```
public A() {
counter++;
A. Compilation of class A fails.
B. Line 28 prints the value 3 to System.out.
C. Line 28 prints the value 1 to System.out.
D. A runtime error occurs when line 25 executes.
E. Compilation fails because of an error on line 28.
```

Answer: A

QUESTION

Given:

```
1. public class A {
2.
public void doit() {
3.
4.
5.
6.
7.
}
public String doit() {
}
return "a";
```

```
public double doit(int x) {
}
return 1.0;
8.
9.
10.}
```

What is the result?

- A. An exception is thrown at runtime.
- B. Compilation fails because of an error in line 7.
- C. Compilation fails because of an error in line 4.
- D. Compilation succeeds and no runtime errors with class A occur.

Answer: C

#### QUESTION

Which three code fragments, added individually at line 29, produce the output 100?

(Choose three.)

```
class Inner {
private int x;
public void setX(int x){ this.x = x; }
public int getX(){ return x;}
}
class Outer {
private Inner y;
public void setY(Inner y){ this.y = y; }
public Inner getY() { return y; }
}
public class Gamma {
public static void main(String[] args) {
Outer o = new Outer();
Inner i = new Inner();
int n = 10;
i.setX(n);
o.setY(i);
// insert code here 29
System.out.println(o.getY().getX());
}
}
```

- A. n = 100;
- B. i.setX( 100 );
- C. o.getY().setX( 100 );
- D. i = new Inner(); i.setX( 100 );
- E. o.setY( i ); i = new Inner(); i.setX( 100 );
- F. i = new Inner(); i.setX( 100 ); o.setY( i );

Answer: BCF

Q>

```
public class Base {
 public static final String FOO = "foo";
 public static void main(String[] args) {
 Base b = new Base();
 Sub s = new Sub();
 }
}
System.out.print(Base.FOO);//foo
System.out.print(Sub.FOO);//bar
System.out.print(b.FOO);//foo
System.out.print(s.FOO);//bar
System.out.print(((Base) s).FOO);//foo
class Sub extends Base {
}
```

public static final String FOO = "bar";

What is the result?

- A. foofoofoofoofoo
- B. foobarfoobarbar
- C. foobarfoofoofoo
- D. foobarfoobarfoo
- E. barbarbarbarbar
- F. foofoofoobabar
- G. foofoofoobarfoo

Answer: D

Given:

```
1. class Mammal {
2. }
3.
4. class Raccoon extends Mammal {//Raccoon IS-A Mammal, Raccoon HAS-A mamal
5.
Mammal m = new Mammal();
6.
7.
8. class BabyRaccoon extends Mammal {//BabyRaccoon IS-A Mammal
9.
10.
```

Which four statements are true? (Choose four.)

- A. Raccoon is-a Mammal.
- B. Raccoon has-a Mammal.
- C. BabyRaccoon is-a Mammal.
- D. BabyRaccoon is-a Raccoon.
- E. BabyRaccoon has-a Mammal.
- F. BabyRaccoon is-a BabyRaccoon.

Answer: ABCF

Q>  
Given  
public class Hello {  
 String title;  
 int value;  
 public Hello() {  
 title += " World";  
 }  
 public Hello(int value) {  
 this.value = value;  
 title = "Hello";  
 Hello();  
 }  
}

and:

Hello c = new Hello(5);  
System.out.println(c.title);

What is the result?

- A. Hello
- B. Hello World
- C. Compilation fails.
- D. Hello World 5
- E. The code runs with no output.
- F. An exception is thrown at runtime.

Answer: C

## 54)01-12-22

(multi threading)

## Multi Threading

Syllabus

=====

1. Introduction.
2. The ways to define, instantiate and start a new Thread.
  1. By extending Thread class
  2. By implementing Runnable interface
3. Thread class constructors
4. Thread priority
5. Getting and setting name of a Thread.
6. The methods to prevent(stop) Thread execution.

1. yield()
2. join()
3. sleep()
7. Synchronization.
8. Inter Thread communication.
9. Deadlock
10. Daemon Threads.
11. Various Conclusion
1. To stop a Thread
2. Suspend & resume of a thread
3. Thread group
4. Green Thread
5. Thread Local
12. Life cycle of a Thread

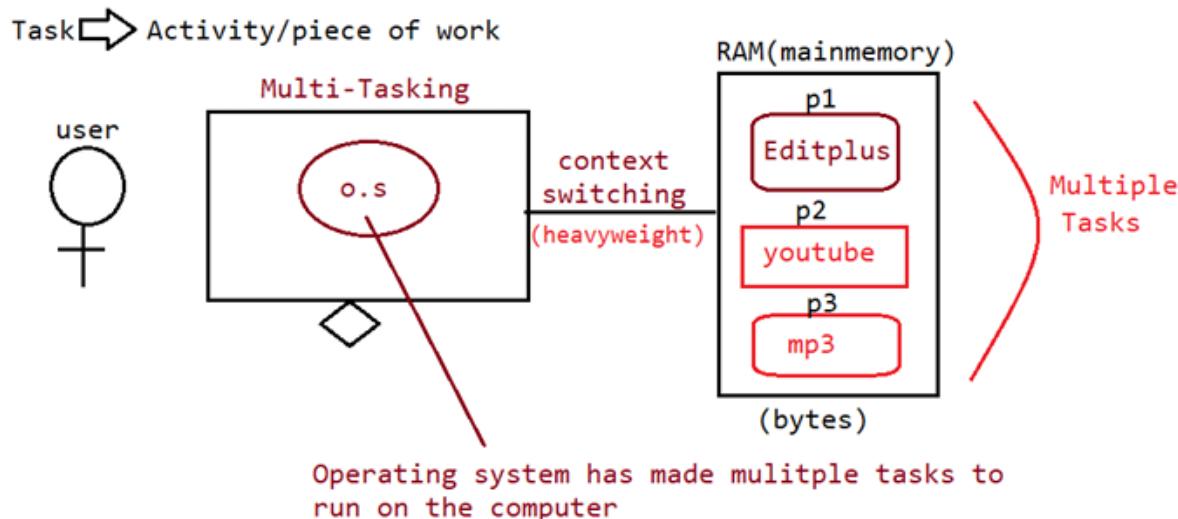
### MultiTasking

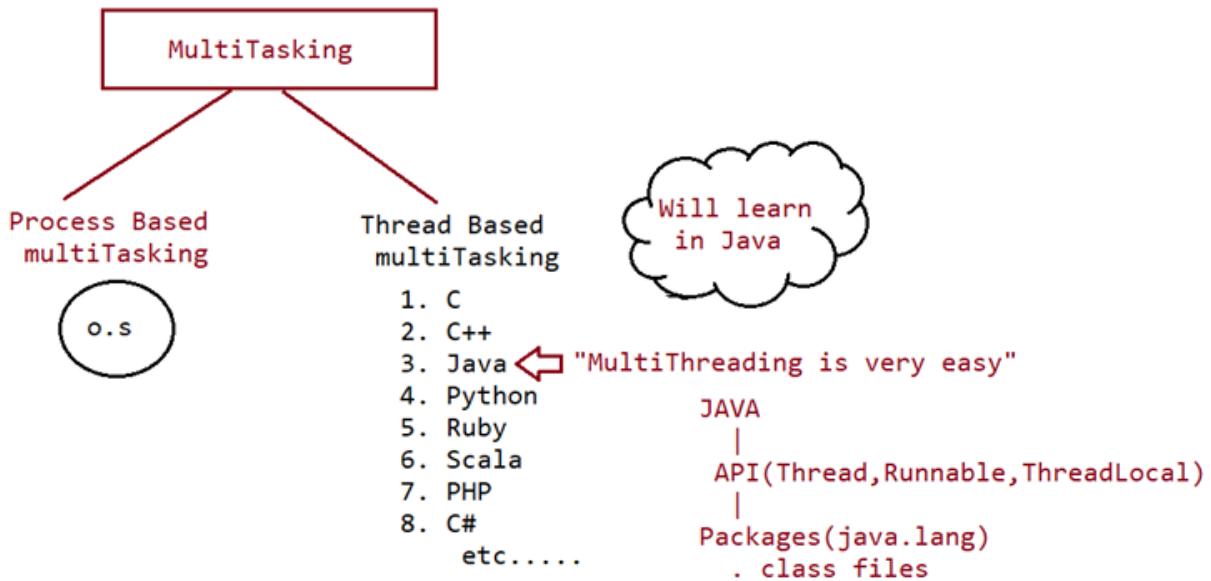
---

Executing several tasks simultaneously is the concept of multitasking.

There are 2 types of Multitasking.

- a. Process based multitasking
- b. Thread based multitasking.





90% work done by API

+

10% work will be done by developer(writing a task)

[Scheduling algorithms and paging algorithms] = OS

Api refers to collection of packages

Process based multitasking

=====

Executing several tasks simultaneously where each task is a separate independent process such type of multitasking is called

"process based multitasking".

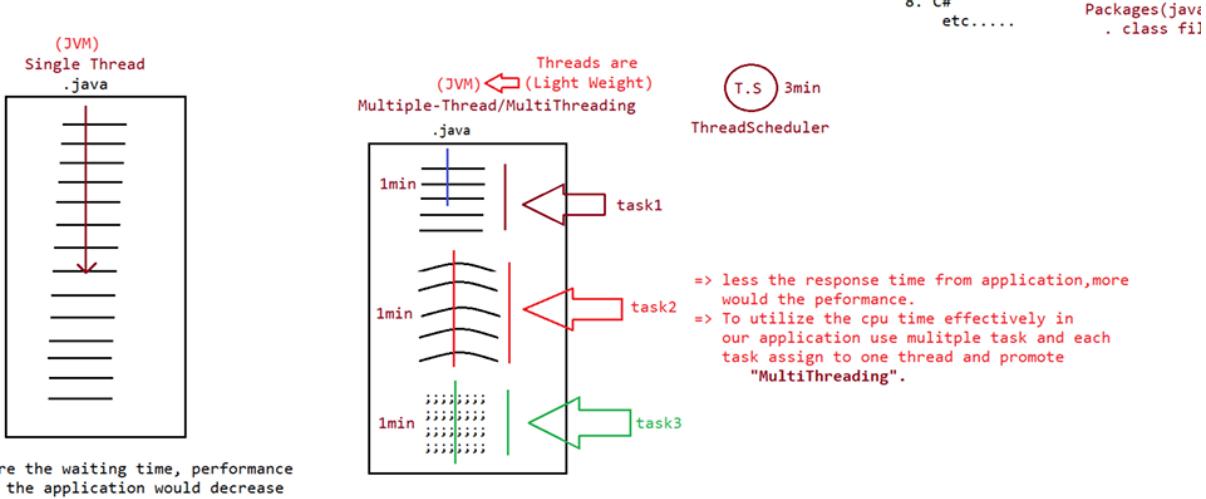
eg:: typing a java pgm

listening to a song

downloading the file from internet

Process based multitasking is best suited at "os level".

(till now all above concepts and programs are single thread based)



(create other line of executions for every task we do in program)

Line of execution is called as Thread

Independent tasks

Jvm is controlled the thread, not a memory level.

Jvm will do the multi threading

=In a single program, divide the tasks into multiple and assign it to multiple threads so that cpu can be effectively utilized.

Thread based multitasking

=====

=>Executing several tasks simultaneously where each task is a separate independent part of the same Program, is called

"Thread based MultiTasking".

Each independent part is called a "Thread".

1. This type of multitasking is best suited at "Programmatic level".

The main advantages of multitasking is to reduce the response time of the system and to improve the performance.

2. The main important application areas of multithreading are

- a. To implement multimedia graphics
- b. To develop web application servers(will learn in JEE)
- c. To develop video games
- d. To develop animations

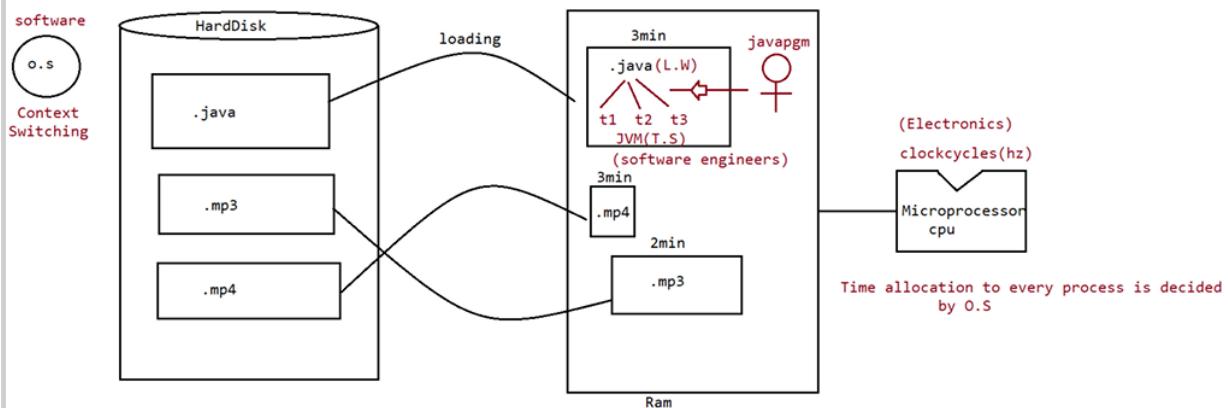
3. Java provides inbuilt support to work with threads through API called

Thread,Runnable,ThreadGroup,ThreadLocal,...

4. To work with multithreading, java developers will code only for 10% remaining 90% java API will take care..

**Agenda of MultiTasking**  
=====

1. To use CPU time effectively, so that performance can be improved



Cpu speed is measured in clock speed (hertz)

JVM(Task Scheduler )

[Software is still not capable enough of 100% effectively utilize the cpu]

What is thread?

A. Separate flow of execution is called "Thread".

if there is only one flow then it is called "SingleThread" programming.

For every thread there would be a separate job.

B. In java we can define a thread in 2 ways

a. Implementing Runnable interface

b. extending Thread class

1. Extending Thread class

=> we can create a Thread by extending a Thread.

```
class MyThread extends Thread{
 @Override
 public void run(){
 for(int i=0;i<10;i++)
 System.out.println("child thread");
 }
}
```

defining a thread(writing a class and extending a Thread)

job a thread(code written inside run())

class ThreadDemo{

```
 public static void main(String... args){
 MyThread t =new MyThread(); //Thread instantiation
 t.start(); //starting a thread
 ;;; // At this line 2 threads are there
 }
}
```

```
 for(int i=1;i<=5;i++)
 System.out.println("Main Thread");
 }
}
```

### Behind the scenes

1. Main thread is created automatically by JVM.
2. Main thread creates child thread and starts the child thread.

ThreadScheduler

=====

If multiple threads are waiting to execute, then which thread will execute 1st is decided by ThreadScheduler which is part of JVM.

In the case of MultiThreading we can't predict the exact output, only possible output we can expect.

Since jobs of threads are important, we are not interested in the order of execution it should just execute such that performance should be improved.

case2: diff b/w t.start() and t.run()

if we call t.start() and separate thread will be created which is responsible to execute run() method.

if we call t.run(), no separate thread will be created rather the method will be called just like normal method by main thread.

if we replace t.start() with t.run() then the output of the program would be

child thread

main thread

case3:: Importance of Thread class start() method

For every thread, required mandatory activities like registering the thread with Thread Scheduler will be taken care by Thread class start() method and programmer is responsible of just doing the job of the Thread inside run() method.

start() acts like an assistance to programmers.

```
public void start()
```

```
{
```

register thread with ThreadScheduler

All other mandatory low level activities  
invoke or call the run() method.

```
}
```

We can conclude that without executing the Thread class start() method there is no chance of starting a new Thread in java.

Due to this, start() is considered the heart of MultiThreading.

case4:: If we are not overriding run() method

If we are not Overriding run() method then Thread class run() method will be executed which has empty implementation and hence we won't get any output.

eg::

```
class MyThread extends Thread{}
class ThreadDemo{
 public static void main(String... args){
 MyThread t=new MyThread();
 t.start();
 }
}
```

It is highly recommended to override run() method, otherwise don't go for MultiThreading concept.

(Executable Framework is advance in Multithreading)

Eg: multi threading is common in almost every application.

Context switching in thread based in done by JVM

Thread parent is object

```

class MyThread extends Thread {
 @Override
 public void run() {
 for (int i=1;i<=10;i++) {
 System.out.println("Child thread");
 }
 }
} Task of a Thread

```

Defining  
a Thread

```

public class Test {
 public static void main(String[] args){
 MyThread t = new MyThread();
 t.start();
 //mainthread and userdefined thread
 for (int i = 1;i<=10;i++)
 {
 System.out.println("Main thread");
 }
 }
}

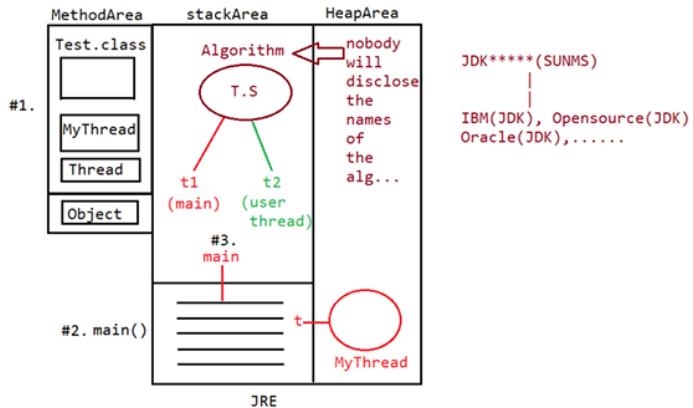
javac Test.java => Test.class,MyThread.class
java Test
 |=> contain main() so load and start
 the execution

```

```

public class Thread{
 public void run(){
 }
 public void start(){
 //logic internally available
 }
}

```



```

class MyThread extends Thread //thread is inbuilt API
{
 @Override
 public void run()//run is a predefined method present in Thread
 {
 //job of a thread
 for(int i=0;i<10;i++){
 System.out.println("Child thread");
 }
 } //task of a thread
} //defination of a thread

public class MT1 {
 public static void main(String args[])
 {
 MyThread t = new MyThread();
 t.start();

 for(int i=0;i<10;i++){
 System.out.println("main thread");
 }
 }
}

```

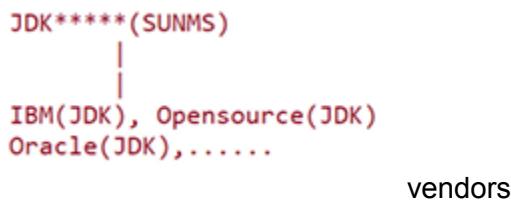
```
}
```

When t.start() method is called two threads are called  
That are mainthread and userdefined thread

### t.start()

It will create a new Thread which is responsible for executing run().

Now, it is the duty of thread scheduler to say which thread has to execute  
One is main thread  
Another is user defined thread  
Task Scheduler works based on Algorithms. No vendor will disclose the names and information  
of algorithm.



```
//logic of start
class Thread
{
 //start() is the heart of MultiThreading
 public void start(){
 1. Register the thread with T.S
 2. All other mandatory low level activities(memory level)
 3. Invoke or call run() method
 }
 public void run(){
 //no implementation
 }
}
```

## 29)Code Snippets

(2:59:00)

Q>

```
class Payload {
private int weight;
public Payload (int w) { weight = w; }
public void setWeight(int w) { weight = w; }
public String toString() { return Integer.toString(weight); }
}
```

```
public class TestPayload {
 static void changePayload(Payload p) {
 p.setWeight(420);
 /* insert code */ //Line 12
 }
 public static void main(String[] args) {
 Payload p = new Payload(200); // weight = 200
 p.setWeight(1024); // weight = 1024
 changePayload(p);
 System.out.println("p is " + p);
 }
}
```

Which code fragment, inserted at the end of line 12, produces the output p is 420?

- A. p.setWeight(420);
- B. p.changePayload(420);
- C. p = new Payload(420);
- D. Payload.setWeight(420);
- E. p = Payload.setWeight(420);

Answer: A

Q>

```
public static void test(String str) {
 int check = 4;
 if (check = str.length()) {
 System.out.print(str.charAt(check -= 1) + ", ");
 } else {
 }
}
```

System.out.print(str.charAt(0) + ", ");

and the invocation:

```
test("four");
test("tee");
test("to");
```

What is the result?

- A. r, t, t,
- B. r, e, o,
- C. Compilation fails.
- D. An exception is thrown at runtime.

Answer: C

Question

```
abstract class C1 {
 public C1() { System.out.print(1); }
}
class C2 extends C1 {
 public C2() { System.out.print(2); }
```

```
}
```

```
class C3 extends C2 {
```

```
}
```

```
public C3() { System.out.println(3); }
```

```
public class Ctest {
```

```
public static void main(String[] a) { new C3(); }
```

```
}
```

What is the result?

- A. 3
- B. 23
- C. 32
- D. 123
- E. 321
- F. Compilation fails.
- G. An exception is thrown at runtime.

Answer: D

Q>

Given:

```
class One {
```

```
public One foo() {
```

```
return this;
```

```
}
```

```
}
```

```
class Two extends One {
```

```
public One foo() {
```

```
return this;
```

```
}
```

```
}
```

```
class Three extends Two {
```

```
// insert method here
```

```
}
```

Which two methods, inserted individually, correctly complete the Three class?

(Choose two.)

- A. public void foo() {}
- B. public int foo() { return 3; }
- C. public Two foo() { return this; }
- D. public One foo() { return this; }
- E. public Object foo() { return this; }

Answer : C,D

Parent

| => IS-A relationship

|

Child

Covariant return types

=====

Object

|

String

Number

|

Integer

Q>

Given:

```
public interface A { public void m1(); }
class B implements A {} //CE
class C implements A { public void m1() {} }
class D implements A { public void m1(int x) {} } //CE
abstract class E implements A {}
abstract class F implements A { public void m1() {} }
abstract class G implements A { public void m1(int x) {} }
```

What is the result?

- A. Compilation succeeds.
- B. Exactly one class does NOT compile.
- C. Exactly two classes do NOT compile.
- D. Exactly four classes do NOT compile.
- E. Exactly three classes do NOT compile

Answer: C

Q>

Given:

```
1. class TestA {
2.
public void start() { System.out.println("TestA"); }
3.
4. public class TestB extends TestA {
5.
public void start() { System.out.println("TestB"); } //overridden method
6.
7.
8.
9. }
public static void main(String[] args) {
((TestA)new TestB()).start();
}
```

What is the result?

- A. TestA
- B. TestB
- C. Compilation fails.
- D. An exception is thrown at runtime.

Answer: B

Q>

Given:

```
class Line {
 public class Point {
 public int x, y;
 }
 public Point getPoint() {
 return new Point();
 }
}

class Triangle {
 public Triangle() {
 // insert code here line 16
 }
}
```

Which code, inserted at line 16, correctly retrieves a local instance of a Point object?

- A. Point p = Line.getPoint();
- B. Line.Point p = Line.getPoint();
- C. Point p = (new Line()).getPoint();
- D. Line.Point p = (new Line()).getPoint()

Answer: D

## 55)02-12-22

(multi threading 2)

18:00

```
class mt extends Thread
{
 @Override
 public void run(){
 System.out.println("no arg run method");
 run(5);
 }

 public void run(int i)
 {
 System.out.println("int arg run method");
 }
}
```

```

public class mt2{
 public static void main(String arg[]){
 mt t = new mt();
 t.start();

 //explicitly making a call
 t.run(6);
 for(int i = 0 ; i<10 ; i++)
 {
 System.out.println("main thread");
 }

 }
}

```

### No multithreading

```

class mt extends Thread
{
 @Override
 public void start()
 {
 System.out.println("start method called");
 }
 @Override
 public void run()
 {
 System.out.println("no arg run method");
 }
}

public class mt2{
 public static void main(String arg[]){
 mt t = new mt();
 //since our class start() is called , so no new thread is created.
 t.start();

 for(int i = 0 ; i<10 ; i++)
 {
 System.out.println("main thread");
 }

 }
}

```

```

}

class mt extends Thread
{
 @Override
 public void start()
 {
 super.start();
 System.out.println("start method called");
 }
 @Override
 public void run()
 {
 System.out.println("no arg run method");
 }
}
public class mt2{
 public static void main(String arg[]){
 mt t = new mt();

 t.start();

 for(int i = 0 ; i<10 ; i++)
 {
 System.out.println("main thread");
 }

 }
}

```

#### case5:Overloading of run() method

we can overload run() method but Thread class start() will always call run() with zero argument.

if we overload run method with arguments, then we need to explicitly call argument based run method and it will be executed just like normal method.

eg::

```

class MyThread extends Thread{
 public void run(){
 System.out.println("no arg method");
 }
}

```

```
public void run(int i){
 System.out.println("zero arg method");
}
}
}
```

```
class ThreadDemo{
 public static void main(String... args){
 MyThread t=new MyThread();
 t.start();
 }
}
```

Output:: NO arg method.

Case6::Overriding of start() method

If we override start() then our start() method will be executed just like normal method, but no new Thread will be created and no new Thread will be started.

eg#1.

```
class MyThread extends Thread{
 public void run(){
 System.out.println("no arg method");
 }
 public void start(){
 System.out.println("start arg method");
 }
}
```

```
class ThreadDemo{
 public static void main(String... args){
 MyThread t=new MyThread();
 t.start();
 }
}
```

Output:: start arg method

It is never recommended to override start() method.

case7::

```
class MyThread extends Thread{
 public void run(){
 System.out.println("run method");
 }
 public void start(){
 System.out.println("start method");
 }
}
```

```
class ThreadDemo{
 public static void main(String... args){
 MyThread t=new MyThread();
 }
}
```

```

t.start();
System.out.println("Main method");
}
}

```

Output::

- MainThread
- a. Main method
- b. start method.

eg#2.

```

class MyThread extends Thread{
public void start(){
super.start();
}
System.out.println("start method");
public void run(){
System.out.println("run method");
}
}
class ThreadDemo{
public static void main(String... args){
MyThread t=new MyThread();
t.start();
System.out.println("Main method");
}
}

```

Output::

- MainThread
- a. Main method
- b. start method

UserDefinedThread

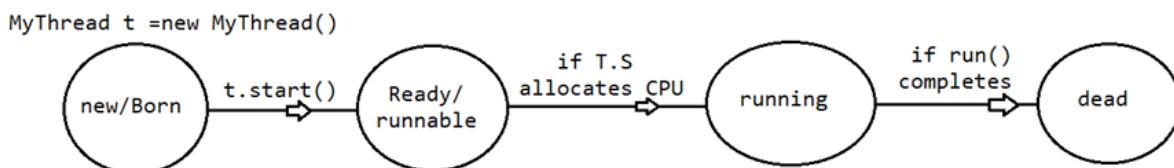
- a. run method

case8:: Life cycle of a Thread

```

Life cycle of Thread
=====

```



(Basic Life Cycle of Thread)

[Physically if I am hitting by gym is no use,  
Until unless you are mentally strong,

No matter what happens you will face.  
If that is there only, we can able to live.  
Otherwise no use.]  
–Nitin M-

MyThread t=new MyThread(); // Thread is in born state  
t.start(); //Thread is in ready/runnable state  
if Thread scheduler allocates CPU time then we say thread entered into Running state.  
if run() is completed by thread then we say thread entered into dead state.  
=> Once we created a Thread object then the Thread is said to be in new state or born state.  
=> Once we call start() method then the Thread will be entered into Ready or Runnable state.  
=> If Thread Scheduler allocates CPU then the Thread will be entered into running state.  
=> Once run() method completes then the Thread will enter into dead state.

case9::

After starting the Thread, we are not supposed to start the same Thread again, then we say Thread is in "IllegalThreadStateException".

MyThread t=new MyThread(); // Thread is in born state  
t.start(); //Thread is in ready state  
....  
....  
t.start(); //IllegalThreadStateException(unchecked exception)

Creation of Thread using Runnable interface

```
=====
1. Creating a Thread using java.lang.Thread class
 a. use start() from Thread class
 b. override run() and define the job of the Thread.
2. Creation of a Thread requirement to SUNMS is an SRS
interface Runnable{ //srs is represented as interface
 void run();
}
class Thread implements Runnable
{ // Adapter class (= to support the class)
public void start(){
 1. Register the thread with ThreadScheduler
 2. All other mandatory low level activities(memory level)
 3. invoke or call run() method
}
public void run(){
 //job for a thread
}
}
```

### shortcuts of eclipse

ctrl+shift+T => To open a definition of any class

ctrl + o => To list all the methods of the class

Note: public java.lang.Thread();

run()

|=> thread class start(), followed by thread class

public java.lang.Thread(java.lang.Runnable);

|=> thread class start(), followed by implementation  
class of Runnable run()

(adaptor class are mostly abstract, in some cases that are not abstract)

Defining a Thread by implementing Runnable Interface

```
=====
public interface Runnable{
 public abstract void run();
}

public class Thread implements Runnable{
 public void start(){
 1. register Thread with ThreadScheduler
 2. All other mandatory low level activites
 3. invoke run()
 }

 public void run(){
 //empty implementation
 }
}
```

eg::1

```
class MyRunnable implements Runnable{
 @Override
 public void run(){
 for(int i=1;i<=10;i++)
 System.out.println("child thread");
 }
}

public class ThreadDemo{
 public static void main(String... args){
 MyRunnable r=new MyRunnable();
 Thread t=new Thread(r);//call MyRunnable run()
 t.start();
 for(int i=1;i<=10;i++)
 }
}
```

```
 System.out.println("main thread");
 }
}
```

Output::

MainThread  
a. main thread  
....  
....

ChildThread  
a. child thread  
...  
...  
....

#### Case study

```
=====
```

```
MyRunnable r=new MyRunnable();
Thread t1=new Thread();
Thread t2=new Thread(r);
case1: t1.start()
Thread class run()
output
mainthread
A new thread will be created,which is responsible for executing
main thread
main thread
main thread
main thread
main thread
```

```
case2: t2.start()
MyRunnable run()
output
mainthread
A new thread will be created,which is responsible for executing
main thread
main thread
main thread
main thread
main thread
```

```
userdefinedthread
 child thread
 child thread
```

child thread  
child thread  
child thread

case3: t1.run()

No new thread will be created, but Thread class run() will be executed just like normal method call.

output

mainthread  
main thread  
main thread  
main thread  
main thread  
main thread

case4: t2.run()

No new thread will be created, but MyRunnable class run() will be executed just like normal method call.

output

mainthread  
    child thread  
child thread  
child thread  
child thread  
child thread  
main thread  
main thread  
main thread  
main thread  
main thread

case5: r.start() //CE

case6. r.run()

No new thread will be created, but MyRunnable class run() will be executed just like normal method call.

output

mainthread  
    child thread  
child thread  
child thread  
child thread  
child thread  
main thread  
main thread  
main thread

```

main thread
main thread
MyRunnable r=new MyRunnable();
Thread t1=new Thread();
Thread t2=new Thread(r);
case1: t1.start()
case2: t2.start()
case3: t2.run()
case4: t1.run()
case5: r.start()
case6: r.run()

```

In which of the above cases a new Thread will be created which is responsible for the execution of MyRunnable run() method ?

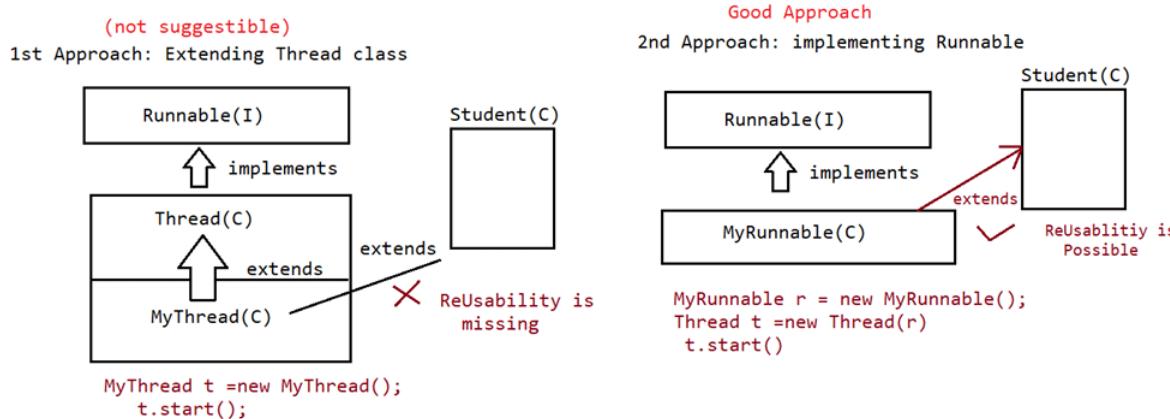
```
t2.start();
```

In which of the above cases a new Thread will be created ?

```
t1.start();
t2.start();
```

In which of the above cases MyRunnable class run() will be executed?

```
t2.start();
t2.run();
r.run();
```



1st case inheritance is not possible(we can't extend another class)

2nd case we can take benefit of inheritance, we can extend another class

Different approach for creating a Thread?

- A. extending Thread class
- B. implementing Runnable interface

Which approach is the best approach?

- a. implements Runnable interface is recommended becoz our class can extend other class through

which inheritance benefit can brought in to our class.

Internally performance and memory level is also good when we work with interface.

b. if we work with extends feature then we will miss out inheritance benefit becoz already our class has inherited the feature from "Thread class", so we normally don't prefer extends approach rather implements approach is used in real time for working with "MultiThreading".

Various Constructors available in Thread class

- ```
=====
a. Thread t=new Thread()
b. Thread t=new Thread(Runnable r)
c. Thread t=new Thread(String name)
d. Thread t=new Thread(Runnable r,String name)
e. Thread t=new Thread(ThreadGroup g, String name);
f. Thread t=new Thread(ThreadGroup g, Runnable r);
g. Thread t=new Thread(ThreadGroup g, Runnable r,String name);
h. Thread t=new Thread(ThreadGroup g, Runnable r,String name,long stackSize);
```

Alternate approach to define a Thread(not recommended)

```
=====
class MyThread extends Thread{
public void run(){
System.out.println("child thread");
}
}
class ThreadDemo {
public static void main(String... args){
MyThread t=new MyThread();
Thread t1=new Thread(t);
t1.start();
System.out.println("main thread");
}
}
```

Output::2 threads are created

```
MainThread
    main thread
ChildThread
    child thread
```

internally related

```
=====
Runnable
 ^
|
Thread
```

```
^
|
MyThread
```

Names of the Thread

Internally for every thread, there would be a name for the thread.

- a. name given by jvm
- b. name given by the user.

eg::

```
class MyThread extends Thread{
}
public class TestApp{
public static void main(String... args){
System.out.println(Thread.currentThread().getName());//main
MyThread t=new MyThread();
t.start();
System.out.println(t.getName());//Thread-0
Thread.currentThread().setName("Yash");//Yash
System.out.println(Thread.currentThread().getName());//Yash
System.out.println(10/0);
//Exception in thread "yash" java.lang.ArithmetricException:/by zero
TestApp.main()
}
}
```

=> It is also possible to change the name of the Thread using setName().

=> It is possible to get the name of the Thread using getName().

methods

```
public final String getName();
public final void setName(String name);
```

eg#2.

```
class MyThread extends Thread{
@Override
public void run(){
    System.out.println("run() executed by Thread ::"
"+Thread.currentThread().getName());
}
}
public class TestApp{
public static void main(String... args){
MyThread t=new MyThread();
t.start();
System.out.println("main() executed by Thread ::
```

```

"+Thread.currentThread().getName());
}
}
Output:: run() executed by Thread:: Thread-0
        main() executed by Thread:: main

class MyThread extends Thread
{
    @Override
    public void run()
    {
        String name = Thread.currentThread().getName();
        System.out.println("run() is executed by :: "+name);
    }
}
public class Test{
    public static void main(String arg[])
    {
        String name = Thread.currentThread().getName();
        System.out.println("main() is executed by :: Thread- "+name);

        MyThread t = new MyThread();
        t.start();
        //Thread.currentThread().setName("Yash Thread");
        System.out.println("Name of child thread is :: "+t.getName());
    }
}

```

30)Code Snippets

(2:53:53)

Q>

```

public class Breaker {
    static String o = "";
    public static void main(String[] args) {
        z: o = o + 2;
        for (int x = 3; x < 8; x++) {
            if (x == 4)
                break;
            if (x == 6)
                break z;
            o = o + x;
        }
    }
}
```

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

What is the result?

- A. 23
- B. 234
- C. 235
- D. 2345
- E. 2357
- F. 23457
- G. Compilation fails.

Answer: G

Q>

```
public class Tahiti {
    Tahiti t;
    public static void main(String[] args) {
        Tahiti t = new Tahiti();
        Tahiti t2 = t.go(t);
        t2 = null;
        // more code here 11
    }
    Tahiti go(Tahiti t) {
        Tahiti t1 = new Tahiti();
        Tahiti t2 = new Tahiti();
        t1.t = t2;
        t2.t = t1;
        t.t = t2;
        return t1;
    }
}
```

When line 11 is reached, how many objects are eligible for garbage collection?

- A. 0
- B. 1
- C. 2
- D. 3

Answer: 0(island of isolation)

Q>

```
public class ItemTest {
    private final int id;
    public ItemTest(int id) {
        this.id = id;
    }
    public void updateId(int newId) {
        id = newId;
```

```
}

public static void main(String[] args) {
    ItemTest fa = new ItemTest(42);
    fa.updateId(69);
    System.out.println(fa.id);
}
}
```

What is the result?

- A. Compilation fails.
- B. An exception is thrown at runtime.
- C. The attribute id in the ItemTest object remains unchanged.
- D. The attribute id in the ItemTest object is modified to the new value.
- E. A new ItemTest object is created with the preferred value in the id attribute.

Answer: A

Q>

```
class Foo {
    private int x;
    public Foo( int x ){ this.x = x;}
    public void setX( int x ) { this.x = x; }
    public int getX(){ return x;}
}

public class Gamma {
    static Foo fooBar(Foo foo) {
        foo = new Foo(100);
        return foo;
    }

    public static void main(String[] args) {
        Foo foo = new Foo( 300 );
        System.out.println( foo.getX() + "-" );
        Foo fooFoo = fooBar(foo);
        System.out.println(foo.getX() + "-");
        System.out.println(fooFoo.getX() + "-");
        foo = fooBar( fooFoo );
        System.out.println( foo.getX() + "-" );
        System.out.println(fooFoo.getX());
    }
}
```

What is the output of the program shown in the exhibit?

- A. 300-100-100-100-100
- B. 300-300-100-100-100
- C. 300-300-300-100-100
- D. 300-300-300-300-100

Answer: B

Q>

```

interface Fish {}
class Perch implements Fish {}
class Walleye extends Perch {}
class Bluegill {}
public class Fisherman {
    public static void main(String[] args) {
        Fish f = new Walleye();
        Walleye w = new Walleye();
        Bluegill b = new Bluegill();
        if (f instanceof Perch)
            System.out.print("f-p ");
        if (w instanceof Fish)
            System.out.print("w-f ");
        if (b instanceof Fish)
            System.out.print("b-f ");
    }
}

```

What is the result?

- A. w-f
- B. f-p w-f
- C. w-f b-f
- D. f-p w-f b-f
- E. Compilation fails.
- F. An exception is thrown at runtime

Answer: B

```

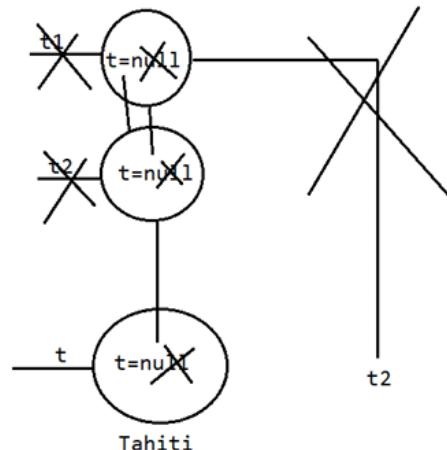
public class Tahiti {

    Tahiti t;

    Tahiti go(Tahiti t) {
        Tahiti t1 = new Tahiti();
        Tahiti t2 = new Tahiti();

        t1.t = t2;
        t2.t = t1;
        t.t = t2;
        return t1;
    }
    public static void main(String[] args) {
        Tahiti t = new Tahiti();
        Tahiti t2 = t.go(t);
        t2 = null;
        // more code here 11
    }
}

```



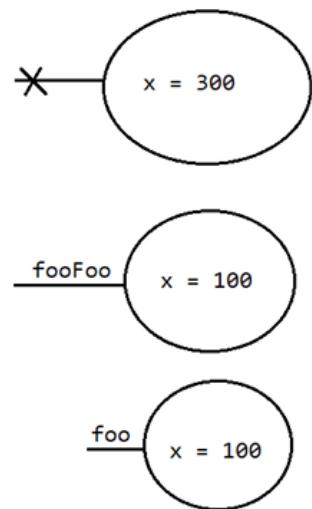
```

class Foo {
    private int x;
    public Foo( int x ){ this.x = x; }
    public void setX( int x ) { this.x = x; }
    public int getX(){ return x; }
}

public class Gamma {
    static Foo fooBar(Foo foo) {
        foo = new Foo(100);
        return foo;
    }
    public static void main(String[] args) {
        Foo foo = new Foo( 300 );
        System.out.println( foo.getX() + "-" );
        Foo fooFoo = fooBar(foo);
        System.out.println(foo.getX() + "-");
        System.out.println(fooFoo.getX() + "-");
        foo = fooBar( fooFoo );
        System.out.println( foo.getX() + "-" );
        System.out.println(fooFoo.getX());
    }
}

```

300-300-100 -100-100



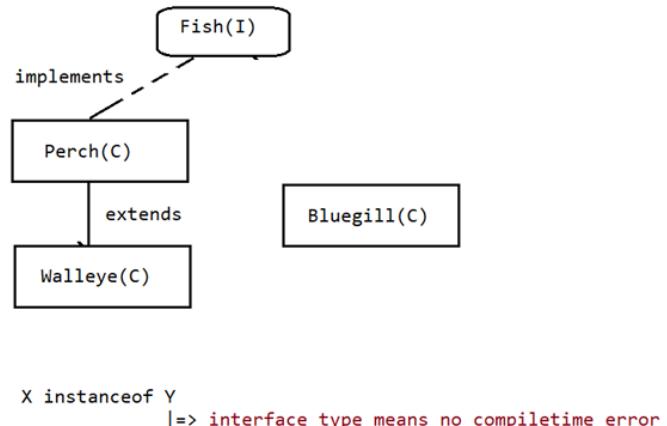
```

Q>
interface Fish {}
class Perch implements Fish {}
class Walleye extends Perch {}
class Bluegill {}

public class Fisherman {
    public static void main(String[] args) {
        Fish f = new Walleye();
        Walleye w = new Walleye();
        Bluegill b = new Bluegill();

        if (f instanceof Perch)
            System.out.print("f-p "); ✓
        if (w instanceof Fish)
            System.out.print("w-f "); ✓
        if (b instanceof Fish)
            System.out.print("b-f "); ✗
    }
}

```



56)03-12-22 dbt

57)05-12-22

(multi threading 3)(12:00)

ThreadPriorities

=====

For every Thread in java has some priority.

The valid range of priority is 1 to 10, it is not 0 to 10.

If we try to give a different value the it would result in "IllegalArgumentException".

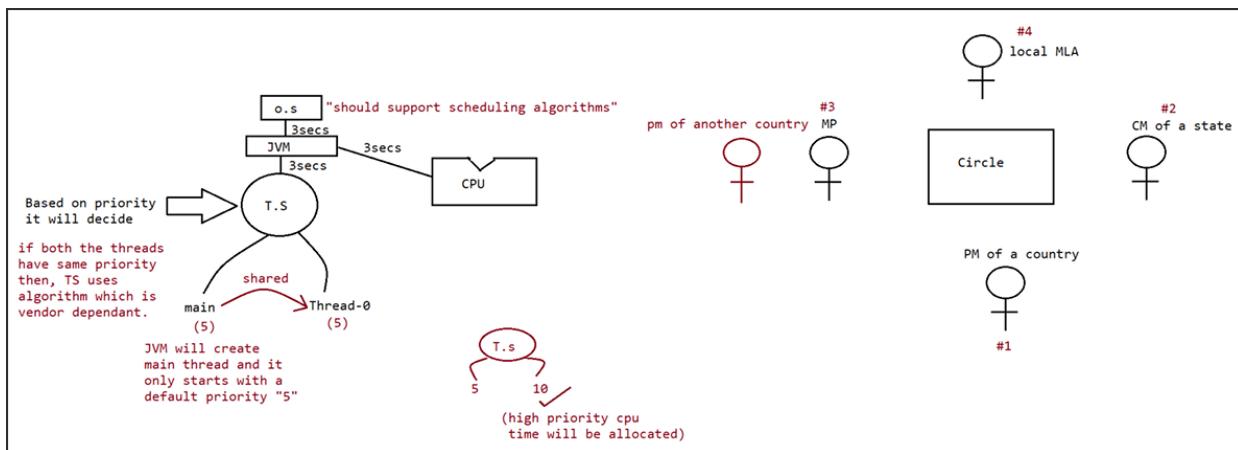
Thread.MIN_PRIORITY = 1

Thread.MAX_PRIORITY = 10

Thread.NORM_PRIORITY = 5

Thread class does not have priorities is Thread.LOW_PRIORITY, Thread.HIGH_PRIORITY.

Thread scheduler allocates cpu time based on "Priority".



Priority is shared or inherited

`sop("priority of thread is "+Thread.currentThread().getPriority());`

If both the threads have the same priority then which thread will get a chance as a prgm we can't predict becoz it is vendor dependent.

We can set and get priority values of the thread using the following methods

- public final void setPriority(int priorityNumber)
- public final int getPriority()

The allowed priorityNumber is from 1 to 10, if we try to give other values it would result in "IllegalArgumentException".

`System.out.println(Thread.currentThread().setPriority(100)); //IllegalArgumentException.`

Every os will support the scheduling algorithm

If it is cracked version then the priority will be wrong due to wrong scheduling algorithm

Dll file is required. shot a mail to os vendors. They will send it.

DefaultPriority

=====

The default priority for only main thread is "5", where as for other threads priority will be inherited from parent to child.

Parent Thread priority will be given as Child Thread Priority.

eg#1.

```
class MyThread extends Thread{}  
public class TestApp{  
    public static void main(String... args){  
        System.out.println(Thread.currentThread().getPriority());//5  
        Thread.currentThread().setPriority(7);  
        MyThread t= new MyThread();  
        System.out.println(Thread.currentThread().getPriority());//7  
    }  
}
```

reference

=====

```
Thread  
^  
|extends  
|  
MyThread
```

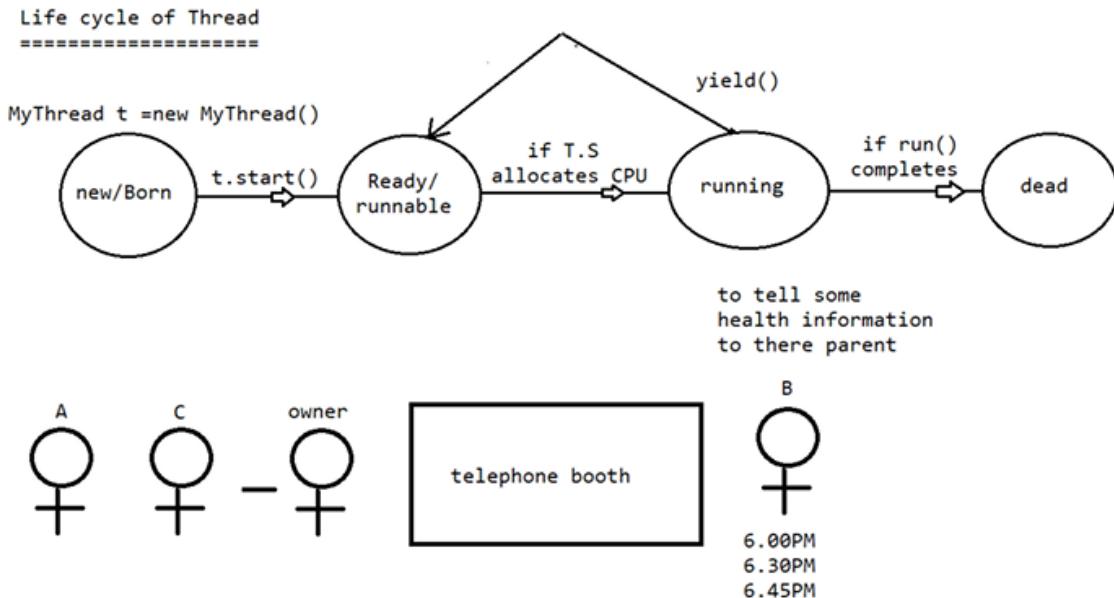
MyThread is creating by "mainThread", so priority of "mainThread" will be shared as a priority for "MyThread".

eg#2.

```
class MyThread extends Thread{  
    @Override  
    public void run(){  
        for (int i=1;i<=5 ;i++ ){  
            System.out.println("child thread");  
        }  
    }  
}  
  
public class TestApp{  
    public static void main(String... args){  
        MyThread t= new MyThread();  
        t.setPriority(7);//line -1  
        t.start();  
        for (int i=1; i<=5; i++){  
            System.out.println("main thread");  
        }  
    }  
}
```

Since priority of child thread is more than main thread, jvm will execute child thread first whereas for the parent thread priority is 5 so it will get last chance.

if we comment line-1, then we can't predict the order of execution becoz both the threads have same priority.



Some platforms won't provide proper support for Thread priorities.

eg:: windows7,windows10,...

We can prevent Threads from Execution

- a. yield()
- b. sleep()
- c. join()

yield() => It causes to pause current executing Thread for giving chance for waiting Threads of same priority.

If there is no waiting Threads or all waiting Threads have low priority then same Thread can continue its execution.

If all the threads have same priority and if they are waiting then which thread will get chance we can't expect, it depends on ThreadScheduler.

The Thread which is yielded, when it will get the chance once again depends on the

mercy on "ThreadScheduler" and we can't expect exactly.

public static native void yield()

MyThread t= new MyThread() //new state or born state

t.start() // enter into ready state/runnable state

if ThreadScheduler allocates processor then enters into running state.

a. if running Thread calls yield() then it enters into runnable state.

if run() is finished with execution then it enters into dead state.

eg#1.

```
class MyThread extends Thread{  
    @Override  
    public void run(){  
        for (int i=1;i<=5 ;i++ ){  
            System.out.println("child thread");  
            Thread.yield(); //line-1  
        }  
    }  
}  
}  
  
public class TestApp{  
    public static void main(String... args){  
        MyThread t= new MyThread();  
        t.start();  
        for (int i=1;i<=5 ;i++ ){  
            System.out.println("Parent Thread");  
        }  
    }  
}
```

(native means is code is written in some other language at the runtime binding will happens

Native is one of the access modifier)

Note::

If we comment line-1, then we can't expect the output becoz both the threads have same priority then which

thread the ThreadScheduler will schedule is not in the hands of programmer but if we don't comment line-1,

then there is a possibility of main thread getting more no of times, so main thread execution is faster than

child thread will get a chance.

Note: Some platforms won't provide proper support for yield(), because it is getting the execution code from other language preferably from 'C'.

b. join()

If the thread has to wait until the other thread finishes its execution then we need to go for join().

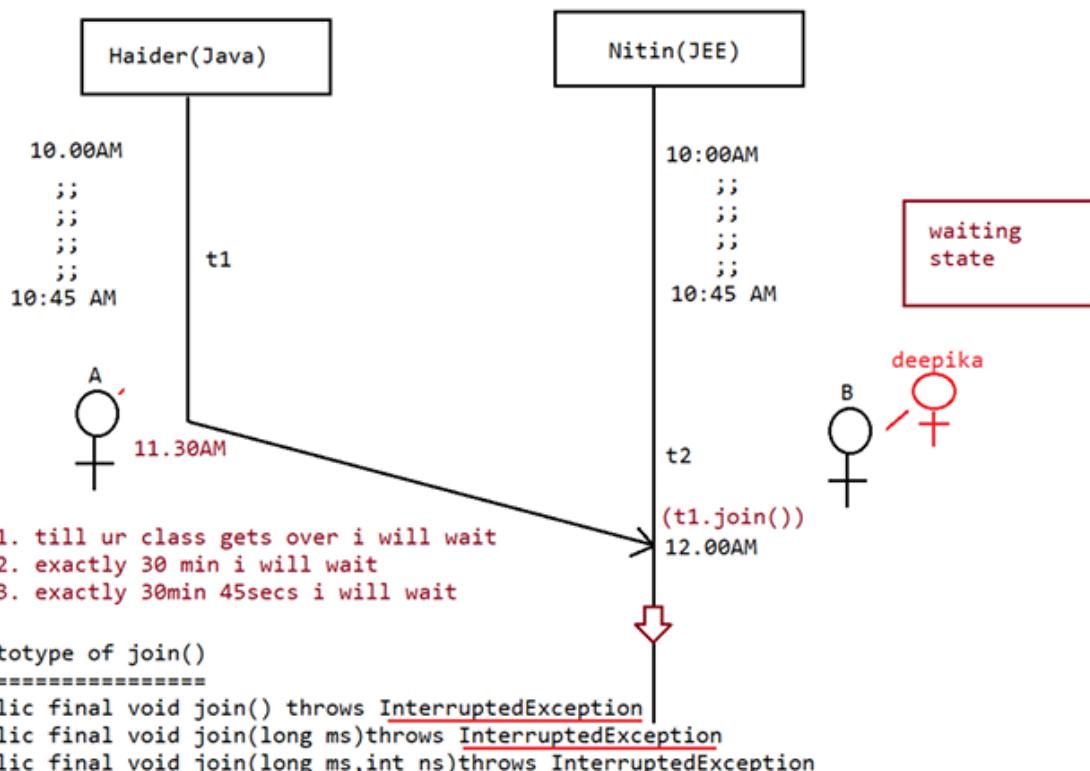
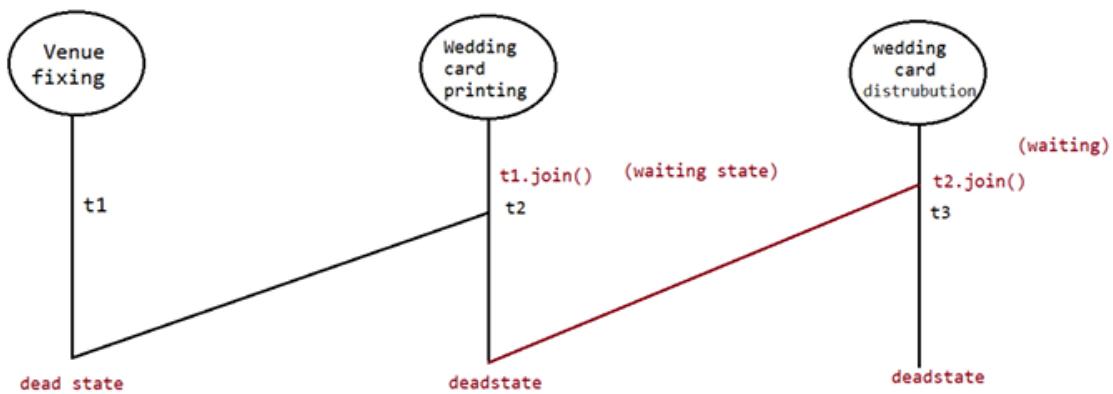
if t1 executes t2.join() then t1 should wait till t2 finishes its execution.

t1 will be entered into waiting state until t2 completes, once t2 completes then eg#1.

t1 can continue with its execution.

venue fixing =====> t1.start()

wedding card printing =====> t2.start()=====> t1.join()
wedding card distribution =====> t3.start()=====> t2.join()



Prototype of join()

=====
=====

```
public final void join() throws InterruptedException  
public final void join(long ms) throws InterruptedException
```

```
public final void join(long ms,int ns) throws InterruptedException
```

Note: While one thread is in waiting state and if one more thread interupts then it would result

in "InterruptedException".InterruptedException is checkedException which should always be handled.

Thread t =new Thread();//new/born state

t.start(); //ready/runnable state-> If T.S allocates cpu time then Thread enters into running state-> If currently executing Thread invokes t.join() /t.join(1000),t.join(1000,100), then it

would enter into waiting state.-> If the thread finishes the execution/time expires/interupted then it would come

back to

ready state/runnable state.

-> If run() is completed then it would enter into dead state.

eg#1.

```
class MyThread extends Thread{
```

```
@Override
```

```
public void run(){
```

```
for (int i=1;i<=10 ;i++ ){
```

```
System.out.println("Sita Thread");
```

```
try{
```

```
Thread.sleep(2000);
```

```
}
```

```
catch (InterruptedException e){
```

```
}
```

```
}
```

```
}
```

```
}
```

```
public class Test3 {
```

```
public static void main(String... args) throws InterruptedException{
```

```
MyThread t=new MyThread();
```

```
t.start();
```

```
t.join(10000);//line-n1
```

```
for (int i=1;i<=10;i++ ){
```

```
System.out.println("rama thread");
```

```
}
```

```
}
```

```
}
```

=> If line-n1 is commented then we can't predict the output becoz it is the duty of the T.S to assign C.P.U time

=> If line-n1 is not commented, then rama thread(main thread) will enter into waiting state till

sita thread(child thread) finishes its execution.

Output
2 Threads
a. Child Thread
sita thread
sita thread
....
b. Main Thread
rama thread
rama thread
....

Waiting of Child Thread until Completing Main Thread

=====

we can make main thread to wait for child thread as well as we can make child thread also to wait for main thread.

eg#1.

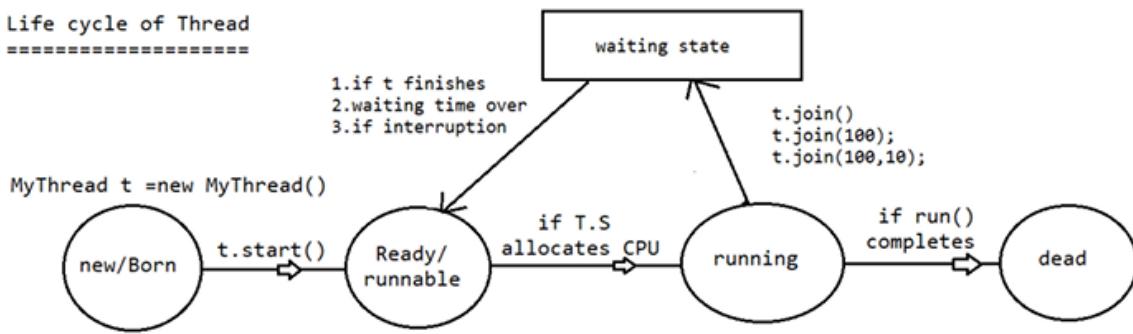
```
class MyThread extends Thread{  
    static Thread mt;  
    @Override  
    public void run(){  
        try{  
            mt.join();  
        }  
        catch (InterruptedException e){  
        }  
        for (int i=1;i<=10 ;i++ ){  
            System.out.println("child thread");  
        }  
    }  
}  
  
public class Test3 {  
    public static void main(String... args)throws InterruptedException{  
        MyThread.mt=Thread.currentThread();  
        MyThread t=new MyThread();  
        t.start();  
        for (int i=1;i<=10;i++ ){  
            System.out.println("main thread");  
            Thread.sleep(2000);//20sec sleep  
        }  
    }  
}
```

Output

2 Threads(MainThread,ChildThread)
MainThread
a. main thread

```
....  
....  
ChildThread  
a. child thread  
....  
....  
eg#2.  
class MyThread extends Thread{  
static Thread mt;  
@Override  
public void run(){  
try{  
}  
mt.join();  
catch (InterruptedException e){  
}  
for (int i=1;i<=10 ;i++ ){  
System.out.println("child thread");  
}  
}  
}  
}  
}  
public class Test3 {  
public static void main(String... args)throws InterruptedException{  
MyThread.mt=Thread.currentThread();  
MyThread t=new MyThread();  
t.start();  
t.join();  
for (int i=1;i<=10;i++ ){  
System.out.println("main thread");  
Thread.sleep(2000);//20sec sleep  
}  
}  
}  
}  
}  
output:  
2 threads(Main,child thread)  
main thread  
:::::  
:::::  
child thread
```

Life cycle of Thread
=====



//child thread is waiting for main thread

```

class MyThread extends Thread{

    static Thread mt;

    @Override
    public void run() {
        try {
            mt.join();
        } catch (InterruptedException e) {
            // TODO: handle exception
            e.printStackTrace();
        }
        for(int i=0;i<5;i++)
            System.out.println("child thread");
    }
}

public class mt4 {
    public static void main(String... args) throws InterruptedException{
        MyThread.mt = Thread.currentThread();

        MyThread t = new MyThread();
        t.start();
        // t.join(2000); //line-n1
        for (int i=1;i<=10;i++ ){
            Thread.sleep(2000);
            System.out.println("main thread");
        }
    }
}
  
```

```
}
```

Note::

DeadLock

If both the threads invoke t.join(),mt.join() then the program would result in "deadlock".

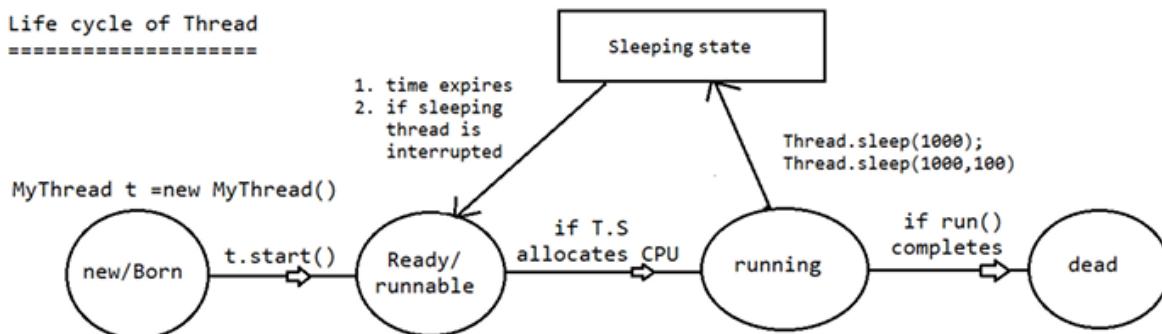
eg#3.

```
public class Test3 {  
    public static void main(String... args) throws InterruptedException{  
        Thread.currentThread().join();  
    }  
}
```

Output:: Deadlock, becoz main thread is waiting for the main thread itself.

sleep()

=====



If a thread don't want to perform any operation for a particular amount of time then we should go for sleep().

Signature

```
public static native void sleep(long ms) throws InterruptedException
```

```
public static void sleep(long ms,int ns) throws InterruptedException
```

every sleep method throws InterruptedException, which is a checked exception so we should compulsorily handle the exception using

try catch or by throws keyword otherwise it would result in compile time error.

```
Thread t=new Thread(); //new or born state
```

```
t.start() // ready/runnable state
```

=> If T.S allocates cpu time then it would enter into running state.

=> If run() completes then it would enter into dead state.

=> If running thread invokes sleep(1000)/sleep(1000,100) then it would enter into Sleeping state

=> If time expires/ if the sleeping thread got interrupted then the thread would come back to "ready/runnable state".

```

eg#1.
public class SlideRotator {
public static void main(String... args) throws InterruptedException{
for (int i=1;i<=10 ;i++ ){
System.out.println("Slide: "+i);
Thread.sleep(5000);
}
}
}
Output::
Slide:: 1
Slide:: 2
Slide:: 3
Slide:: 4
Slide:: 5
Slide:: 6
Slide:: 7
Slide:: 8
Slide:: 9
Slide:: 10

```

31)Code Snippets

(2:53:00)

Given:

```

class Converter {
public static void main(String[] args) {
Integer i = args[0]; // line 13
int j = 12;
System.out.println("It is " + (j == i) + " that j==i.");
}
}

```

What is the result when the programmer attempts to compile the code and run it with the command

`java Converter 12?`

- A. It is true that `j==i`.
- B. It is false that `j==i`.
- C. An exception is thrown at runtime.
- D. Compilation fails because of an error in line 13.

Answer: D

QUESTION

Click the Exhibit button.

```
1. public class A {  
2.  
public String doit(int x, int y){  
3.  
4.  
5.  
6.  
7.  
8.  
9. }
```

Given:

```
return "a";  
}  
public String doit(int... vals){  
return "b";  
}
```

25. A a = new A();

26. System.out.println(a.doit(4, 5));

What is the result?

- A. Line 26 prints "a" to System.out.
- B. Line 26 prints "b" to System.out.
- C. An exception is thrown at line 26 at runtime.
- D. Compilation of class A will fail due to an error in line 6.

Answer: A

Which two code fragments correctly create and initialize a static array of int elements? (Choose two.)

- A. static final int[] a = { 100,200 };
- B. static final int[] a; static { a=new int[2]; a[0]=100; a[1]=200; }
- C. static final int[] a = new int[2]{ 100,200 };
- D. static final int[] a;
 static void init() { a = new int[3]; a[0]=100; a[1]=200; }

if a variable is static an final the only place where we can initialize is

a. at the time of declaration

b. inside static block

```
static final int[] a = { 100,200 };
```

```
static{
```

```
    a = new int[2];
```

```
    a[0] = 100;
```

```
    a[1] = 200;
```

```
}
```

Q>

```
1. public class GoTest {  
2.  
public static void main(String[] args) {
```

```

3.
4.
5.
6.
7. }
8.
Sente a = new Sente(); a.go();
Goban b = new Goban(); b.go();
Stone c = new Stone(); c.go();
}
9. class Sente implements Go {
10.
public void go(){
11.
12. }
13.}
14.
System.out.println("go in Sente");
15.class Goban extends Sente {
16.
public void go(){
17.
18. }
19.
20.}
System.out.println("go in Goban");
21.class Stone extends Goban implements Go{
22.}
23.
24.interface Go { public void go(); }

What is the result?
A. go in Goban go in Sente go in Sente
B. go in Sente go in Sente go in Goban
C. go in Sente go in Goban go in Goban
D. go in Goban go in Goban go in Sente
E. Compilation fails because of an error in line 17.

Answer: C

Q>
What statements are true about the following code? (Choose all that apply.)
public class Tail {}
public class Animal {//Animal HAS-A name
public String name;
}
public class Canine extends Animal {//Canine IS-A Animal, Canine HAS-A Tail, Canine

```

```
HAS-A name  
public Tail tail;  
}  
public class Wolf extends Canine {}//Wolf IS-A Canine,Wolf IS-A Animal,Wolf HAS-A  
Tail,Wolf HAS-A name
```

- A. Wolf has-a name.
- B. Wolf has-a Tail.
- C. Wolf is-a Tail.
- D. Wolf is-a Animal.
- E. Canine is-a Wolf.
- F. Animal has-a Tail.

Answer: A,B,D

Q>

Which is a true statement about the following code?

```
public class IsItFurry {  
    static interface Mammal {}  
    static class Furry implements Mammal {}  
    static class Chipmunk extends Furry {}  
        public static void main(String[] args) {  
            Chipmunk c = new Chipmunk();  
            Mammal m = c;  
            Furry f = c;  
            int result = 0;  
            if (c instanceof Mammal) result += 1;  
            if (c instanceof Furry) result += 2;  
            if (null instanceof Chipmunk) result += 4;  
            System.out.println(result);  
        }  
}
```

- A. The output is 0.
- B. The output is 3.
- C. The output is 7.
- D. c instanceof Mammal does not compile.
- E. c instanceof Furry does not compile.
- F. null instanceof Chipmunk does not compile.

Answer: B

Q>

Which of the following can be inserted in main?

```
public class Outer {  
    class Inner {}  
    public static void main(String[] args) {  
        // INSERT CODE HERE  
    }  
}
```

- A. Inner in = new Inner();
- B. Inner in = Outer.new Inner();
- C. Outer.Inner in = new Outer.Inner();
- D. Outer.Inner in = new Outer().Inner();
- E. Outer.Inner in = new Outer().new Inner();
- F. Outer.Inner in = Outer.new Inner();

Answer: E

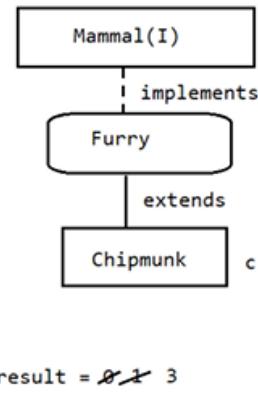
```

public class IsItFurry {
    static interface Mammal { }
    static class Furry implements Mammal { }
    static class Chipmunk extends Furry { }
    public static void main(String[] args) {

        Chipmunk c = new Chipmunk();
        Mammal m = c;
        Furry f = c;
        int result = 0;
        if (c instanceof Mammal) result += 1;
        if (c instanceof Furry) result += 2;
        if (null instanceof Chipmunk) result += 4;
        System.out.println(result);

    }
}

```



result = ~~0, 2~~ 3

58)05-12-22

(multi threading 4)(0:9:00)

When we get InterruptedException ?

=When waiting and sleeping thread is interrupted

Interupting a Thread

=====

public void interrupt()

=> If thread is in sleeping state or in waiting state we can interupt a thread.

eg#1.

```

class MyThread extends Thread{
    @Override
    public void run(){
        try{
            for (int i=1;i<=10;i++){
                System.out.println("I am lazy thread"+i);
                Thread.sleep(2000);
            }
        }
    }
}

```

```
        catch (InterruptedException e){
            System.out.println("I got interrupted");
        }
    }
}

public class Test3 {
    public static void main(String... args) throws InterruptedException{
        MyThread t=new MyThread();
        t.start();
        t.interrupt(); //line-n1
        System.out.println("End of Main...");
    }
}
```

Scenario:: If a comment line-n1

2 thread

a. Main Thread

End of Main...

b. Child Thread

I am lazy thread

.....

.....

Scneario:: If t.interrupt() then

2 thread

a. Main Thread

main thread

b. Child Thread

I am lazy thread

I got interrupted

eg#2.

```
class MyThread extends Thread{
    @Override
    public void run(){
        for (int i=1;i<=10000 ;i++ ){
            System.out.println("I am lazy thread : "+i);
        }
        System.out.println("I am entering into sleeping state");
        try
        {
        }
        Thread.sleep(2000);
        catch (InterruptedException ie){
            ie.printStackTrace();
        }
    }
}
```

```
}

}

public class TestApp {
    public static void main(String[] args) throws InterruptedException {
        MyThread t = new MyThread();
        t.start();
        t.interrupt(); //line-n1
        System.out.println("main thread");
    }
}
```

line-n1 is commented then no problem

line-n1 is not commented, then interrupt() will wait till the Thread enters into waiting state/sleeping state.

Note::

If thread is interrupting another thread, but target thread is not in waiting state/sleeping state then there would be no exception.

interrupt() call be waiting till the target thread enters into waiting state/sleeping state so this call wont be wasted.

once the target thread enters into waiting state/sleeping state then interrupt() will interrupt and it causes the exception.

interrupt() call will be wasted only if the Thread does not enters into waiting state/sleeping state.

yield() join() sleep()

=====

1) Purpose

yield()

To pause current executing Thread for giving the chance of remaining waiting Threads of same priority.

join()

If a Thread wants to wait until completing some other Thread then we should go for join.

sleep()

If a Thread don't want to perform any operation for a particular amount of time then we should go for sleep() method.

2) Is it static

yield() yes

join() no

sleep() yes

3) Is it final?

yield() no

join() yes

sleep() no

4) Is it overloaded?

yield() no

```
join() yes
sleep() yes
5) Is it throws IE?
    yield() no
join() yes
sleep() yes
6) Is it native method?
    yield() yes
join() no
sleep()
sleep(long ms) -->native
    sleep(long ms,int ns) -->non-native
```

Note::using lambda expression

```
Runnable r = ()-> {
```

```
Thread t = new Thread(r);
```

```
t.start();
```

using anonymous inner class

```
=====
```

```
new Thread(new Runnable(){
@Override
public void run(){
for (int i = 1;i<=5 ; i++)
{
System.out.println("child thread");
}
};
for (int i = 1;i<=5 ;i++ )
{
System.out.println("child thread");
}
}
}
);
.start();
```

synchronization

```
=====
```

1. synchronized is a keyword applicable only for methods and blocks
2. if we declare a method/block as synchronized then at a time only one thread can execute that method/block on that object.
3. The main advantage of synchronized keyword is we can resolve data inconsistency problems.
4. But the main disadvantage of synchronized keyword is it increases waiting time of the Thread and effects performance of the system.

5. Hence if there is no specific requirement then never recommended to use synchronized keyword.

6. Internally synchronization concept is implemented by using lock concept.

```
class X{  
    synchronized void m1(){}
    synchronized void m2(){}
    void m3(){}
}
```

KeyPoints

=====

1. if t1 thread invokes m1() then on the Object X lock will applied.

2. if t2 thread invokes m2() then m2() can't be called because lock of X object is with m1.

3. if t3 thread invokes m3() then execution will happen becoz m3() is non synchronized.

Lock concept is applied at the Object level not at the method level.

7. Every object in java has a unique lock. Whenever we are using synchronized keyword then only lock concept will come into the picture.

8. If a Thread wants to execute any synchronized method on the given object 1st it has to get the lock of that object.

Once a Thread got the lock of that object then it's allow to execute any synchronized method on that object.

If the synchronized method execution completes then automatically Thread releases lock.

9. While a Thread executing any synchronized method the remaining Threads are not allowed execute any synchronized

method on that object simultaneously. But remaining Threads are allowed to execute any non-synchronized method

simultaneously. [lock concept is implemented based on object but not based on method].

Note::

Every object will have 2 area[Synchronized area and NonSynchronized area]

Synchronized Area => write the code only to perform update,insert,delete

NonSynchronized Area => write the code only to perform select operation

```
class ReservationApp{
    checkAvailability(){
        //perform read operation
    }
    synchronized bookTicket(){
        //perform update operation
    }
}
eg#1.
```

```

class Display{
public void wish(String name){
for (int i=1;i<=10 ;i++ )
{
System.out.print("Good Morning: ");
try{
Thread.sleep(2000);
}
catch (InterruptedException e){
}
}
System.out.println(name);
}
}

class MyThread extends Thread{
Display d;
String name;
MyThread(Display d,String name){
this.d=d;
this.name=name;
}
@Override
public void run(){
d.wish(name);
}
}

public class Test3 {
public static void main(String... args){
Display d=new Display();
MyThread t1= new MyThread(d,"dhoni");
MyThread t2= new MyThread(d,"yuv");
t1.start();
t2.start();
}
}

Output:: As noticed below the output is irregular becoz at a time on a resource
called wish()
2 threads are acting simultaneously.
3 Threads
a. Main Thread
b. Child Thread-1
c. Child Thread-2
GoodMorning :GoodMorning : ...
....
```

```
....  
....  
....  
....  
eg#2.  
class Display{  
public synchronized void wish(String name){  
for (int i=1;i<=10 ;i++ )  
{  
System.out.print("Good Morning: ");  
try{  
}  
Thread.sleep(2000);  
catch (InterruptedException e){  
}  
System.out.println(name);  
}  
}  
}  
}  
class MyThread extends Thread{  
Display d;  
String name;  
MyThread(Display d,String name){  
this.d=d;  
this.name=name;  
}  
@Override  
public void run(){  
d.wish(name);  
}  
}  
public class Test3 {  
public static void main(String... args)throws InterruptedException{  
Display d=new Display();  
MyThread t1= new MyThread(d,"dhoni");  
MyThread t2= new MyThread(d,"yuvi");  
t1.start();  
t2.start();  
}  
}  
Ouput::  
3 Threads  
a. Main Thread  
b. Child Thread-1
```

```
GoodMorning:dhoni  
GoodMorning:dhoni
```

.....

.....

.....

c. Child Thread-2

```
GoodMorning:yuv  
GoodMorning:yuv
```

.....

.....

.....

Note::

As noticed above there are 2 threads which are trying to operate on single object called

"Display" we need synchronization to resolve the problem of "Data inconsistency".

casestudy::

```
Display d1=new Display();  
Display d2=new Display();  
MyThread t1=new MyThread(d1,"yuvraj");  
MyThread t2=new MyThread(d2,"dhoni");  
t1.start();  
t2.start();
```

In the above case we get irregular output, because two different object and since the method

is synchronized lock is applied w.r.t object and both the threads will start simultaneously on different java objects due to which the output is "irregular".

Conclusion :

If multiple threads are operating on multiple objects then there is no impact of Syncronization.

If multiple threads are operating on same java objects then syncronized concept is required(applicable).

classlevel lock

=====

1. Every class in java has a unique level lock.

2. If a thread wants to execute static synchronized method then the thread requires

"class level lock".

3. While a Thread executing any static synchronized method the remaining Threads are not allow

to execute any static synchronized method of that class simultaneously.

4. But remaining Threads are allowed to execute normal synchronized methods, normal static methods, and normal instance

methods simultaneously.

5. Class level lock and object lock both are different and there is no relationship

between these two.

eg::

```
class X{  
    static synchronized m1(){}//class level lock  
    static synchronized m2(){}  
    static m3(){}//no lock required  
    synchronized m4(){}//object level lock  
    m5(){}//no lock required  
}
```

t1=> m1() => class level lock applied and chance is given

t2=> m2() => enter into waiting state

t3=> m3() => gets a chance for execution without any lock

t4=> m4() => object level lock applied and chance is given

t5=> m5() => gets a chance for execution without any lock

eg#1.

```
class Display{  
    public synchronized void displayNumbers(){  
        for (int i=1;i<=10 ;i++ )  
        {  
            System.out.print(i);  
            try{  
                Thread.sleep(2000);  
            }  
            catch (InterruptedException e){  
            }  
        }  
        public synchronized void displayCharacters(){  
            for (int i=65;i<=75 ;i++ )  
            {  
                System.out.print((char)i);  
                try{  
                    Thread.sleep(2000);  
                }  
                catch (InterruptedException e){  
                }  
            }  
            class MyThread1 extends Thread{  
                Display d;
```

```
MyThread1(Display d){  
    this.d=d;  
}  
@Override  
public void run(){  
    d.displayNumbers();  
}  
}  
  
class MyThread2 extends Thread{  
    Display d;  
    MyThread2(Display d){  
        this.d=d;  
    }  
    @Override  
    public void run(){  
        d.displayCharacters();  
    }  
}  
  
public class Test3 {  
    public static void main(String... args){  
        Display d1=new Display();  
        MyThread1 t1= new MyThread1(d1);  
        MyThread2 t2= new MyThread2(d1);  
        t1.start();  
        t2.start();  
    }  
}  
  
Output::  
3 Threads  
a.MainThread  
b.userdefinedThread  
displayCharacters()  
c.userdefinedThread  
displayNumbers()  
Synchronized block  
=====  
synchronized void m1(){  
    ...  
    ...  
    ...  
    ...  
    ...  
=====  
=====
```

```
=====
=====
...
...
...
...
...
}
}
```

if few lines of code is required to get synchronized then it is not recommended to make method only as synchronized.

If we do this then for threads performance will be low, to resolve this problem we use "synchronized block", due to synchronized block performance will be improved.

Case Study

```
=====
```

If a thread got a lock of current object, then it is allowed to execute that block

a.

```
synchronized(this){
```

```
....
```

```
....
```

```
....
```

```
}
```

To get a lock of particular object:: B

b.

```
synchronized(B){
```

```
....
```

```
....
```

```
....
```

```
}
```

If a thread got a lock of particular object B, then it is allowed to execute that block.

c. To get class level lock we have to declare synchronized block as follow

```
synchronized(Display.class){
```

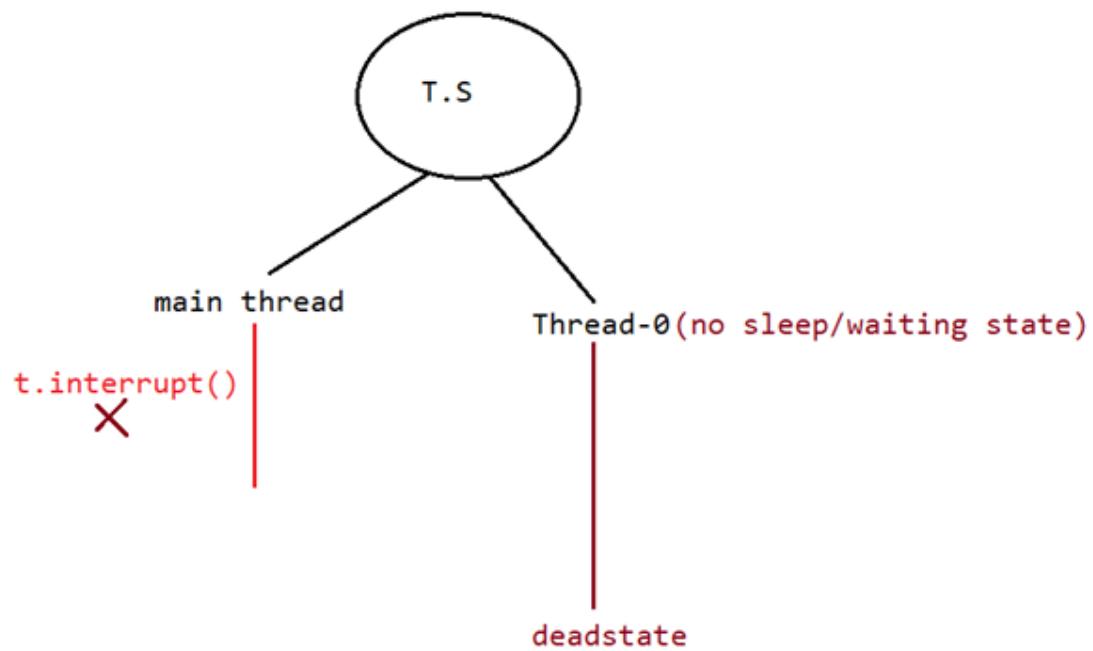
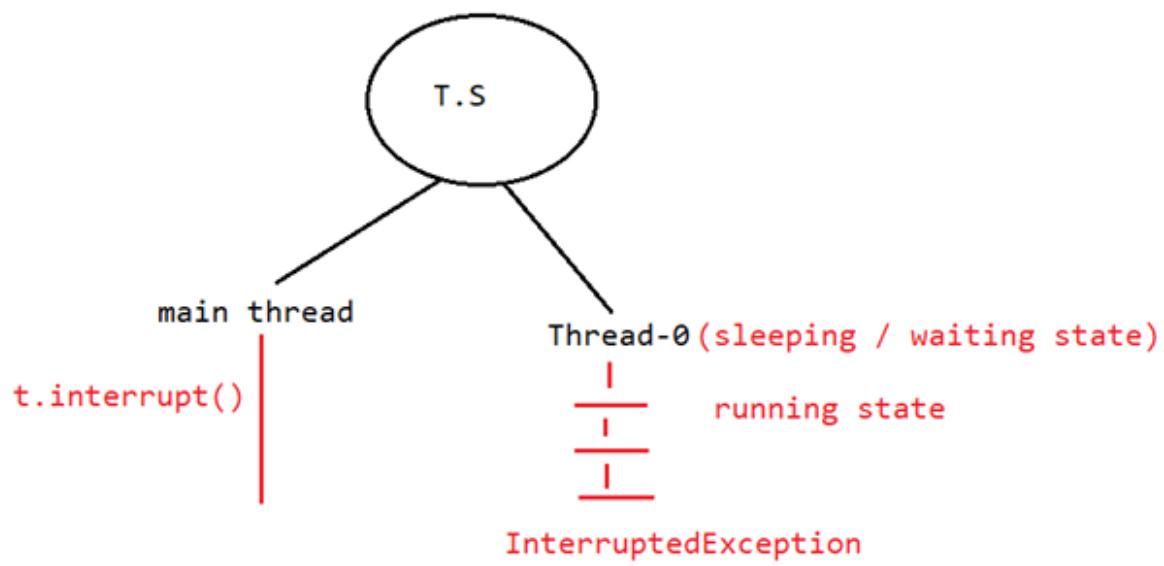
```
....
```

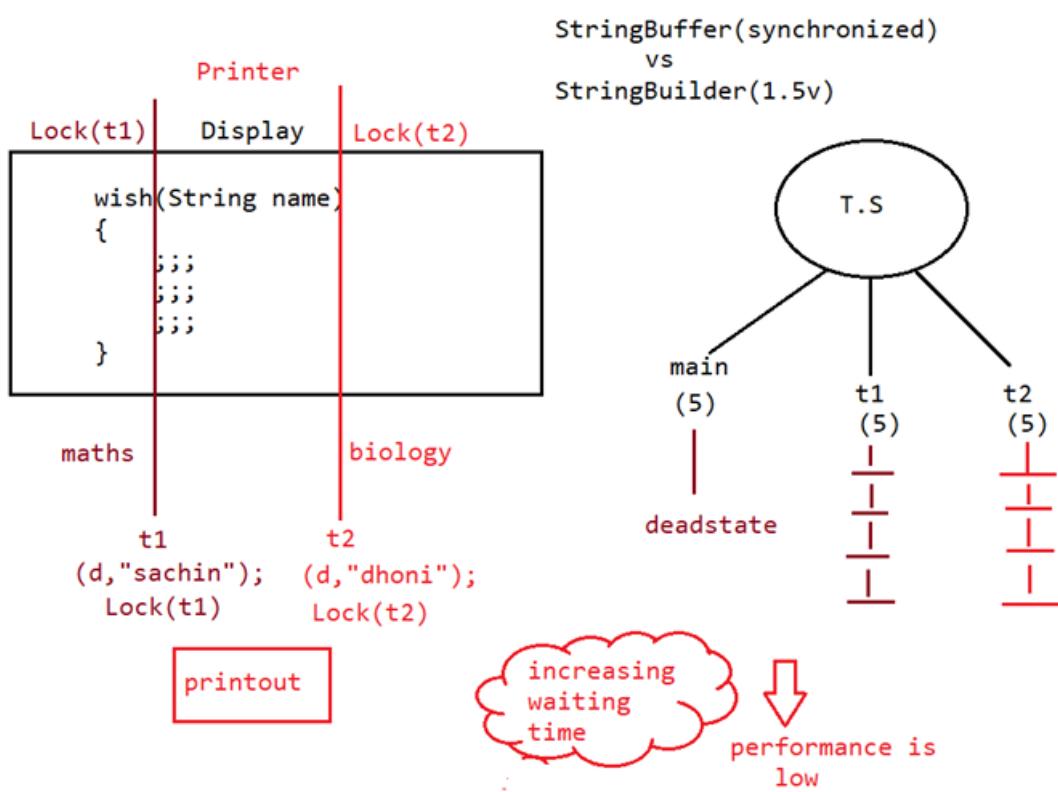
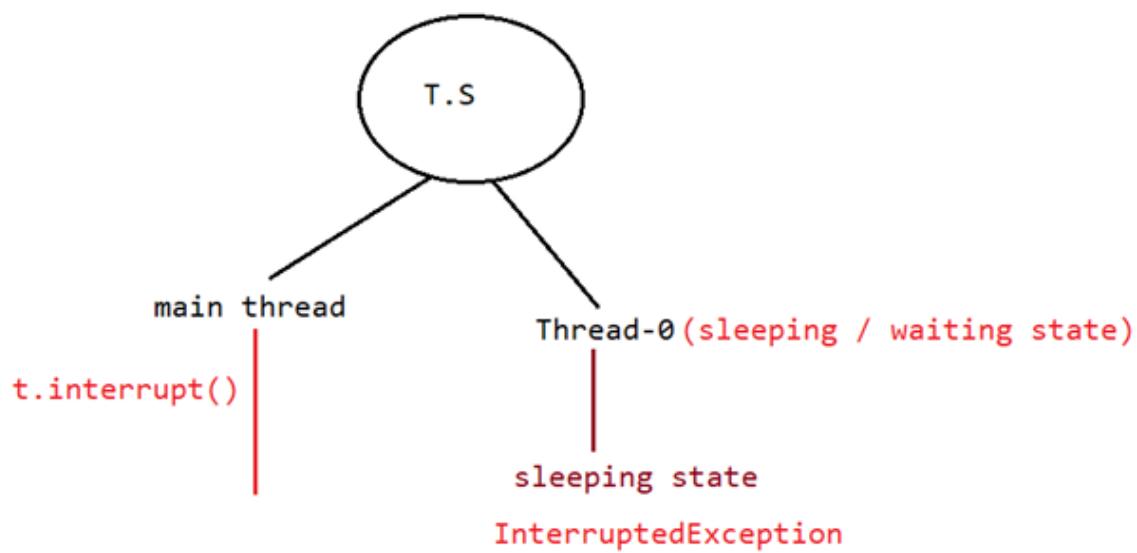
```
....
```

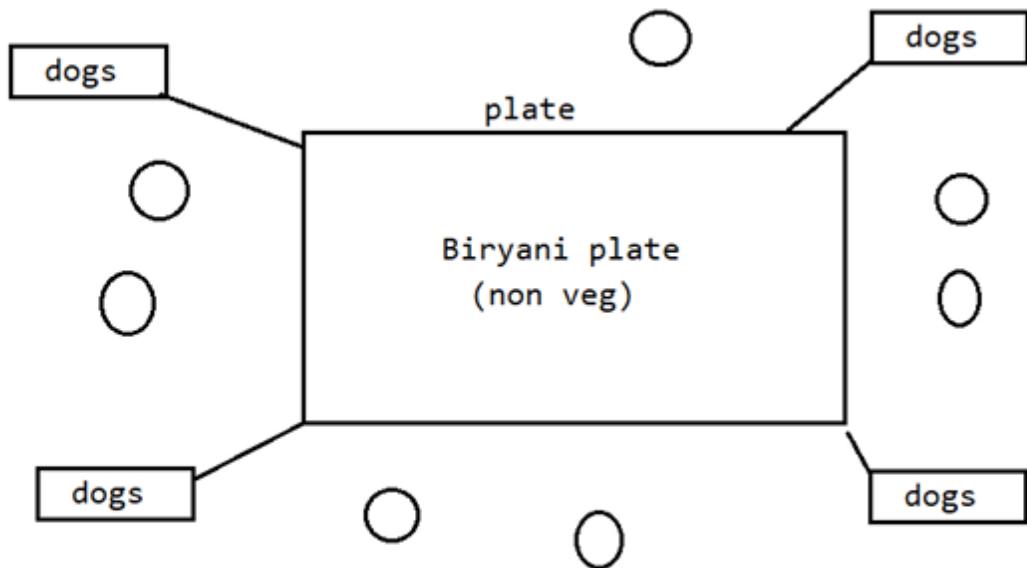
```
....
```

```
}
```

If a thread gets class level lock, then it is allowed to execute that block







BiryaniInconsistencyProblem

```

class X{
    synchronized void m1(){}
    synchronized void m2(){}
    void m3(){}
}

Lock(t1)
X
t1
m1()
synchronized
t2
m2()
synchronized
t3
m3()
"lock is not required"

```

t1 t2 t3 t4

```

class RailwayReservationApp{

    void searchTicket(){
        //non-synchronized region
    }

    void bookTicket(){
        //synchronized region
    }
}

```

t1 (operated by only one thread)

~~synchronized~~

```

class X{
    static synchronized m1(){}
    static synchronized m2(){}

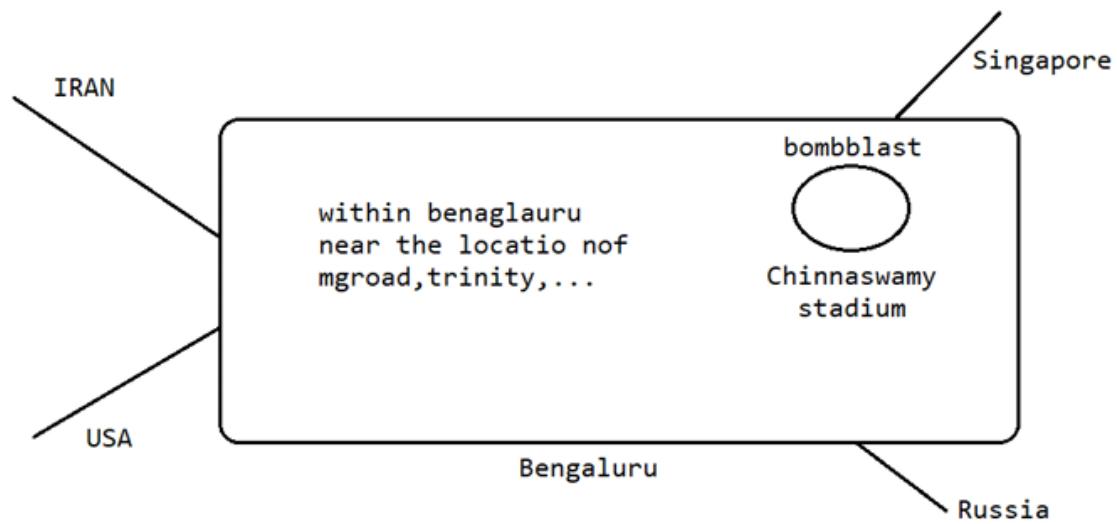
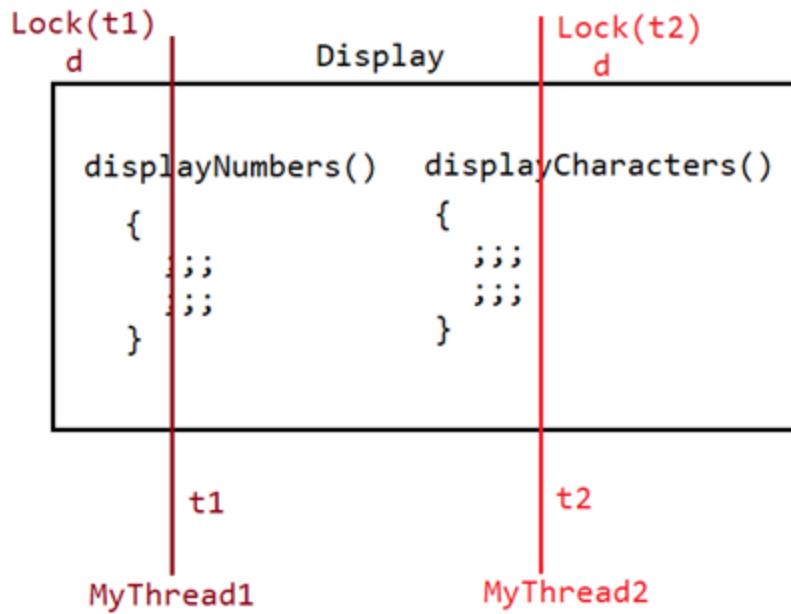
    static m3(){}
    synchronized m4(){}
    m5(){}
}

```

Lock(t1)

X.class

Method	Lock Holder	Thread	Annotation
m1()	t1	t1	static synchronized
m2()	t2	t2	static synchronized
m3()		t3	no lock is required
m4()		t4	no lock is required
m5()		t5	no lock is required



32)Code Snippets

()

Q>

1. Which ONE of the following statements is TRUE?
 - A. You cannot extend a concrete class and declare that derived class abstract
 - B. You cannot extend an abstract class from another abstract class
 - C. An abstract class must declare at least one abstract method in it
 - D. You can create an instance of a concrete subclass of an abstract class but cannot create an instance of an abstract class itself.

Answer: D

Q>Choose the correct answer based on the following class definition:

```
public abstract final class Shape { }
```

- A. Compiler error: a class must not be empty
- B. Compiler error: illegal combination of modifiers abstract and final
- C. Compiler error: an abstract class must declare at least one abstract method
- D. No compiler error: this class definition is fine and will compile successfully

Answer: B

Q>Choose the best option based on this program:

```
class Shape {  
    public Shape() {  
        System.out.println("Shape constructor");  
    }  
}  
  
public class Color {  
    public Color() {  
        System.out.println("Color constructor");  
    }  
}  
  
class TestColor {  
    public static void main(String []args) {  
        Shape.Color black = new Shape().Color(); // #1  
    }  
}
```

- A. Compiler error: the method Color() is undefined for the type Shape
- B. Compiler error: invalid inner class
- C. Works fine: Shape constructor, Color constructor
- D. Works fine: Color constructor, Shape constructor

Answer: A

Q>

```
class Shape {  
    private boolean isDisplayed;  
    protected int canvasID;  
    public Shape() {
```

```

isDisplayed = false;
canvasID = 0;
}
public class Color {
public void display() {
}
}
}
}
System.out.println("isDisplayed: "+isDisplayed);
System.out.println("canvasID: "+canvasID);
class TestColor {
public static void main(String []args) {
Shape.Color black = new Shape().new Color();
black.display();
}
}

```

- A. Compiler error: an inner class can only access public members of the outer class
- B. Compiler error: an inner class cannot access private members of the outer class
- C. Runs and prints this output:

isDisplayed: false
 canvasID: 0

- D. Compiles fine but crashes with a runtime exception

Answer: C

Q>

Determine the behavior of this program:

```

interface DoNothing {
default void doNothing() { System.out.println("doNothing"); } //from jdk1.8
inside an interface we can have concrete methods also.
}
@FunctionalInterface
interface DontDoAnything extends DoNothing {
@Override
abstract void doNothing();
}
class LambdaTest {
public static void main(String []args) {
DontDoAnything beldle = () -> System.out.println("be idle");
beldle.doNothing();
}
}

```

- A. This program results in a compiler error for DontDoAnything interface: cannot override default method to be an abstract method
- B. This program results in a compiler error: DontDoAnything is not a functional interface

C. This program prints: doNothing

D. This program prints: be idle

Answer: D

Q>

Determine the behavior of this program:

```
interface BaseInterface {  
    default void foo() { System.out.println("BaseInterface's foo"); }  
}  
  
interface DerivedInterface extends BaseInterface {  
    default void foo() { System.out.println("DerivedInterface's foo"); }  
}  
  
interface AnotherInterface {  
    public static void foo() { System.out.println("AnotherInterface's foo"); }  
}  
  
public class MultipleInheritance implements DerivedInterface, AnotherInterface {  
    public static void main(String []args) {  
        new MultipleInheritance().foo();  
    }  
}
```

A. This program will result in a compiler error: Redundant method definition for function foo

B. This program will result in a compiler error in MultipleInheritance class:

Ambiguous call to function foo

C. The program prints: DerivedInterface's foo

D. The program prints: AnotherInterface's foo

Answer: C

Q>

Determine the behavior of this program:

```
class LambdaFunctionTest {  
    @FunctionalInterface  
    interface LambdaFunction {  
        int apply(int j);  
        boolean equals(java.lang.Object arg0);  
    }  
    public static void main(String []args) {  
        LambdaFunction lambdaFunction = i -> i * i; // #1  
        System.out.println(lambdaFunction.apply(10));  
    }  
}
```

A. This program results in a compiler error: interfaces cannot be defined inside classes

B. This program results in a compiler error: @FunctionalInterface used for LambdaFunction that defines two abstract methods

C. This program results in a compiler error in code marked with #1: syntax error

D. This program compiles without errors, and when run, it prints 100 in console

Answer: D

59)03-12-22 dbt

60)07-12-22

(multi threading 5)

Bank level project works with the records

```
//package mtSyn;
class Display{
    public void wish(String name) {
        ;;;;;// lakhs of lines of code
        //object level lock
        synchronized(this){//whichever thread gets lock of current thread
that executes
            System.out.println("Thread Which is getting locked :
"+Thread.currentThread().getName());
            for(int i=0;i<5;i++) {
                System.out.print("Good Morning : ");
                try {
                    Thread.sleep(1000);
                } catch (InterruptedException e) {}
                // TODO: handle exception
                System.out.println(name);
            }
            System.out.println("Thread got released :
"+Thread.currentThread().getName());
        }
    }
}
class MyThread extends Thread
{
    Display d;
```

```
String name;

MyThread(Display d, String name) {
    this.d = d;
    this.name = name;
}

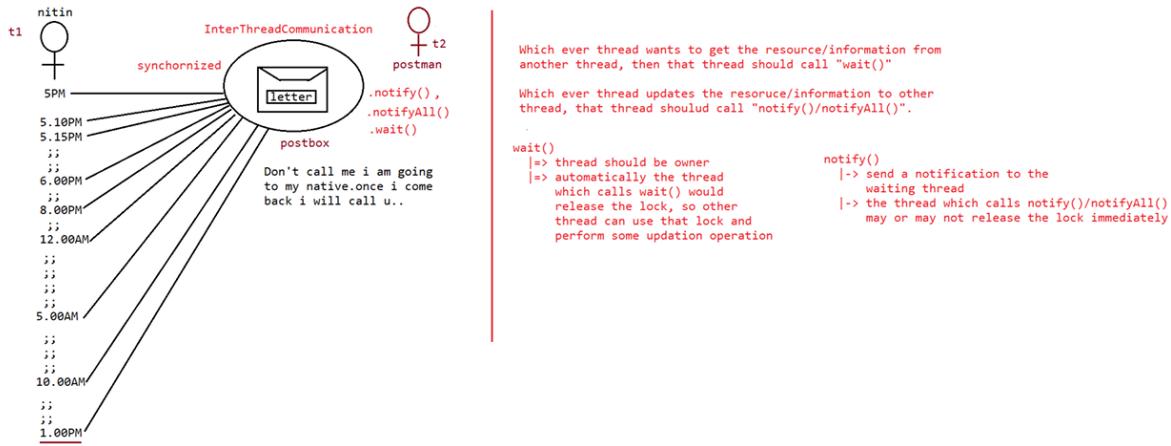
@Override
public void run() {
    d.wish(name);
}
}

public class MtSyn{
    public static void main(String arg[]){
        Display d = new Display();

        MyThread t1 = new MyThread(d, "sid");
        MyThread t2 = new MyThread(d, "sidduGanesh");

        t1.setName("sid");
        t2.setName("sidduganesh");

        t1.start();
        t2.start();
    }
}
```



synchronized block

=====

eg:1

eg#1.

```
class Display{
    public void wish(String name){
        ;;;;;;;;;;; //1-lakh lines of code
        synchronized(this){
            for (int i=1;i<=10;i++ )
            {
                System.out.print("Good morning:");
                try{
                    Thread.sleep(2000);
                }
                catch (InterruptedException e){}
                System.out.println(name);
            }
        }
        ;;;;;;;;;;;//1-lakh lines of code
    }
}

class MyThread extends Thread{
    Display d;
    String name;
    MyThread(Display d,String name){
        this.d=d;
    }
}
```

```

this.name=name;
}
public void run(){
d.wish(name);
}
}

class Test {
public static void main(String[] args) {
Display d=new Display();
MyThread t1=new MyThread(d,"dhoni");
MyThread t2=new MyThread(d,"yuvi");
t1.start();
t2.start();
}
}

eg#2.

class Display{
public void wish(String name){
;;;;;;;;;; //1-lakh lines of code
synchronized(this){
for (int i=1;i<=10;i++ )
{
System.out.print("Good morning:");
try{
Thread.sleep(2000);
}
catch (InterruptedException e){}
System.out.println(name);
}
}
}
}

class MyThread extends Thread{
Display d;
String name;
MyThread(Display d,String name){

```

```

this.d=d;
this.name=name;
}
public void run(){
d.wish(name);
}
}

public class Test {
public static void main(String[] args) {
Display d1=new Display();
Display d2=new Display();
MyThread t1=new MyThread(d1,"dhoni");
MyThread t2=new MyThread(d2,"yuvি");
t1.start();
t2.start();
}
}

```

Output::Irregular output becoz two object and two threads acting on two different objects

eg#3.

```

class Display{
public void wish(String name){
;;;;;;;;;; //1-lakh lines of code
synchronized(Display.class){
for (int i=1;i<=10;i++ )
{
System.out.print("Good morning:");
try{
Thread.sleep(2000);
}
catch (InterruptedException e){}
System.out.println(name);
}
}
}
}

;;;;;;;;;;//1-lakh lines of code

```

```

class MyThread extends Thread{
Display d;
String name;
MyThread(Display d,String name){
this.d=d;
}
this.name=name;
public void run(){
d.wish(name);
}
}
public class Test {
public static void main(String[] args) {
Display d1=new Display();
Display d2=new Display();
MyThread t1=new MyThread(d1,"dhoni");
MyThread t2=new MyThread(d2,"yuvi");
t1.start();
t2.start();
}
}

```

Note:: 2 object, 2 thread, but the thread which gets a chance applied class level lock so output is regular.

Note:: lock concept applicable only for objects and class types, but not for primitive types. if we try to do it would

result in compile time error saying "unexpected type".

eg:: int x=10;
synchronized(x){//CE: unexpected type found:int required:reference
}

```

>>javap java.lang.Object
InterThreadCommunication(remember postbox example)
=====

```

Two threads can communicate each other with the help of
a. notify()
b. notifyAll()
c. wait()

`notify()=>` Thread which is performing updation should call `notify()`, so the waiting thread will get notification so it will continue with its execution with the updated items.

`wait() =>` Thread which is expecting notification/updation should call `wait()`, immediately the Thread will enter into waiting state.

If a thread wants to call `wait(),notify()/notifyall()` then compulsorily the thread should be

the owner of the object otherwise it would result in "IllegalMonitorStateException".

We say thread to be owner of that object if thread has lock of that object.

It means these methods are part of synchronized block or synchronized method, if we try to use

outside synchronized area then it would result in `RunTimeException` called "IllegalMonitorStateException".

if a thread calls `wait()` on any object, then first it immediately releases the lock on that object and it enters into waiting state.

if a thread calls `notify()` on any object, then he may or may not release the lock on that object immediately.

Except `wait(),notify(),notifyAll()` lock can't be released by other methods.

Note::

`yield(),sleep(),join() =>` can't release the lock.

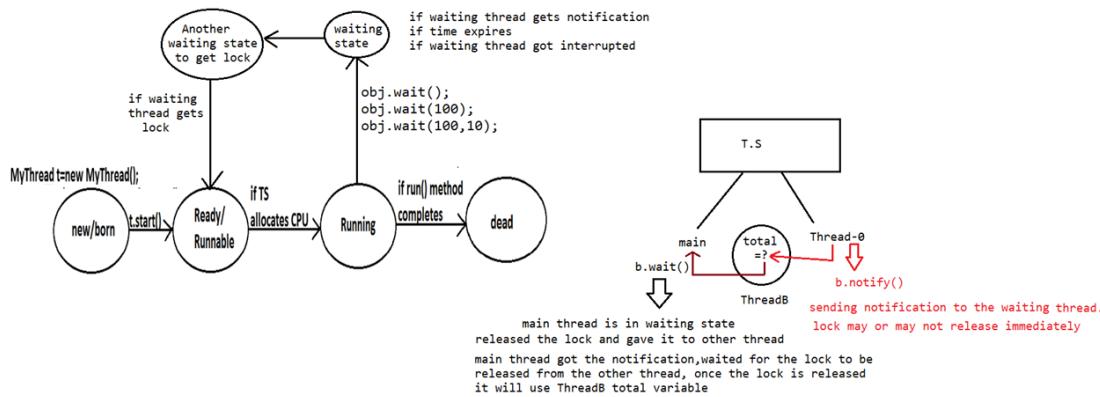
`wait(),notify(),notifyAll() =>` will release the lock, otherwise interthread communication can't happen.

Once a Thread calls `wait()`, `notify()`, `notifyAll()` methods on any object then it releases the lock of that particular object but not all locks it has.

Method prototype of `wait(),notify(),notifyAll()`

1. `public final void wait()throws InterruptedException`
2. `public final native void wait(long ms) throws InterruptedException`
3. `public final void wait(long ms,int ns) throws InterruptedException`
4. `public final native void notify()`
5. `public final void notifyAll()`

Life Cycle of Thread



Interview Question

=====
Method like `wait()`,`notify()`,`notifyAll()` are present inside Object class, y not in Thread class?

Thread will call `wait()`,`notify()`,`notifyAll()` on Objects like PostBox, Stack, Customer, Student,....

=> `obj.wait()`,`obj.notify()`,`obj.notifyAll()`

These methods should be available for every object in java, if the method has to be available for every object in java then those methods should come from "Object" class.

Program

=====

eg#1.

```
class ThreadB extends Thread{
int total =0;
@Override
public void run(){
for (int i=0;i<=100 ; i++){
total+=i;
}
}
}

public class Test {
public static void main(String[] args)throws InterruptedException {
```

```
ThreadB b=new ThreadB();
b.start();
stmt-1;
System.out.println(b.total);
}
}
```

A. stmt-1

if i replace with Thread.sleep(10000) then thread will enter into waiting statement

but within 1ns only the updation value is ready.

with in 10 sec if the updation is not ready, then we should not use Thread.sleep(10000)

B. stmt-2

if i replace with b.join(), then main thread will enter into waiting state,then child will

execute for loop,till then main thread has to wait.

main thread is waiting for updation result.

```
for (int i=0;i<=100 ; i++){
total+=i;
}
```

//1cr lines of code is available

main thread has to wait till 1 cr lines of code,y main thread should wait for the

eg#2.

complete code to finish.

```
class ThreadB extends Thread{
int total =0;
@Override
public void run(){
synchronized(this){
System.out.println("Child thread started calculation");
for (int i=0;i<=100 ; i++){
total+=i;
}
System.out.println("Child thread trying to give notification");
this.notify();
}
}
```

```

    }
}

public class Test {
    public static void main(String[] args) throws InterruptedException {
        ThreadB b = new ThreadB();
        b.start();
        Thread.sleep(10000); // 10sec
        synchronized(b) {
            System.out.println("Main thread is calling wait on B object");
            b.wait();
            System.out.println("Main thread got notification");
            System.out.println(b.total);
        }
    }
}

```

Output

=====

Child thread started calculation

Child thread trying to give notification

Main thread is calling wait on B object

becoz of Thread.sleep(10000) main thread will never get notification.

eg#3.

```

class ThreadB extends Thread{
    int total = 0;
    @Override
    public void run(){
        synchronized(this){
            System.out.println("Child thread started calculation");//step-2
            for (int i=0;i<=100 ; i++){
                total+=i;
            }
            System.out.println("Child thread trying to give notification");//step-3
            this.notify();
        }
    }
}

```

```
public class Test {  
    public static void main(String[] args) throws InterruptedException {  
        ThreadB b = new ThreadB();  
        b.start();  
        synchronized(b){  
            System.out.println("Main thread is calling wait on B object");//step-1  
            b.wait(10000);//10sec  
            System.out.println("Main thread got notification");//step-4  
            System.out.println(b.total);  
        }  
    }  
}  
  
Output  
=====
```

Child thread started calculation
Child thread trying to give notification
Main thread is calling wait on B object for 10sec
Main thread got notification
5050

```
Thread.activeCount();
```

ProducerConsumer Problem

```
=====
```

Producer => produce the item and update in the Queue

Consumer => consume the item from the Queue

```
class Producer extends Thread{
```

```
    Producer(){
```

```
        synchronized(q){
```

```
            produce the item and update it to queue
```

```
            q.notify();
```

```
        }
```

```
    }
```

```
}
```

```
class Consumer extends Thread{
```

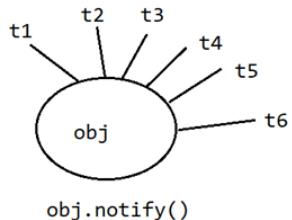
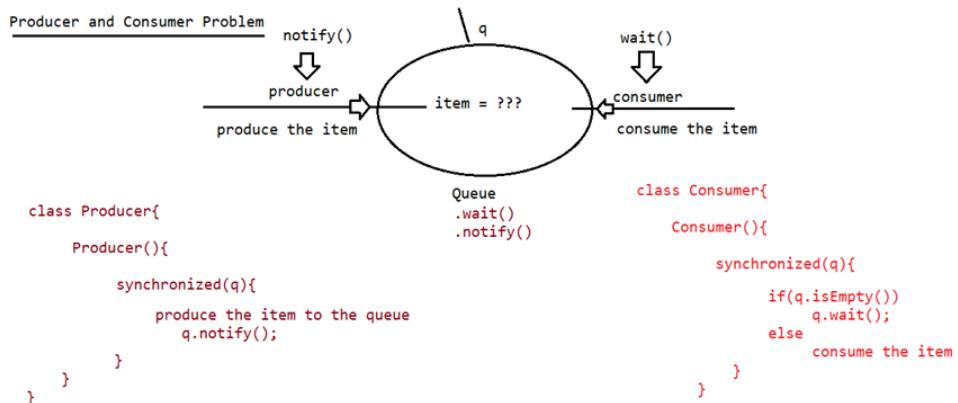
```
    Consumer(){
```

```
        synchronized(q){
```

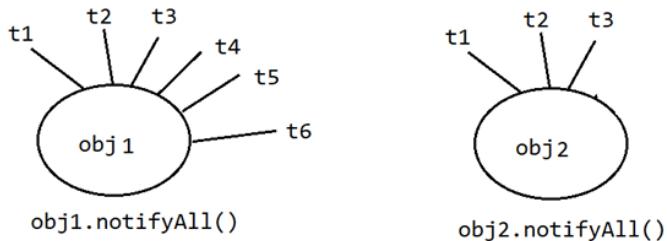
```

if(q is empty){
q.wait();
}else{
}
consume the item from the queue
}
}
}
}

```



notify() is used to give notification to one thread. if there are multiple waiting threads, then only one thread will get a chance, remaining threads should wait for further notification. But which thread will get a chance is totally depends on T.S.



notifyAll() is used to give notification to all waiting threads. All waiting threads will be notified. and it will be executed one by one.

notifyAll() eg is railway train arrival to platform

Difference b/w notify and notifyAll()

`notify()` => To give notification only for one waiting thread
`notifyAll()` => To give notification for many waiting threads
=> We can use `notify()` method to give notification for only one Thread. If multiple Threads are waiting then only one Thread will get the chance and remaining Threads have to wait for further notification.

But which Thread will be notified (inform) we can't expect exactly it depends on JVM.

eg::

```
waiting state
|
|   obj1.wait(); 60 threads are waiting
obj1.notify() |
|
Running state
```

Among 60 threads which thread will get a chance we don't have control over that it is decided

by JVM (threadscheduler).

=> We can use `notifyAll()` method to give the notification for all waiting Threads of particular object.

All waiting Threads will be notified and will be executed one by one, because they required lock.

eg::

```
waiting state
|
|   obj1.wait(); 60 threads are waiting
obj1.notifyAll() |   obj2.wait(); 40 threads are waiting
|
Running state
```

Note: On which object we are calling `wait()`, `notify()` and `notifyAll()` methods that corresponding object lock we have to get but not other object locks.

eg:: main method

```
Stack s1=new Stack();
Stack s2=new Stack();
synchronized(s1){
    s2.wait(); //RE: IllegalMonitorStateException
}
synchronized(s2){
    s2.wait(); //valid
}
```

Question based on lock

- =====
1. If a thread calls wait() immediately it will enter into waiting state without releasing any lock.(false)
 2. If a thread calls wait() it releases the lock of that object but may not immediately (false)
 3. If a thread calls wait() on any object,it releases all locks acquired by that thread and enters into waiting state(false)
 4. If a thread calls wait() on any object,it immediately releases the lock of that particular object and enters into waiting state(true).
 5. If a thread calls notify() on any object,it immediately releases the lock of that particular object(invalid)
 6. If a thread calls notify() on any object,it releases the lock of that object but may not immediately(true)

33)Code Snippets

2:59:00

Q>

```
public interface A111 {  
    String s = "yo";  
    public void method1();  
}  
interface B {  
}  
interface C extends A111, B {  
    public void method1();  
    public void method1(int x);  
}
```

What is the result?

- A. Compilation succeeds.
- B. Compilation fails due to multiple errors.
- C. Compilation fails due to an error only on line 20.
- D. Compilation fails due to an error only on line 21.
- E. Compilation fails due to an error only on line 22.
- F. Compilation fails due to an error only on line 12.

Answer: A

Q>

Given:

```
public class Yikes {  
    public static void go(Long n) {  
        System.out.print("Long ");  
    }  
    public static void go(Short n) {  
        System.out.print("Short ");  
    }  
    public static void go(int n) {  
        System.out.print("int ");  
    }  
    public static void main(String[] args) {  
        short y = 6;  
        long z = 7;  
        go(y); //short----> int  
        go(z); //long =====> Long  
    }  
}
```

What is the result?

- A. int Long
- B. Short Long
- C. Compilation fails.
- D. An exception is thrown at runtime.

Answer: A

Given:

```
class Alpha {  
    public void foo() { System.out.print("Afoo "); }  
}  
public class Beta extends Alpha {  
    public void foo() { System.out.print("Bfoo "); } //overridden method  
    public static void main(String[] args) {  
        Alpha a = new Beta();  
        Beta b = (Beta)a;  
        a.foo();  
    }  
    b.foo();  
}
```

What is the result?

- A. Afoo Afoo
- B. Afoo Bfoo
- C. Bfoo Afoo
- D. Bfoo Bfoo
- E. Compilation fails.
- F. An exception is thrown at runtime.

Answer: D

Q>

Given:

```
1. public class Target {  
2.  
private int i = 0;  
3.  
4.  
5.  
6. }
```

And:

```
public int addOne() {  
return ++i;  
}  
1. public class Client {  
2.  
public static void main(String[] args){  
3.  
4.  
5. }  
System.out.println(new Target().addOne());// 1  
}
```

Which change can you make to Target without affecting Client?

- A. Line 4 of class Target can be changed to return i++; -> result will be 0
- B. Line 2 of class Target can be changed to private int i = 1; -> result will be 2
- C. Line 3 of class Target can be changed to private int addOne(){-> we can't make a call becoz it is private method
- D. Line 2 of class Target can be changed to private Integer i = 0;

Answer: D

Q>

Given:

```
class Animal {  
public String noise() {  
return "peep";  
}  
}  
  
class Dog extends Animal {  
public String noise() {  
return "bark";  
}  
}  
  
class Cat extends Animal {  
public String noise() {  
return "meow";  
}
```

```
}
```

...

```
30. Animal animal = new Dog();  
31. Cat cat = (Cat)animal;  
32. System.out.println(cat.noise());
```

What is the result?

- A. peep
- B. bark
- C. meow
- D. Compilation fails.
- E. An exception is thrown at runtime.

Answer: E

Q>

Given:

```
public class Venus {  
    public static void main(String[] args) {  
    }  
    int[] x = { 1, 2, 3 };  
    int y[] = { 4, 5, 6 };  
    new Venus().go(x, y);  
    void go(int[]... z) {  
        for (int[] a : z)  
    }  
        System.out.print(a[0]);  
    }
```

What is the result?

- A. 1
- B. 12
- C. 14
- D. 123
- E. Compilation fails.
- F. An exception is thrown at runtime.

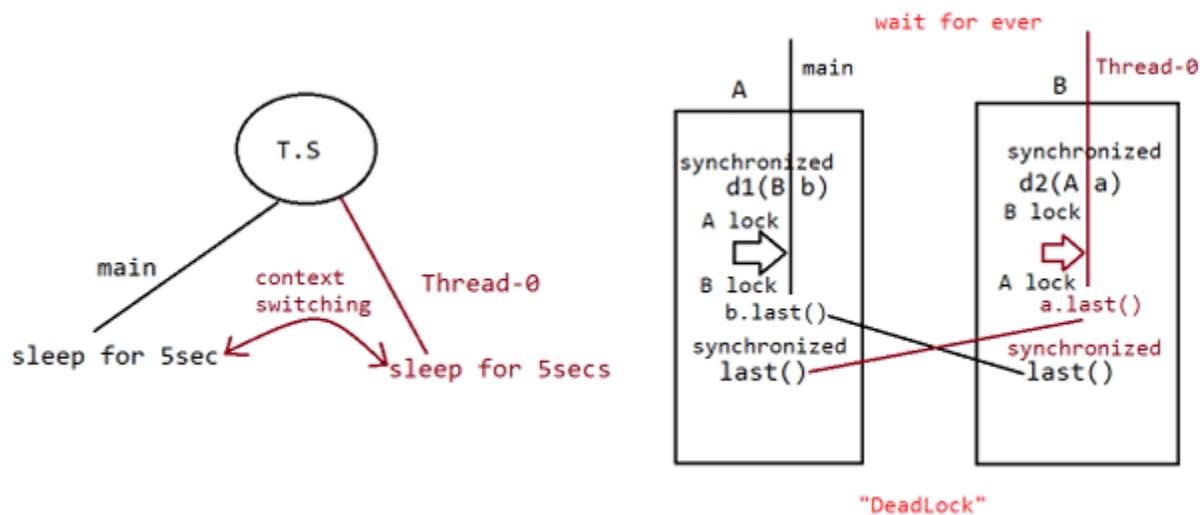
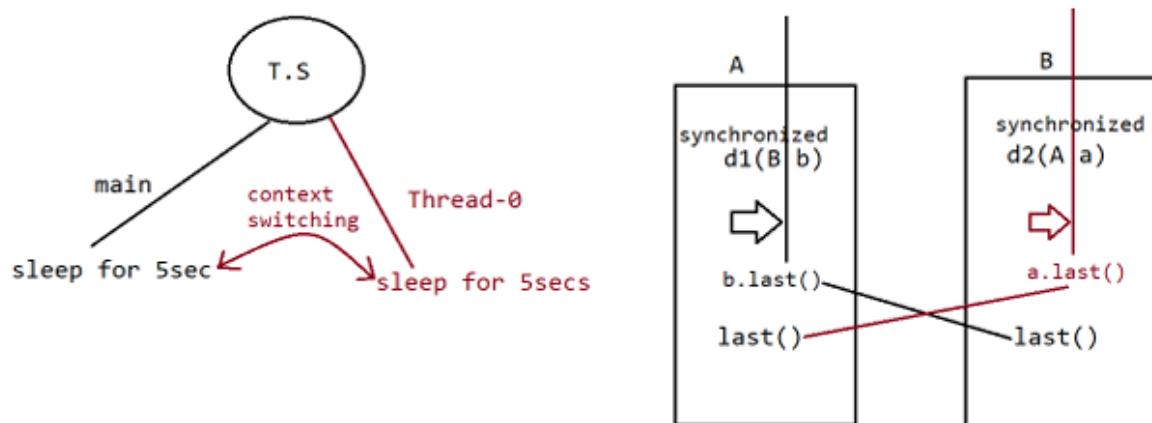
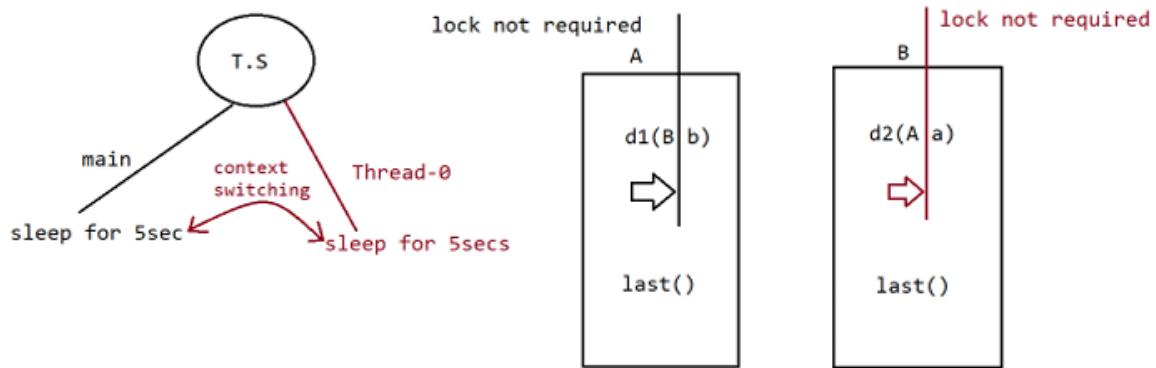
Answer: C

61)08-12-22

(multi threading 6 and summary revision of core java)

DeadLock

=====



If 2 Threads are waiting for each other forever(without end) such type of situation (infinite waiting) is called dead lock.

There are no resolution techniques for dead lock but several prevention(avoidance)

techniques

are possible.

Synchronized keyword is the cause for deadlock hence whenever we are using synchronized keyword

we have to take special care.

eg#1.

```
class A {  
    public void d1(b b){  
        System.out.println("Thread-1 starts execution of d1()");  
        try{  
            Thread.sleep(5000);//5sec  
        }  
        catch (InterruptedException e){  
        }  
        System.out.println("Thread-1 trying to call b last()");  
        b.last();  
    }  
    public void last(){  
        System.out.println("Inside A last() method");  
    }  
}  
  
class B {  
    public void d2(A a){  
    }  
    System.out.println("Thread-2 starts execution of d2()");  
    try{  
        Thread.sleep(5000);//5sec  
    }  
    catch (InterruptedException e){  
    }  
    System.out.println("Thread-2 trying to call A last()");  
    a.last();  
    public void last(){  
        System.out.println("Inside B last() method");  
    }  
}  
  
public class Test extends Thread {
```

```
A a=new A();
B b=new B();
public void m1(){
this.start();
a.d1(b);//executed by main thread
}
public void run(){
b.d2(a);//executed by child thread
}
public static void main(String[] args){
Test t=new Test();
t.m1();
}
}
```

since methods are not synchronized, lock is not required, so no deadlock

Output

Thread-1 starts execution of d1()

Thread-2 starts execution of d2()

Thread-1 trying to call B last()

Inside B last() method

Thread-2 trying to call A last()

Inside A last() method

eg#3.

```
class A extends Thread{
public synchronized void d1(B b){
System.out.println("Thread-1 starts execution of d1()");
try{
Thread.sleep(5000);//5sec
}
catch (InterruptedException e){
}
System.out.println("Thread-1 trying to call B last()");
b.last();
}
public synchronized void last(){
System.out.println("Inside A last() method");
}
}
```

```

}

class B extends Thread{
public synchronized void d2(A a){
System.out.println("Thread-2 starts execution of d2()");
try{
Thread.sleep(5000);//5sec
}
catch (InterruptedException e){
}
System.out.println("Thread-2 trying to call A last()");
a.last();
}
public synchronized void last(){
System.out.println("Inside B last() method");
}
}

public class Test extends Thread {
A a=new A();
B b=new B();
public void m1(){
this.start();
a.d1(b);//line executed by main thread
}
public void run(){
b.d2(a);//line executed by child thread
}
public static void main(String[] args){
Test t=new Test();
t.m1();//main thread s executing
}
}

```

In the above program, there is a possibility of "deadlock".

Output

Thread-1 starts execution of d1()

Thread-2 starts execution of d2()

Thread-1 trying to call B last()

Thread-2 trying to call A last()

//here cursor will be waiting

t1 => starts d1(),since d1() is synchronized and a part of 'A' class so t1 applies lockof(A) and

starts the execution, while executing it encounters Thread.sleep().so T.S gives chance

for t2 thread.

After getting a chance again by TS, it tries to execute b.last.

but lock of b is with t2 thread, so t1 enters into waiting state.

t2=> starts d2(),since d2() is synchronized and a part of 'B' class so t2 applies lockof(B) and

starts the execution, while executing it encounter Thread.sleep(),so TS gives chance again for t1 thread.

After getting a chance again by TS, it tries to execute a.last

but lock of a is with t1 thread, so t2 enters into waiting state.

Since both the threads are in waiting state and it would be waiting for ever,so we say the above pgm would result in "DeadLock".

Note:

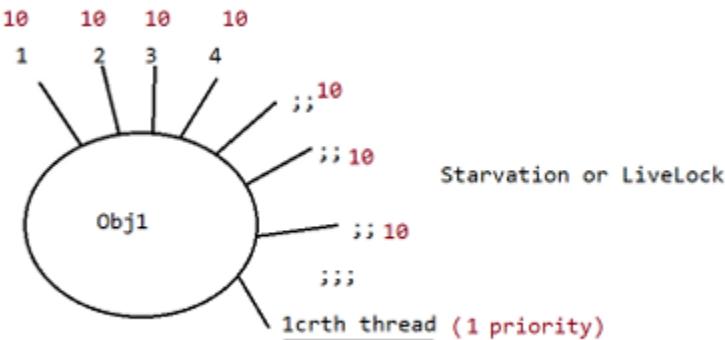
synchronized is the only reason why there is a deadlock,so we should be careful when we use

synchronized keyword,if we remove atleast one synchronized word then the program wont enter into

dead lock.

DeadLock vs starvation

=====



Long waiting of a thread, where waiting never ends is termed "deadlock".

Long waiting of a thread, where waiting ends at certain point is called "starvation".

eg:: Assume we have 1cr threads, where all 1cr threads have priority is 10, but one thread is there

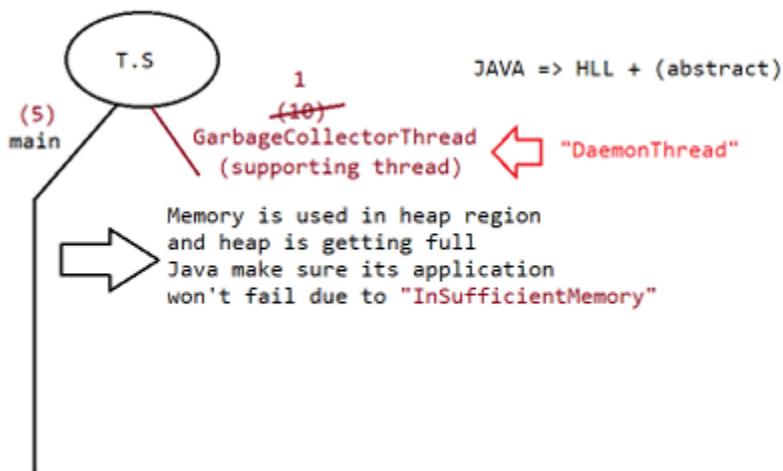
which has priority 1, now the thread with a priority-1 has to wait for long time but still

it gets a chance, but it has to wait for long time, this scenario is called "Starvation".

Note::

Low priority thread has to wait until completing all priority threads but ends at certain

point which is nothing but starvation.



Daemon Threads

The thread which is executing in the background is called "DaemonThread".

eg: AttachListener, SignalDispatcher, GarbageCollector,

remember the example of movie

1. producer
2. director
3. music director
4.
5.
6.

MainObjective of DaemonThread

The main objective of DaemonThread, to provide support for Non-Daemon threads(main thread).

eg:: if main threads runs with low memory then jvm will call GarbageCollector thread, to destroy the useless objects, so that no of bytes of free memory will be improved with this free

memory main thread can continue its execution.

Usually Daemon threads having low priority, but based on our requirement daemon threads can run

with high priority also.

JVM => creates 2 threads

- a. Daemon Thread(priority=1,priority=10)
- b. main (priority=5)

while executing the main code, if there is a shortage of memory then immediately

collector

immediately

jvm will change the priority of Daemon thread to 10, so Garbage activates Daemon thread and it frees the memory after doing it it changes the priority to 1, so main thread it will continue.

How to check whether the Thread is Daemon or not?

public boolean isDaemon() => To check whether the thread is "Daemon"

public void setDaemon(boolean b) throws IllegalThreadStateException

b=> true, means the thread will become Daemon, before starting the Thread we need

to make the thread as "Daemon" otherwise it would result in "IllegalThreadStateException".

What is the default nature of the Thread?

Ans. By default the main thread is "NonDaemon".

for all remaining thread Daemon nature is inherited from Parent to child, that is

if the parent thread is "Daemon" then child thread will become "Daemon" and if the parent

thread is "NonDaemon" then automatically child thread is also "NonDaemon".

Is it possible to change the NonDaemon nature of Main Thread?

Ans. Not possible, becoz the main thread starting is not in our hands, it will be started by

"JVM".

eg::

```
class MyThread extends Thread{}
```

```
public class Test {  
    public static void main(String[] args){  
        System.out.println(Thread.currentThread().isDaemon());//false  
        Thread.currentThread().setDaemon(true);//RE:IllegalThreadStartException  
        MyThread t=new MyThread();  
        System.out.println(t.isDaemon());//false  
        t.setDaemon(true);  
        t.start();  
        System.out.println(t.isDaemon());//true  
    }  
}
```

//Daemon thread is predefined, we don't use much in real time

//we use the already present daemon threads

Note::

Whenever last NonDaemon threads terminates, automatically all Daemon Threads will be terminated

irrespective of their position.

eg:: makeup man in shooting is a DaemonThread

hero is main thread

if hero role is over, then automatically the makeup role is also over automatically.

eg::

```
class MyThread extends Thread{  
    public void run(){  
        for (int i=1;i<=10 ;i++ ){  
            System.out.println("child thread");  
        try{  
            Thread.sleep(2000);//2sec  
        }  
        catch (InterruptedException e){  
            System.out.println(e);  
        }  
    }  
}
```

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

```
MyThread t=new MyThread();
t.setDaemon(true);//stmt-1
t.start();
System.out.println("end of main thread");
    }
}
```

Output:

if we comment stmt-1, then both the threads are NonDaemon threads it would continue with its execution.

end of main thread

child thread

child thread

...

...

...

Output

If we remove comment on stmt-1, then main thread is NonDaemon thread where as user defined thread is DaemonThread, if the main thread finishes the execution then automatically the

DaemonThread also will finish the execution

(1:15:00 – 1:19:00 dbt)

REVISION OF CORE JAVA by naveen reddy

62)09-12-22

(collection 1)

Collection FrameWork | API

=====

Joshua created an hierarchy consists of collection of interfaces and classes
Inbuilt methods using data structures.

⇒ He proposed and gave name as Collection Framework for Java 1.2 version
⇒ Sun Microsystems has approved it.

⇒ Collection major of 7 classes and more added later

1. ArrayList
2. LinkedList
3. ArrayDeque
4. PriorityQueue
5. TreeSet
6. HashSet
7. LinkedHashSet

This three are divided as List(I) , Queue(I) , Set(I)

Each of this follow their own specific data structures

Java developers life became easy with collection framework

ArrayList

ArrayList = internally Dynamic Array DataStructure is followed

All this classes are present inside the util package

Internally everything is stored as objects

Size will grow based on data ⇒ Dynamically

It can store Heterogeneous Data.

We can add data at front , middle but not recommended

It is good recommendation to add at rear end.

It is efficient and easy to add at rear end.

Switching is happen to right side and time consuming.

Efficient is not there in switching so don't do at front and middle

ArrayList ⇒ index Based accessing is allowed.

```
//package CollectionAPI;

import java.util.*;

public class LaunchAL1 {
    public static void main(String arg[])
    {
        ArrayList al1 = new ArrayList();
        al1.add(10);
        al1.add(20);
        System.out.println(al1);
```

```

System.out.println("*****");
ArrayList al2 = new ArrayList();
al2.add("sidduganesh");
al2.add(21.9);
al2.add('M');
System.out.println(al2);
System.out.println("*****");

ArrayList al3 = new ArrayList<>();
al3.addAll(al2);
System.out.println(al3);

System.out.println("*****");
ArrayList al4 = new ArrayList<>();
al4.add(23);
al4.add(44);
al4.add(55);
System.out.println("existing data "+al4);

al4.add(2,28);
System.out.println("after adding in 2nd index "+al4);

al4.add(0,1);
System.out.println("after adding in 0nd index "+al4);

al4.add(8);
System.out.println("after adding in rear end "+al4);

}
}

```

- It can store large data
- Both heterogeneous and homogeneous data
- Dynamic array data structure
- Index based accessing allowed
- duplicates allowed
- Efficient insertion at rear end
- It belongs to List()
- Using inbuilt methods operation can be performed.

```

import java.util.ArrayList;

public class A12 {
    public static void main(String args[])
    {
        ArrayList al4 = new ArrayList<>();
        al4.add(23);
        al4.add(44);
        al4.add(55);
        System.out.println("existing data "+al4);

        //boolean res = al4.contains(44);
        System.out.println(al4.contains(44));
        //we use contains whether the control is comming or not

        int index = al4.indexOf(22);
        System.out.println(index);

        System.out.println(al4.isEmpty());
        al4.ensureCapacity(10);
        System.out.println(al4.size());
        al4.trimToSize();
        System.out.println(al4.getClass());

        al4.clear();
        System.out.println(al4);
        System.out.println(al4.size());

        //capacity means how much you can have it
        //size means how much you already utilized it.
    }
}

```

LinkedList

⇒ it follows the doubly linkedlist data structure
 It supports homogeneous and heterogeneous data stored as objects

Ram is the collection of bits

If data is added it can store in disperse locations in memory that are automatically linked

al.add(1, 30); only links will change no shifting of data.

Based on index we can add at front and middle with doubly linked list and it is also efficient.

>When to use Arrays Over the Collections ?

when ever size of the data is known and sure that data is homogeneous

Then you must go with Arrays.

Array is faster than ArrayList.

In array you can store primitive data and object data

In ArrayList and LinkedList duplicates are allowed.

Inbuilt methods that are present in ArrayList

- Doubly linkedlist
- Nodes and links with previous node and next node
- Can perform insertion at any given position
- One collection another collection can add
- Stores data as objects
- It follows two interfaces List(i) and Deque(l)
- addFirst and addLast inbuilt are present.
- Duplicates are allowed
- Index based accessing is allowed

```
import java.util.LinkedList;

public class LL1 {
    public static void main(String arg[])
    {
        LinkedList l11 = new LinkedList<>();
        l11.add(10);
        l11.add("RamChandar");
        l11.add("23.3");
        System.out.println(l11);

        l11.addFirst("sidduGanesh");
        System.out.println(l11);
        l11.addLast("sid");
        System.out.println(l11);
    }
}
```

```
    }  
}
```

```
LinkedList l12 = new LinkedList<>();  
l12.add(223);  
l12.add(435);  
l12.add(345);  
l12.add(223);  
System.out.println(l12.getFirst());  
System.out.println(l12.getLast());  
System.out.println(l12.indexOf(435));  
System.out.println(l12.lastIndexOf(223));  
l12.offerFirst(5); //comes from the deque  
//offer means your wish wheather to take or not  
//offer method is may or may not  
//but addFirst and addLast will add compulsary  
System.out.println(l12);  
  
System.out.println(l12.peekFirst()); //collection is not affected  
System.out.println(l12);  
System.out.println(l12.pollFirst()); //collection is affected  
System.out.println(l12);
```

ArrayDeque

- Internally follows the double ended queue.
- Index based accessing is not allowed.
- Insertion and deletion can happen at front end and rear end
- addFirst and addLast works
- Duplicates are allowed
- Index based accessing are not allowed
- Dequeue()

```
import java.util.ArrayDeque;  
  
public class AD1 {  
    public static void main(String arg[])  
    {
```

```

        ArrayDeque<E> ad = new ArrayDeque();
        ad.add(10);
        ad.add(20);
        ad.add(20);
        System.out.println(ad);
        ad.addFirst(5);
        ad.addLast(100);
        System.out.println(ad);
        System.out.println(ad.peekFirst());
    }
}

```

63)10-12-22 dbt

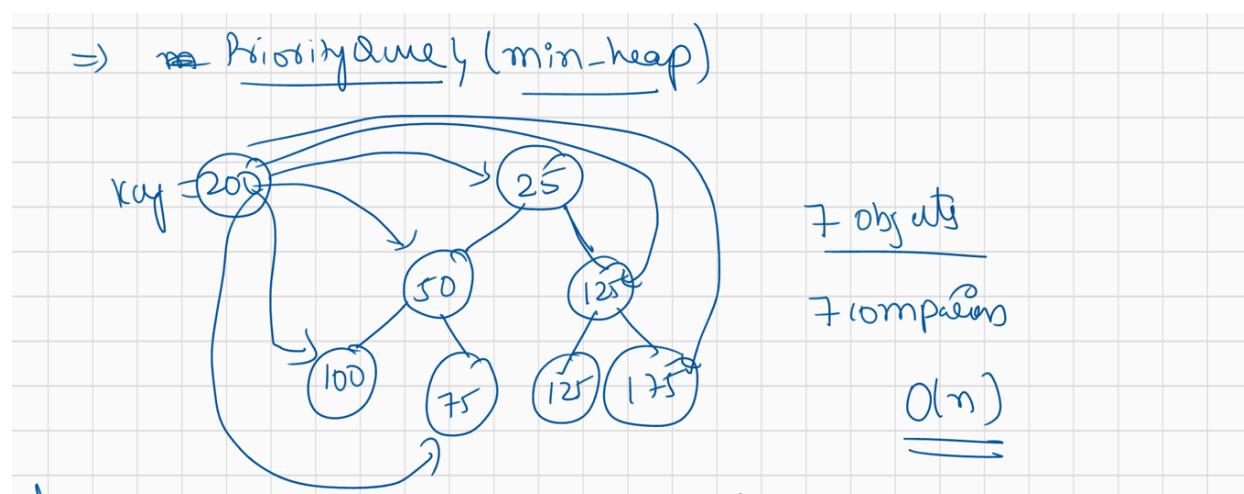
64)12-12-22

(collection 2)

Behind the scene auto boxing is happening

Collections follows specific data structure

PriorityQueue



- It follows the min-heap data structure
- It implements Queue()
- Whichever element has the highest priority will appear at the first element.
- It is implemented in left and right way

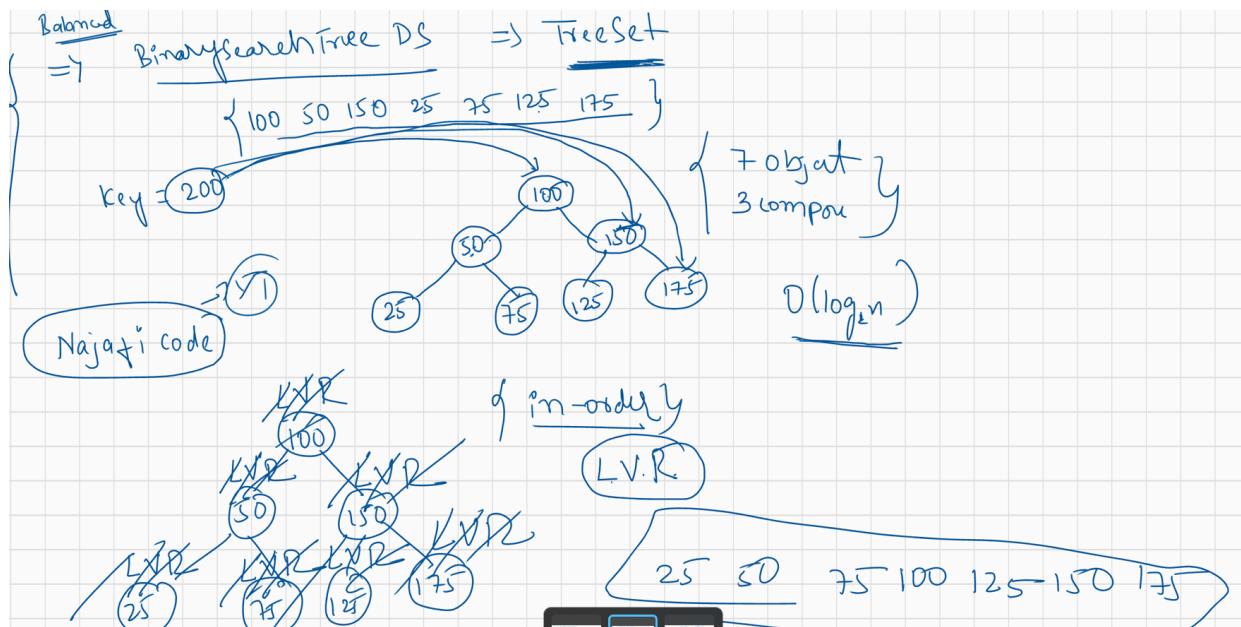
```
import java.util.*;
```

```

public class LaunchPQ {
    public static void main(String[] args) {
        PriorityQueue pq = new PriorityQueue();
        pq.add(100);
        pq.add(50);
        pq.add(150);
        pq.add(25);
        pq.add(75);
        pq.add(125);
        pq.add(175);
        System.out.println(pq);
        pq.add(25);
        System.out.println(pq);
    }
}

```

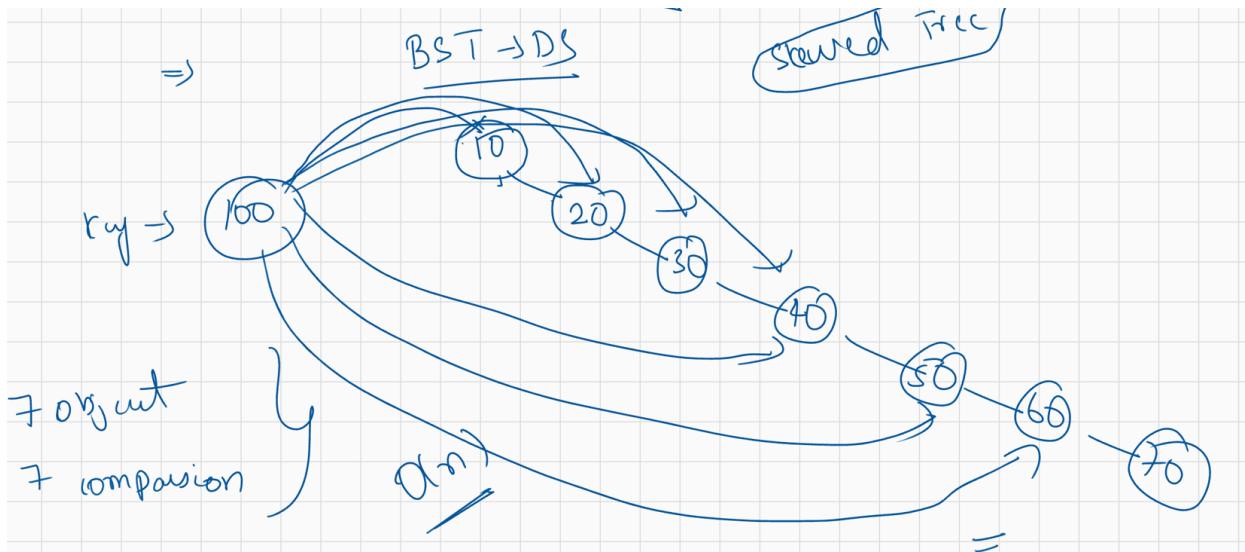
TreeSet



- It follows Balanced Binary Search Tree Data Structure.
- It implements Tree
- It works on the LVR basis
- If it is a balanced binary search tree then the searching is fast $O(\log n)$. if it is skewed tree then it would take the time complexity same as of the other collections $O(n)$

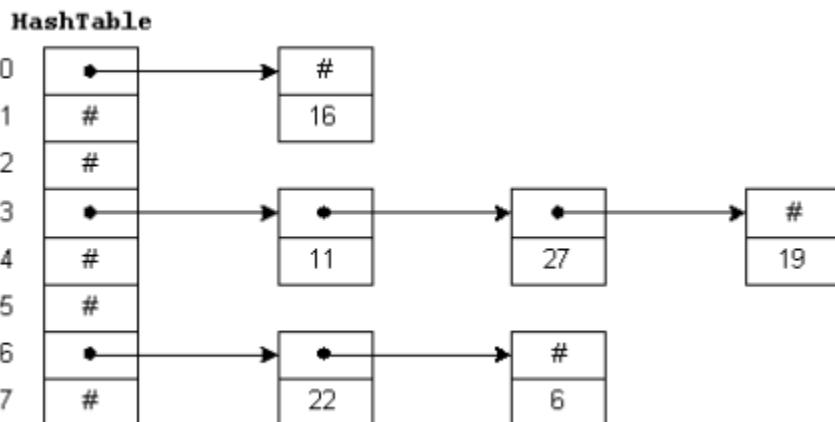
```
import java.util.*;  
  
public class LaunchTS {  
    public static void main(String arg[])  
    {  
        TreeSet ts = new TreeSet<>();  
        ts.add(100);  
        ts.add(50);  
        ts.add(150);  
        ts.add(25);  
        ts.add(75);  
        ts.add(125);  
        ts.add(175);  
        System.out.println(ts);  
        System.out.println(ts.ceiling(50));  
        System.out.println(ts.higher(50));  
        System.out.println(ts.floor(50));  
        System.out.println(ts.lower(40));  
  
        System.out.println("*****");  
        System.out.println(ts.ceiling(40));  
        System.out.println(ts.higher(40));  
        System.out.println(ts.floor(40));  
        System.out.println(ts.lower(40));  
    }  
}
```

Skewed Tree



HashSet

- It follows Hashing algorithm (red and black algorithm)
- There will be hash function and associated hashtable with load factor 75%
- Hashtable will have buckets where data is stored
- Order of insertion is not preserved. HashSet will not maintain the order of insertion
- Searching operations will be faster.



Java Hashtable

```
import java.util.HashSet;

public class Hs {
    public static void main(String arg[])
    {
        HashSet hs = new HashSet<>();
        hs.add(100);
        hs.add(50);
```

```
        hs.add(150);
        hs.add(25);
        hs.add(125);
        System.out.println(hs);

    }
}
```

LinkedHashSet

- LinkedHashSet also follows the Hashing algorithm behind the scenes.
- LinkedHashSet is a subclass of HashSet.
- It preserves the order of insertion.
- Searching operation will be faster

```
import java.util.HashSet;
import java.util.LinkedHashSet;

public class Lhs {
    public static void main(String arg[])
    {
        LinkedHashSet lhs = new LinkedHashSet<>();
        lhs.add(100);
        lhs.add(50);
        lhs.add(150);
        lhs.add(25);
        lhs.add(125);
        System.out.println(lhs);

    }
}
```

HOW TO ACCESS THE ELEMENTS FROM THE COLLECTION ?

```
import java.util.ArrayDeque;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.ListIterator;
```

```
public class Launchimpl {
    public static void main(String arg[])
    {
        ArrayList al = new ArrayList<>();
        al.add(10);
        al.add(10.5);
        al.add('c');
        al.add("sid");
        al.add(10.4f);
        System.out.println(al);

        System.out.println("*****");
        ArrayList al2 = new ArrayList<>();
        al2.add(10);
        al2.add(20);
        al2.add(30);
        al2.add(40);
        //diff ways to access data present within collection
        //loop normal
        System.out.println(al2.size());
        for(int i=0;i<al2.size();i++)
        {
            Object o = al2.get(i);
            System.out.println(o);
        }

        System.out.println("*****");
        for(Object obj : al2)
        {
            System.out.println(obj);
        }

        System.out.println("*****0000000*****");
        //iterator
        //by nature traversing it will get all values
        Iterator itr1 = al2.iterator();
        // if(itr1.hasNext()==true)//check will data next to you present
        // {
        //     System.out.println(itr1.next());
        
```

```
// }

//iterator is like cursor ,
//while traversing to fetch the data

System.out.println("000000000000");
while(itrl.hasNext()){
    //Object obj = itrl.next();
    Integer i = (Integer)itrl.next();
    System.out.println(i);
}

//how to fetch in reverse using the iterator
System.out.println("reverse");
//out of 7 collection class, (ListIterator) reversing is only
present in ArrayList and LinkedList
//List based

ListIterator litr = al2.listIterator(al2.size());
while(litr.hasPrevious())
{
    System.out.println(litr.previous());
}

//what if I want to reverse or access data of other classes in
reverse direction ?
ArrayDeque ad = new ArrayDeque<>();
ad.add(12);
ad.add(20);
ad.add(30);
ad.add(40);
Iterator it = ad.iterator();

while(it.hasNext() == true)
{
    Integer i = (Integer)it.next();
    System.out.println("Array De " + i);
}

System.out.println("added linked list");
```

```

LinkedList ll = new LinkedList<>();
ll.addAll(ad);
System.out.println(ll);

ListIterator ll2 = ll.listIterator(ll.size());
while(ll2.hasPrevious()) {
    System.out.println(ll2.previous());
}
}
}

```

65)13-12-22

(collection 3)
Java as a language
JDBC SQL JEE

```

DescendingIterator();
import java.util.LinkedList;
import java.util.Iterator;
import java.util.ListIterator;

//import javax.swing.text.html.HTMLDocument.Iterator;

public class launchimpl2 {
    public static void main(String arg[]){
        //LinkedList , ArrayDeque and TreeSet

        //descendingIterator is present in 3 class only
        LinkedList ll2=new LinkedList();
        ll2.add(100);
        ll2.add(200);
        ll2.add(300);
        ll2.add(400);
        ll2.add(500);
        System.out.println(ll2);
    }
}

```

```
Iterator ditr=ll2.descendingIterator();

while(ditr.hasNext())
{
    //Integer i=(Integer) ditr.next();
    System.out.print( ditr.next() + " ");
}

}
```

Other Concepts

Vector and Enumeration

//They have not removed vector and enumeration because already lots of softwares are developed using vectors and enumeration

```
import java.util.Enumeration;
import java.util.Vector;

class vec {
    public static void main(String arg[]){

        //before collection and iterator. this are well used
        Vector v = new Vector();
        v.add(100);
        v.add(200);
        v.add(300);
        v.add(400);

        //Enumeration works like iterator only
        Enumeration em = v.elements(); //this method is present in
enumeration
        while(em.hasMoreElements()){
            System.out.println(em.nextElement());
        }
    }
}
```

```

        }

        //only if vector is used then only we can use the Enumeration
    }
}

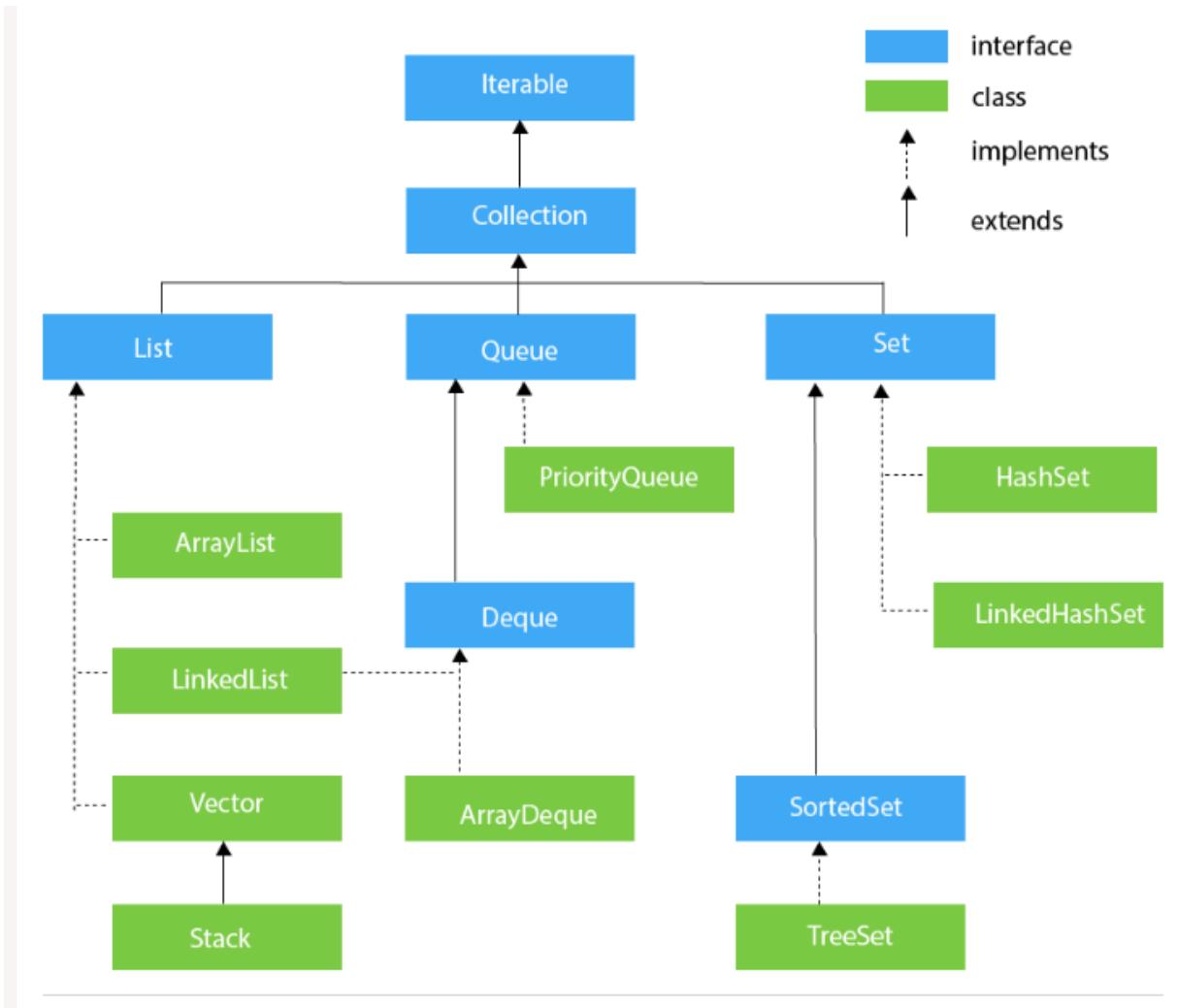
```

The screenshot shows the Java documentation for the `descendingIterator()` method of the `LinkedList` class. The method signature is `descendingIterator() : Iterator<T>`. The description states: "Returns an iterator over the elements in this deque in reverse sequential order. The elements will be returned in order from last (tail) to first (head)." It includes a bullet point for **Returns:** "an iterator over the elements in this deque in reverse sequence" and another for **Since:** "1.6". On the right, there is sample code demonstrating how to use this iterator to print elements in reverse order.

After `:` whatever class it shows the we need write that parent type

	for() loop	for-each	Iterator	List Iterator	descending iterator	Enumeration
ArrayList	✓	✓	✓	✓	✗	✗
LinkedList	✓	✓	✓	✗	✓	✗
ArrayDeque	✗	✓	✓	✗	✓	✗
PriorityQueue	✗	✓	✓	✗	✓	✗
TreeSet	✗	✓	✓	✗	✓	✗
HashSet	✗	✓	✓	✗	✗	✗
LinkedHashSet	✗	✓	✓	✗	✗	✗

<https://marcus-biel.com/java-collections-framework/>



There is a concept of fail fast and fail safe , concurrency modification

classy	internal DS	parallel ordering invariant	permits null value	Duplicate
ArrayList	Dynamic array DS	✓	✓	✓
LinkedList	Doubly linked list	✓	✓	✓
ArrayDeque	double ended queue	✓	✗	✓
PriorityQueue	min-heap	✗	✗	/
TreeSet	Binary search tree	✗	✗	✗
HashSet	Hashtable (Hasing)	✗	✓	✗
LinkedHashSet	Hashtable	✓	✓	✗

FailFast and FailSafe

It possible only in Concurrent Package. Not in all other collection classes

Assignment = study more about concurrent packages. (what if we can use try and catch)[study more on it]

FailFast means While you are attempting concurrent modification you application fails fast leading to abnormal termination

```
ArrayList al = new ArrayList<>();

        al.add(100);
        al.add(200);
        al.add(300);
        al.add(400);
        al.add(500);

        for(int i=0;i<al.size();i++)
        {
            //this line access and prints
            System.out.println(al.get(i));
            al.add(100); //Structural modification or Concurrent
modification = while accessing your trying to modifiy the data the
collection.
```

```

        //infinite times it goes

        //if we use normal for loop or for each loop in collection it
may lead to structural modification
        //so instead of for or for each , iterator is used.
        Iterator itr = al.iterator();

        //FailFast
        while(itr.hasNext())
        {
            System.out.println(itr.next());
            al.add(300);
            //this only we call as FailFast
            //=while accessing data from collection,add new data to
collection leads to FailFast.
            //it keeps on growing if for or foreach loop is
used.(illlogical)

        }
    }
}

```

FastSafe means While you are attempting concurrent modification it will stop the modification without failing the runtime at execution.

```

import java.util.ArrayList;
import java.util.Iterator;
//import javax.swing.text.html.HTMLDocument.Iterator;
import java.util.concurrent.CopyOnWriteArrayList;

public class LaunchFFFS {
    public static void main(String arg[])
    {

        ArrayList al = new ArrayList<>();

        al.add(100);
        al.add(200);
        al.add(300);
        al.add(400);
        al.add(500);

        for(int i=0;i<al.size();i++)
    }
}

```

```

{
    //this line access and prints
    System.out.println(al.get(i));
    // al.add(100); //Structural modification or Concurrent
modification = while accessing your trying to modifiy the data the
collection.
    //infinite times it goes

    //if we use normal for loop or for each loop in collection it
may lead to structural modification
    //so instead of for or for each , iterator is used.
    Iterator itr = al.iterator();

    //FailFast
    while(itr.hasNext())
    {
        System.out.println(itr.next());
        // al.add(300);
        //this only we call as FailFast
        //=while accessing data from collection,add new data to
collection leads to FailFast.
        //it keeps on growing if for or foreach loop is
used.(illlogical)

    }

    //FailSafe
    //=where concurrent modification shouldn't happen and
exception shouldn't happen
    //we have concurrent package under all collection classes are
there
    CopyOnWriteArrayList cal = new CopyOnWriteArrayList<>();
    cal.add(1000);
    cal.add(2000);
    cal.add(3000);

    Iterator it = cal.iterator();
    while(it.hasNext())
    {
        System.out.println(it.next());
    }
}

```

```
        cal.add(12345); //FailSafe
    }
}
}
}
```

Collection vs Collections

Collection is an Interface, It is a framework

Collection is a hierarchy, set of interfaces and classes

Collections is a class

Collections is an Individual class

```
import java.util.ArrayList;
import java.util.Collection;
import java.util.Collections;

class Student
{
    private String name;
    private int age;
    private int marks;

    public Student(String name,int age,int marks)
    {
        this.name = name;
        this.age = age;
        this.marks = marks;
    }

    public String getName() {
        return name;
    }
    public int getAge() {
        return age;
    }
    public int getMarks() {
        return marks;
    }
}
```

```
}

public class LaunchGen {
    public static void main(String arg[])
    {
        Student s1 = new Student("Ram", 19, 100);
        Student s2 = new Student("sham", 21, 95);
        Student s3 = new Student("Lakshman", 17, 98);

        ArrayList<Student> al = new ArrayList<Student>();
        al.add(s1); //value as object
        al.add(s2);
        al.add(s3);
        // al.add(100);

        System.out.println(al);
        //you can't perform sorting using normal Collections
        //Collections.sort(al);
        //then the concept of comparator and comparable comes into
picture.

        //few more important inbuilt methods of Collections class
        ArrayList al4 = new ArrayList<>();
        al4.add(10);
        al4.add(20);
        al4.add(30);
        al4.add(40);
        int index = Collections.binarySearch( al4, 40);
        System.out.println("Index "+ index);

        Collections.shuffle(al4);
        System.out.println(al4);

        Collections.max(al4);
        System.out.println(Collections.max(al4));

        System.out.println(Collections.frequency(al4, 40));
    }
}
```

34)CodeSnippets

(2:37:00)

66)14-12-22 dbt

67)14-12-22

Collection hierarchy has set of interfaces

Duplicate data is allowed only in set based classes

(Map)

eg.

Aadhar Card is an identity card as Indian

Aadhar Passport => duplicate data is name,age,gender , father name

Aadhar passport numbers are unique

In real world unique data is present in the form of pair

That we call as

Key - Value

Key = Associated data/value

Eg. your aadhar card has unique number, to that number your data is associated

Data in our world is also present in the form of key-value pair.

Eg. hallticket number - name

Eg.

football/Circket

10 : Sachin

18 : virat

07 : MSD

10 : Messi

: Ronaldo

If data in the world is present as key-value pair.In collection which class can store like this.

{Dictionary , Maps and Property } this can store in key-value, but they are not under the collection

=>So, java people came up with an hierarchy called Map hierarchy

Whenever data in the form of key and value then it has to be stored in Maps

MAP HIERARCHY

Under maps hierarchy there are many class
⇒ **HashMap** , **Hashtable** , **TreeMap** , **Properties**.
When it is key-value we go with Map
Mostly used is **HashMap**

⇒ It is introduced in JDK 1.2v

Jdk 1.0v Legacy classes

Dictionary(AC)

 ^

 |

Hashtable(c)

 ^

 |

Properties(c)

Whenever there is key-value pair then we go with Map hierarchy

Map is different hierarchy from collections

Both pair of Key-Value we call it as entry

10 : Sachin ⇒ entry

```
import java.util.HashMap;
import java.util.LinkedHashMap;

public class HashMapConc {
    public static void main(String arg[])
    {
        //Map hm = new HashMap();
        HashMap hm = new HashMap();
        //Not maintains the order of insertion //jdk 1.2

        hm.put(10, "Sachin");
        hm.put(7, "MSD");
        hm.put(18, "Kohli");
        System.out.println(hm);
```

```

//linkedhashmap is a sub class of hashmap
LinkedHashMap lhm = new LinkedHashMap<>();
//maintains the order of insertion //jdk 1.4
lhm.put(10, "Sachin");
lhm.put(7, "MSD");
lhm.put(18, "Kohli");
System.out.println(lhm);
}
}

```

Key should be always unique, value need not to be unique
 Like dictionaries in python

We have an interface called Map
 Interface inside interface is allowed
 It is tightly coupled
 (by knowing the aggregation and composition concepts)

Eg. heart and bike of a student
 As long as student is alive heart is along with him= you can access heart \Rightarrow it is tightly coupled = aggregation.
 If he is dead = you can't access his heart

Bike even you can access whether he is alive or not \Rightarrow it is loosely coupled.==>composition

Fetching data from collection = is done using iterator

How do we fetch the data from the Map
 \Rightarrow there is something called as view and entryset

```

Collection c = hm.values();
Iterator itr = c.iterator();
while(itr.hasNext())
{
    System.out.println(itr.next());
}

Set s = hm.keySet();
Iterator itr2 = s.iterator();
while(itr2.hasNext())
{

```

```

        System.out.println(itr2.next());
    }

    Set es = hm.entrySet();
    Iterator itr3 = es.iterator(); //traversing and fetching
    while(itr3.hasNext())
    {
        System.out.println(itr3.next());
    }

```

If there key have to be number ?

Ans. NO

Build aadhar and passport.

HashMap

- Internally it follows the Hashtable
- Order of insertion is not preserved
- Parent of LinkedHashMap

LinkedHashMap

- Internally it follows the Hashtable and LinkedList
- Order of insertion is preserved

```

import java.util.Collection;
import java.util.HashMap;
import java.util.Iterator;
import java.util.LinkedHashMap;
import java.util.Map;
import java.util.Set;
import java.util.Map.Entry;

public class HashMapConc {
    public static void main(String arg[])
    {
        //Map hm = new HashMap();
        HashMap hm = new HashMap();
        //Not maintains the order of insertion //jdk 1.2
    }
}

```

```

hm.put(10, "Sachin");//Entry
hm.put(7, "MSD");
hm.put(18, "Kohli");
System.out.println(hm);

Collection c = hm.values();
Iterator itr = c.iterator();
while(itr.hasNext())
{
    System.out.println(itr.next());
}

Set s = hm.keySet();
Iterator itr2 = s.iterator();
while(itr2.hasNext())
{
    // System.out.println(itr2.next());
    Integer i = (Integer)itr2.next();
    System.out.println("key : "+i);
}

Set es = hm.entrySet();
Iterator itr3 = es.iterator(); //traversing and fetching
while(itr3.hasNext())
{
    //System.out.println(itr3.next());
    Map.Entry data=(Entry) itr3.next();
    //Entry is an interface inside the Map interface, so '.' operator is used.
    System.out.println(data.getKey() + " : " + data.getValue());
}

//linkedhashmap is a sub class of hashmap
LinkedHashMap lhm = new LinkedHashMap<>();
//maintains the order of insertion //jdk 1.4
lhm.put(10, "Sachin");
lhm.put(7, "MSD");
lhm.put(18, "Kohli");

```

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

```
//package Maps;

import java.util.Collection;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
import java.util.Set;
import java.util.Map.Entry;

//import javax.swing.text.html.HTMLDocument.Iterator;

class Student
{
    private String name;
    private int age;
    private String city;

    public Student(String name,int age,String city)
    {
        this.name = name;
        this.age = age;
        this.city = city;
    }

    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }

    public String getCity() {
        return city;
    }

    //overriding the toString() method
    //bcz to get the values instead of addresses
    @Override
```

```
public String toString()
{
    return name + " " + age + " " + city;
}

}

public class LaunchHM {
    public static void main(String arg[]){
        Student std1 = new Student("Sita",18,"hyderabad");
        Student std2 = new Student("Ram",21,"Ayodhaya");
        Student std3 = new Student("Lakshman",20,"Ayodhaya");
        Student std4 = new Student("Hanuman",25,"kerala");

        HashMap hm = new HashMap();
        hm.put(1, std1);
        hm.put(2, std2);
        hm.put(3, std3);
        hm.put(4, std4);
        System.out.println(hm);

        Collection col = hm.values();
        Iterator itr4 = col.iterator();
        while(itr4.hasNext())
        {
            System.out.println("values of Students : "+itr4.next());
        }

        System.out.println("*****");
        Set s=hm.keySet();
        Iterator itr6 = s.iterator();
        while(itr6.hasNext())
        {
            System.out.println("key of Students : "+itr6.next());
        }

        Set bothData = hm.entrySet();
        Iterator i = bothData.iterator();
```

```
        while(i.hasNext())
    {
        Map.Entry en = (Entry) i.next();
        System.out.println(en.getKey() + " : " + en.getValue());
    }
}
```

```
import java.util.*;
import java.util.Map.Entry;

class PassPortDetails
{
    private String name;
    private int age;
    private String fatherName;
    private String city; //try dependency injection

    public PassPortDetails(String name, int age, String fatherName, String
city)
    {
        this.name = name;
        this.age = age;
        this.fatherName = fatherName;
        this.city = city;
    }

    public String getName() {
        return name;
    }
    public int getAge() {
        return age;
    }
    public String getFatherName() {
        return fatherName;
    }
    public String getCity() {
        return city;
    }
}
```

```

@Override
public String toString()
{
    return name + " " + age + " " + fatherName + " " + city;
}

}

// class Key
// {
//     int key;
//     public Key(int key)
//     {
//         this.key = key;
//     }

//     @Override
//     public String toString()
//     {
//         return ""+key;
//     }
// }

public class LaunchPassport {
    public static void main(String arg[])
    {
        PassPortDetails pp1 = new PassPortDetails("sidduganesh", 21,
"srinivas", "Hyderabad");
        PassPortDetails pp2 = new PassPortDetails("AakashGanesh", 19,
"srinivas", "Hyderabad");
        // Key k1 = new Key(23347789);

        HashMap hm = new HashMap<>();
        // hm.put(k1, pp1);
        // System.out.println(hm);
        //for key you can creat separate class
        hm.put("crio123", pp1);
        hm.put("crio234", pp2);

        Set bothData = hm.entrySet();
        Iterator i = bothData.iterator();
    }
}

```

```
        while(i.hasNext())
        {
            Map.Entry en = (Entry) i.next();
            System.out.println("Passport Details "+ en.getKey() + " : " +
en.getValue());
        }

    }
}
```

CodeSnippets

(2:37:00)

68)15-12-22

(Maps and Interface New Features)

weakHashMap(c) it will not dominate the garbage collector
LinkedHashMap will dominate the garbage collector

All the method in hashtable are synchronized

```
import java.util.*;
import java.util.Map.Entry;

class PassPortDetails
{
    private String name;
    private int age;
    private String fatherName;
    private String city; //try dependency injection

    public PassPortDetails(String name, int age, String fatherName, String
city)
    {
        this.name = name;
        this.age = age;
    }
}
```

```
        this.fatherName = fatherName;
        this.city = city;
    }

    public String getName() {
        return name;
    }
    public int getAge() {
        return age;
    }
    public String getFatherName() {
        return fatherName;
    }
    public String getCity() {
        return city;
    }

    @Override
    public String toString()
    {
        return name + " " + age + " " + fatherName + " " + city;
    }

}
// class Key
// {
//     int key;
//     public Key(int key)
//     {
//         this.key = key;
//     }

//     @Override
//     public String toString()
//     {
//         return ""+key;
//     }
// }

public class LaunchPassport {
    public static void main(String arg[])
}
```

```

{
    PassPortDetails pp1 = new PassPortDetails("sidduganesh", 21,
"srinivas", "Hyderabad");
    PassPortDetails pp2 = new PassPortDetails("AakashGanesh", 19,
"srinivas", "Hyderabad");
    // Key k1 = new Key(23347789);

    HashMap hm = new HashMap<>();
    // hm.put(k1, pp1);
    // System.out.println(hm);
    //for key you can crear separate class
    hm.put(123, pp1);
    hm.put(456, pp2);
    System.out.println("Welcome to the passport details APP");
    Scanner sc = new Scanner(System.in);
    System.out.println("Enter the number of the passport : ");
    Integer k = sc.nextInt();
    boolean flag = false;

    Set bothData = hm.entrySet();
    Iterator i = bothData.iterator();
    while(i.hasNext())
    {
        Map.Entry passport = (Entry)i.next();
        Integer key = (Integer)passport.getKey();
        if(k==key)
        {
            System.out.println("Passport Details "+ passport.getKey() + " :
" + passport.getValue());
            flag = true;
            //System.exit(0);
        }
        // else
        // {
        //     System.out.println("Invalid passport number");
        // }
    }
    if(flag==false)
        System.out.println("Invalid passport number");
}

```

```
    }
}
```

WeakHashMap

```
import java.util.*;  
  
class Employee  
{  
    private int empId;  
    private String empName;  
    private String empAddr;  
  
    @Override  
    public String toString()  
    {  
        return "sid";  
    }  
  
    @Override  
    public void finalize()  
    {  
        System.out.println("Garbage collector collected the object");  
    }  
}  
  
public class WeakhashMap {  
    public static void main(String arg[])  
    {  
        Employee emp = new Employee();  
        WeakHashMap hm = new WeakHashMap<>();  
        hm.put(emp, "sidduganesh");  
  
        emp = null; // eligible for garbage collector  
  
        System.gc(); //gc internally uses the finalize method
```

```
        System.out.println("Last line");

    }
}
```

Synchronize means at a time only one thread can run
>javap java.util.Hashtable

Hashtable is a legacy class present from jdk 1.0
HashMap ia a present from the jdk 1.2

//Hashtable - all are synchronized methods (multithreading is not possible)
//HashMap - all are non sychronized methods (multithreading is possible)

Hashtable is thread safe
hashMap is not thread safe

Hashtable is low performance
hashMap is high performance

```
import java.util.TreeMap;

public class LaunchTM {
    public static void main(String arg[])
    {
        TreeMap tm = new TreeMap<>();
        tm.put(11, "sid");
        tm.put(12, "siddu");
        tm.put(13, "ganesh");

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

o/p
{11=sid, 12=siddu, 13=ganesh}

Like a TreeSet

collection.sort()
Can't sort the complex data, then comparator and comparable concepts are used

```
// equals(); HashMap  
// ==: IdentityHashMap
```

Default access specifier is used in an interface for the implementation of body of method inside interface

CodeSnippets

69)16-12-22

(Introduction to Generics)

Agenda:

1. Introduction
2. Type-Safety
3. Type-Casting
4. Generic Classes
5. Bounded Types
6. Generic methods and wild card character(?)
7. Communication with non generic code
8. Conclusions

Def : The main objective of Generics is to provide Type-Safety and to resolve Type-Casting problems.

Case 1: Type-Safety

Arrays are always type safe that is we can give the guarantee for the type of elements present inside array.

For example if our programming requirement is to hold String type of objects it is recommended to use String array.

In the case of string array we can add only string type of objects by mistake if we are trying to add any other type we will get compile time error.

eg:

```
String name[] =new String[500];  
name[0] = "Navin Reddy";  
name[1] = "Haider";  
name[2] = new Integer(100); //CE: incompatible types found: java.lang.Integer  
required: java.lang.String
```

That is we can always provide guarantee for the type of elements present inside array and hence arrays are safe to use with respect to type that is arrays are type safe.

But collections are not type safe that is we can't provide any guarantee for the type of elements present inside collection.

For example if our programming requirement is to hold only string type of objects it is never recommended to go for ArrayList.

By mistake if we are trying to add any other type we won't get any compile time error but the program may fail at runtime.

eg:

```
ArrayList al =new ArrayList();
al.add("NavinReddy");
al.add("Haider");
al.add(new Integer(10));
...
...
...
String name1 = (String)al.get(0);
String name2 = (String)al.get(1);
String name3 = (String)al.get(2); //Exception in thread "main" ::  
java.lang.ClassCastException
```

java.lang.Integer cannot

be cast to java.lang.String

Hence we can't provide guarantee for the type of elements present inside

collections that is collections are not safe to use with respect to type.

Case 2: Type-Casting

In the case of array at the time of retrieval it is not required to perform any type casting.

eg::

```
String name[] =new String[500];
name[0] = "Navin Reddy";
name[1] = "Haider";
...
...
String data =name[0]; //here type casting is not required.
```

But in the case of collection at the time of retrieval compulsory we should perform type casting otherwise we will get compile time error.

eg::

```
ArrayList al =new ArrayList();
al.add("NavinReddy");
al.add("Haider");
String name1= al.get(0); //CE: incompatible types : found : java.lang.Object  
required:
```

```
java.lang.String  
String name1=(String) al.get(0);//At the time of retrieval type casting is  
madantory  
That is in collections type casting is bigger headache.  
To overcome the above problems of collections(type-safety, type casting)sun people  
introduced generics concept in 1.5v  
hence the main objectives of generics are:
```

1. To provide type safety to the collections.
2. To resolve type casting problems.

To hold only string type of objects we can create a generic version of ArrayList as follows.

```
ArrayList<String> al =new ArrayList<String>();  
al.add("NavinReddy");  
al.add(10); //CE: can't find symbol
```

symbol: method add(int)
location : class

```
java.util.ArrayList<java.lang.String>
```

```
al.add(10)
```

For this ArrayList we can add only string type of objects by mistake if we are trying to add any other type we will get compile time error that is through generics we are getting type safety.

At the time of retrieval it is not required to perform any type casting we can assign elements directly to string type variables.

eg:

```
ArrayList<String> al =new ArrayList<String>();  
al.add("NavinReddy");  
....  
....  
String name =al.get(0); //type casting is not required as it is an TypeSafe
```

That is through generic syntax we can resolve type casting problems.

Conclusions

=====

1. Polymorphism concept is applicable only for the base type but not for parameter type

[usage of parent reference to hold child object is called polymorphism].

```
eg: ArrayList<String> al =new ArrayList<String>();  
List<String> al =new ArrayList<String>();  
Collection<String> al =new ArrayList<String>();
```

Collection<Object> al =new ArrayList<String>();//CE: incompatible types

2.

Collections concept applicable only for objects , Hence for the parameter type we can use any class or interface name but not primitive value(type).Otherwise we will get compile time error.

eg: ArrayList<int> al =new ArrayList<int>();//CE: unexcpeted type
found:primitive

required:

reference

70)17-12-22 dbt

71)19-12-22

(Enum and Annotations)

Enum is too less used

Enum introduced in JDK 1.5v or JAVA 5

How to make a variable constant

=using final keyword

Contents name recommandation is used as CAPITALS

Enum = group of named or predefined constants

Our own data types

We can defined enum inside or outside of the class it is valid.

For enum also separate .class file is generated.

The screenshot shows a Java code editor with the following code:

```

public class LaunchEnum2 {
    Run | Debug
    public static void main(String arg[])
    {
        Result res = Result.PASS;
        System.out.println(res);

        Result.FAIL.
    }
}

```

A tooltip for the `compareTo` method of the `Result` enum is displayed, showing its signature and documentation:

`Enum.compareTo(Result o) : int`

Compares this enum with the specified object for order. Returns a negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object. Enum constants are only comparable to other enum constants of the same enum type. The natural order implemented by this method is the order in which the constants are declared.

- Parameters:
 - o the object to be compared.
- Returns:

At last of methods present is only know as the type of data

Annotations are most important

Inbuilt annotations uses the enum concepts

ANNOTATIONS

In real world there are thousands and lakhs of lines of code.

To understand the specific code we write the comments and annotations

Also give message to compilers, developer and browser

So other than comments, then Annotations used.

Compiler keeps the annotations code.

Some extra info embedding to code.

Annotation → Annotation

Parent of all annotations

Types

Annotation → Annotation

Annotation → Built in

Customized annotation

Assigning the data to annotations

//multi valued

//Annotation

@Interface CricketPlayer

{

//@--> its not interface but its annotation being created.

```

String name();
Int runs();

//String country() default "India";
//int runs() default 20000;

}

//specifiy using annotation
@CricketPlayer(country="India", runs=2000)
Class ViratKohli
{
    Private int innings;
    private String name;
    //generate setters and getters
}

//main code
ViratKohli vk = new ViratKohli();
vk.setInnings(200);
vk.setName("VK");

sysout(vk.getInnings());
sysout(vk.getName());

Class c = vk.getClass();
Annotation an = c.getAnnotation(CricketPlayer.class);
CricketPlayer cp = (CricketPlayer)an;

Int run = cp.runs();
sysout(run);
String cn = cp.country();
sysout(cn);

@FunctionallInterface
Interface Trail
{
    int getNum();
}

//inside main code
Trail t = ()->{
    return 10;
}

```

annotations

Applicable to

- 1)class
- 2)interface
- 3)method
- 4)fields //instance var
- 5)local variables
- 6)constructor
- 7)parameters
- 8)enum

Whenever you use annotations u need to specify the 2 things

//Target
//retention policy

Meta annotation

Annotation for another annotations

@Target(ElementType.TYPE) = then it can be used for class,interface and parameter

@Retention(RetentionPolicy.RUNTIME) = till where , to compiler or jvm or ...

@interface CricketPlayer

//98% we won't create custom annotations

Enums can be passed in switch..

Adding extra info to code.

It can reach to compiler and interpreter level

Applicable to class , interface ,....

Single , multi valued annotation

Every annotation has target for customized

Customized annotations works based on interfaces

@interface Circket

@Target({ElementType.TYPE, ElementType.METHOD, ElementType.LOCAL})

GENERICS(from jdk1.5)

Regex if not much important

Before generics

class ArrayList

{

 add(Object o);
 Object get(int index);

```
}
```

```
ArrayList all = new ArrayList();
```

```
al.add("sachin");
```

```
al.add(new Integer(10)); //no ce
```

```
String data = (String)al.get(0);
```

```
Integer i = (Integer)al.get(1);
```

```
String result = (String)al.get(1); //RE: java.lang.ClassCastException
```

After Generics

```
class ArrayList<T>
{
    add(T t);
    T get(int index);
}

ArrayList<String> all = new ArrayList<String>();
al.add("sachin");
String data = al.get(0);
al.add(new Integer(10)); //ce

//predefined-api
//ArrayList<T>
//LinkedList<T>
//HashSet<T>
//LinkedHashSet<T>

Echo %path%
```

```
class Demo<T extends Number>{
```

```
}
```

```
//X -> class means, we can pass either X type or its child Type
```

```
//X -> interface means , we can pass either X or its implementation class Sample< T extends Runnable>
```

```
{
```

```
}
```

```
//implements and super is not used in place of extends
```

Class Test

```
{
Public static void main(String arg[])
{
    Demo<Integer> d1 = new Demo<Integer>();
    Demo<Number> d3 = new Demo<Number>();
```

```
Demo<String> d2 = new Demo<String>();//ce

Sample<Runnable> s1 = new Sample<Runnable>();
Sample<Thread> s2 = new Sample<Thread>();

Sample<String> s3 = new Sample<String>();
}

}
```

CodeSnippets

72)20-12-22

(Generics)
ArrayList<T> al = new ArrayList<T>();

```
class ArrayList<T>{
    void add(T t)
    T get(int index)
}
```

```
class Demo<T extends X>{

}
new Demo<String>();
new Demo<Integer>();
```

Bounded Types

=====

```
class Demo<T extends X>{}
class Demo<T implements X>{}
class Demo<T super X>{}
```

If X -> is class, then X type of its Child Type
X -> interface, then X type of its implementation class type

73)21-12-22 dbt

74)21-12-22

(comparator vs comparable)

```
/*
//meant for natural sorting order
//public java.util.TreeSet;
* interface Comparable{
*   public int compareTo(Object obj);
* }
* obj1.compareTo(obj2)
*           |=>returns -ve value, if obj1 has to come before obj2
*           |=>returns +ve value, if obj1 has to come after obj2
*           |=>returns 0 value, if obj1 is equals to obj2
*
* //meant for customized sorting
* //public java.util.TreeSet(java.util.Comparator<? super E>);
* interface Comparator{
*   public int compare(Object obj1, Object obj2);
*   public boolean equals (Object obj);
* }
*   compare(Object obj1, Object obj2)
*           |=>returns -ve value, if obj1 has to come before obj2
*           |=>returns +ve value, if obj1 has to come after obj2
*           |=>returns 0 value, if obj1 is equals to obj2
* If we keep objects inside TreeSet, internally JVM uses compareTo() and
it sorts the object based on the
* sorting result the object will be stored in treeset
*/
import java.util.*;

class MyComparator implements Comparator
{
    @Override
    public int compare(Object obj1, Object obj2)
    {
        //logic comming soon....
```

```

        Integer i1 = (Integer)obj1;
        Integer i2 = (Integer)obj2;

        // return i1.compareTo(i2);
        // return -i1.compareTo(i2);
        // return +1 ⇒ insertion order
        // return -1 ⇒ reverse of insertion order
        // return 0 ⇒ only first element

        if(i1<i2)
            return -1;
        else if(i1>i2)
            return +1;
        else
            return 0;
    }

}

public class compareTo {
    public static void main(String arg[])
    {
        TreeSet ts = new TreeSet();
        ts.add("A");
        ts.add("Z");
        ts.add("K");

        System.out.println("A".compareTo("Z")); // -ve
        System.out.println("Z".compareTo("K")); // +ve
        System.out.println("Z".compareTo("Z")); // 0

        System.out.println("Z".compareTo("null")); // java.lang.NullPointerException

        TreeSet ts1 = new TreeSet(new MyComparator());
        ts1.add(10);
        ts1.add(5);
        ts1.add(3);
        ts1.add(20);
        System.out.println(ts1);
    }
}

```

```
return i1.compareTo(i2);
return -i1.compareTo(i2);
```

Note:

If we are Depending on Default Natural Sorting Order Compulsory Objects should be Homogeneous and Comparable.

Otherwise we will get RE: ClassCastException.

An object is said to be Comparable if and only if corresponding class implements Comparable interface.

All Wrapper Classes, String Class Already Implements Comparable Interface. But StringBuffer Class doesn't Implement Comparable Interface.

```
@FunctionalInterface
public interface Comparable<T> {
    public abstract int compareTo(T);
}
obj1.compareTo(obj2)
|=> returns -ve value, if obj1 has to come before obj2
|=> returns +ve value, if obj1 has to come after obj2
|=> returns 0 value, if both obj1 and obj2 are equal
import java.util.*;
class Test {
    public static void main(String[] args) {
        TreeSet ts =new TreeSet();
        ts.add("A");
        ts.add("Z");
        ts.add("L ");
        ts.add("K");
        ts.add("B");
        System.out.println(ts);
    }
}
```

Comparable (I):

Comparable Interface Present in java.lang Package and it contains Only One Method compareTo().

obj1.compareTo(obj2)

Returns –ve if and Only if obj1 has to Come Before obj2.

Returns +ve if and Only if obj1 has to Come After obj2.

Returns 0 if and Only if obj1 and obj2 are Equal.

eg#1.

```
System.out.println("A".compareTo("Z")); // -ve value
```

```
System.out.println("Z".compareTo("K")); // +value
```

```
System.out.println("Z".compareTo("Z")); // zero
```

```
System.out.println("Z".compareTo(null));//NPE
```

Whenever we are Depending on Default Natural Sorting Order and if we are trying to Insert Elements then Internally JVM will Call compareTo() to Identify Sorting Order.

```
TreeSet t = new TreeSet();
t.add("K");
t.add("Z"); "Z".compareTo("K");
t.add("A"); "A".compareTo("K");
t.add("A"); "A".compareTo("A");
System.out.println(t);//[A,K,Z] => Sorting is ascending order
```

Note:

For String default natural sorting order is "Ascending order".

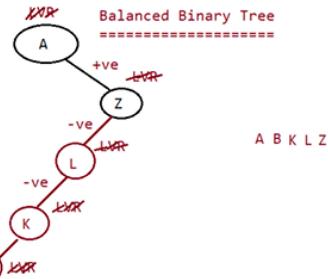
For Number default natural sorting order is "Ascending order"

```
System.out.println("A".compareTo("Z")); // -ve value
System.out.println("Z".compareTo("K")); // +ve value
System.out.println("Z".compareTo("Z")); // 0
```

Rules while constructing a binary tree

-ve means in binary tree the node should be to the left
+ve means in binary tree the node should be to the right
zero means in binary tree the nodes are duplicated

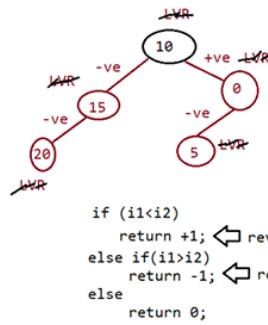
```
TreeSet ts =new TreeSet();
ts.add("A");
ts.add("Z"); "Z".compareTo("A");
ts.add("L"); "L".compareTo("A"); "L".compareTo("Z");
ts.add("K"); "K".compareTo("A"); "K".compareTo("Z");
"K".compareTo("L");
ts.add("B"); "B".compareTo("A"); "B".compareTo("Z")
"B".compareTo("L"); "B".compareTo("K");
System.out.println(ts); [ A B K L Z ]
```



```
TreeSet ts =new TreeSet(new MyComparator());
ts.add(10);
ts.add(0); compare(0,10)
ts.add(15); compare(15,10)
ts.add(5); compare(5,10)
compare(5,0)

ts.add(20); compare(20,10)
compare(20,15)
ts.add(20); compare(20,10)
compare(20,15)
compare(20,20)

20 15 10 5 0
```



compare(Object obj1, Object obj2)
|=> returns -ve value, if obj1 has to come before obj2
|=> returns +ve value, if obj1 has to come after obj2
|=> returns 0 value, if both obj1 and obj2 are equal

```
if (i1 < i2)
    return +1; ⇫ reversing the logic
else if (i1 > i2)
    return -1; ⇫ reversing the logic
else
    return 0;
```

Comparator(I)

=====

Note: If we are Not satisfied with Default Natural Sorting Order OR if Default Natural Sorting Order is Not Already Available then

we can Define Our Own Sorting by using Comparator Object.

```
public interface java.util.Comparator<T> {
    public abstract int compare(T, T);
    public abstract boolean equals(java.lang.Object);
}
```

Comparator (I):

This Interface Present in java.util Package.

Methods: It contains 2 Methods compare() and equals().

public int compare(Object obj1, Object obj2);

Returns -ve if and Only if obj1 has to Come Before obj2.

Returns +ve if and Only if obj1 has to Come After obj2.

Returns 0 if and Only if obj1 and obj2 are Equal.

public boolean equals(Object o);

Whenever we are implementing Comparator Interface Compulsory we should Provide Implementation for compare().

Implementing equals() is Optional because it is Already Available to Our Class from Object Class through Inheritance.

```
import java.util.*;
class TreeSetDemo {
    public static void main(String[] args) {
        TreeSet t = new TreeSet(new MyComparator());//line-1
        t.add(10);
        t.add(0);
        t.add(15);
        t.add(5);
        t.add(20);
        t.add(20);
        System.out.println(t);//[20, 15, 10, 5, 0]
    }
}
class MyComparator implements Comparator {
    public int compare(Object obj1, Object obj2) {
        Integer i1 = (Integer)obj1;
        Integer i2 = (Integer)obj2;
        if(i1 < i2)
            return +1;
        else if(i1 > i2)
            return -1;
        else
            return 0;
    }
}
```

At Line 1 if we are Not Passing Comparator Object as an Argument then Internally JVM will Call compareTo(),

Which is Meant for Default Natural Sorting Order (Ascending Order).

In this Case the Output is [0, 5, 10, 15, 20].

At Line 1 if we are Passing Comparator Object then JVM will Call compare() Instead of compareTo().

Which is Meant for Customized Sorting (can be Ascending /Descending Order).

In this Case the Ouput is [20, 15, 10, 5, 0]

Various Possible Implementations of compare():

```
=====
public int compare(Object obj1, Object obj2) {
    Integer I1 = (Integer)obj1;
    Integer I2 = (Integer)obj2;
    return I1.compareTo(I2);
    return -I1.compareTo(I2);
    return I2.compareTo(I1);
    return -I2.compareTo(I1);
    return +1;
    return -1;
    return 0;
}
```

Output:

1. Ascending order
2. Descending order
3. Descending order
4. Ascending order
5. insertion order
6. reverse of insertion order
7. only first element will be inserted.

Write a Program to Insert String Objects into the TreeSet where the Sorting Order is of Reverse of Alphabetical Order:

```
import java.util.*;
class TreeSetDemo {
    public static void main(String[] args) {
        TreeSet t = new TreeSet(new MyComparator());
        t.add("sachin");
        t.add("ponting");
        t.add("sangakkara");
        t.add("fleming");
        t.add("lara");
        System.out.println(t);
    }
}
```

```
class MyComparator implements Comparator {
    public int compare(Object obj1, Object obj2) {
        String s1 = obj1.toString();
        String s2 = (String)obj2;
        return s2.compareTo(s1);
        //return -s1.compareTo(s2);
    }
}
```

```
}
```

Write a Program to Insert StringBuffer Objects into the TreeSet where Sorting Order is Alphabetical Order.

```
import java.util.*;
class TreeSetDemo {
    public static void main(String[] args) {
        TreeSet t = new TreeSet(new MyComparator1());
        t.add(new StringBuffer("A"));
        t.add(new StringBuffer("Z"));
        t.add(new StringBuffer("K"));
        t.add(new StringBuffer("L"));
        System.out.println(t);
    }
}

class MyComparator1 implements Comparator {
    public int compare(Object obj1, Object obj2) {
        String s1 = obj1.toString();
        String s2 = obj2.toString();
        return s1.compareTo(s2); // [A, K, L, Z]
    }
}
```

Write a Program to Insert String and StringBuffer Objects into the TreeSet where Sorting Order is Increasing Length Order.

If 2 Objects having Same Length then Consider their Alphabetical Order:

eg: A,ABC,AA,XX,ABCE,A

output: A,AA,XX,ABC,ABCE

```
import java.util.*;
class TreeSetDemo {
    public static void main(String[] args) {
        TreeSet t = new TreeSet(new MyComparator());
        t.add("A");
        t.add(new StringBuffer("ABC"));
        t.add(new StringBuffer("AA"));
        t.add("XX");
        t.add("ABCE");
        t.add("A");
        System.out.println(t);
    }
}

class MyComparator implements Comparator {
    public int compare(Object obj1, Object obj2) {
```

```
String s1 = obj1.toString();
String s2 = obj2.toString();
int i1 = s1.length();
int i2 = s2.length();
if(i1 < i2) return -1;
else if(i1 > i2) return 1;
else return s1.compareTo(s2);
}
}
```

Note:

if we are use TreeSet(), then the condition is

- a. Object should be homogenous.
 - b. Object should be comparable(class should implement Comparable(I)).
- if we are use TreeSet(Comparator c) then what is the condition?
- a. Object need not be homogenous.
 - b. Object need not implement Comparable.

When to go for Comparable interface and When to go Comparator interface?

Ans. Predefined Comparable classes like String,Wrapper class =====> Default natural sorting is already available

if we are not happy with natural sorting order, we want customization then we need to go for "Comparator(I)".

For Predefined Non-Comparable class like StringBuffer => Comparator(I) is used for both natural sorting order and

and customized sorting order.

For userdefined class like Employee,Student =====> Developer if he comes up with own logic of sorting,then he should

implement Comparable(I) and give it as a ready made logic.

Nitin.M

=====

```
class Employee implements Comparable
{
int id;
String name;
int age;
public int compareTo(Object obj){
//sorting is done based on "id"
::::
}
}
```

If the developer who is using Employee class, if he is not interested with sorting based on "id" given by the api, then he can

use "Comparator".

When we go for Comparable and When we go for Comparator:

Comparable Vs Comparator:

=> For Predefined Comparable Classes (Like String) Default Natural Sorting Order is Already Available. If we are Not satisfied

with that we can Define Our Own Sorting by Comparator Object.

=> For Predefine Non- Comparable Classes (Like StringBuffer) Default Natural Sorting Order is Not Already Available.

If we want to Define Our Own Sorting we can Use Comparator Object.

=> For Our Own Classes (Like Employee) the Person who is writing Employee Class he is Responsible to Define Default Natural

Sorting Order by implementing Comparable Interface.

=> The Person who is using Our Own Class if he is Not satisfied with Default Natural Sorting Order he can Define his Own

Sorting by using Comparator Object.

If he is satisfied with Default Natural Sorting Order then he can Use Directly Our Class.

Write a Program to Insert Employee Objects into the TreeSet where DNSO is Based on Ascending Order of EmployeeId and

Customized Sorting Order is Based on Alphabetical Order of Names:

DNSO -> Default Natural Sorting Order

```
import java.util.*;
class Employee implements Comparable {
String name;
int eid;
Employee(String name, int eid) {
this.name = name;
this.eid = eid;
}
public String toString() { return name+"----"+eid;}
public int compareTo(Object obj) {
int eid1 = this.eid;
Employee e = (Employee)obj;
int eid2 = e.eid;
if(eid1 < eid2) return -1;
else if(eid1 > eid2) return 1;
else return 0;
}
}
class Test {
public static void main(String[] args) {
Employee e1 = new Employee("sachin", 10);
Employee e2 = new Employee("ponting", 14);
Employee e3 = new Employee("lara", 9);
```

```
Employee e4 = new Employee("flintoff", 17);
Employee e5 = new Employee("anwar", 23);
TreeSet t = new TreeSet();
t.add(e1);
t.add(e2);
t.add(e3);
t.add(e4);
t.add(e5);
System.out.println(t);
TreeSet t1 = new TreeSet(new MyComparator());
t1.add(e1);
t1.add(e2);
t1.add(e3);
t1.add(e4);
t1.add(e5);
System.out.println(t1);
}
}
class MyComparator implements Comparator {
public
int compare(Object obj1, Object obj2) {
Employee e1 = (Employee) obj1;
Employee e2 = (Employee) obj2;
String s1 = e1.name;
String s2 = e2.name;
return s1.compareTo(s2);
}
}
import java.util.Comparator;
import java.util.TreeSet;

class MyComparator implements Comparator
{
    @Override
    public int compare(Object obj1, Object obj2)
    {
        //customization => sort based on name
        return 0;
    }
}
class Employee implements Comparable
{
    String name;
```

```
int eid;

Employee(String name,int eid) {
    this.name = name;
    this.eid = eid;
}

public String toString()
{
    return name+"==>"+eid;
}

@Override
public int compareTo(Object obj1) {
    //DNSO -> based on id, sort the objects
    int id1 = this.eid;

    Employee e = (Employee)obj1;
    int id2 = e.eid;

    if(id1<id2)
        return -1;
    else if(id1>id2)
        return +1;
    else
        return 0;
}

}
}

public class ComparaTo1 {
    public static void main(String arg[])
    {
        TreeSet ts = new TreeSet();
        Employee e1 = new Employee("sachin", 10);
        Employee e2 = new Employee("ponting ", 14);
        Employee e3 = new Employee("gayle", 99);
        Employee e4 = new Employee("develliers", 17);
        ts.add(e1);
        ts.add(e2);
        ts.add(e3);
        ts.add(e4);
```

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

Comparison of Comparable and Comparator:

Comparable(I)

Present in java.lang Package

It is Meant for Default Natural Sorting Order.

Defines Only One Method compareTo()

All Wrapper Classes and String Class implements Comparable Interface.

Comparator(I)

Present in java.util Package

It is Meant for Customized Sorting Order.

Defines 2 Methods compare() and equals().

The Only implemented Classes of Comparator are Collator and RuleBaseCollator.

CodeSnippets

2:57:00

Question

Consider below code:

```
public class Test {
    static Double d1; // d1 =null
    static int x = d1.intValue(); // null.intValue() ---> NullPointerException
    public static void main(String[] args) {
        System.out.println("HELLO");
    }
}
```

On execution, does Test class print "HELLO" on to the console?

A.Yes HELLO is printed on the console

B.NO Hello is not printed on the console

Answer: B

Question

Consider below code:

```
public class Test {
    static Double d1; // static variable ====> d1 =null
    int x = d1.intValue(); // instance variable ====> only upon creating an object
    public static void main(String[] args) {
        System.out.println("HELLO"); //HELLO
    }
}
```

On execution, does Test class print "HELLO" on to the console?

- A.Yes HELLO is printed on the console
- B.NO Hello is not printed on the console

Answer: A

Question:

What will be the result of compiling and executing Test class?

```
public class Test {  
    public static void main(String[] args) {  
        Error obj = new Error();  
        boolean flag1 = obj instanceof RuntimeException; //Line n1  
        boolean flag2 = obj instanceof Exception; //Line n2  
        boolean flag3 = obj instanceof Error; //Line n3  
        boolean flag4 = obj instanceof Throwable; //Line n4  
        System.out.println(flag1 + ":" + flag2 + ":" + flag3 + ":" + flag4);  
    }  
}
```

- A. Compilation Error
- B. false:false:true:true
- C. false:true:true:true
- D. true:true:true:true
- E. false:true:true:false

Note: Error and RunTimeException no relation in hierarchy as parent and child

Error and Exception no relation in hierarchy as parent and child

Answer: A

```
String s = "sachin";  
class Student{}  
Student s1 = new Student();  
System.out.println( s instanceof String);//true  
System.out.println( s instanceof StringBuffer);//CE(String and StringBuffer no  
relationship)  
System.out.println( s1 instanceof Runnable);//false  
System.out.println( null instanceof StringBuilder);//false
```

Fill in the blanks for the definition of java.lang.Error class:

```
public class java.lang.Error extends _____ {...}
```

- A. RunTimeException
- B. Exception
- C. Throwable

Answer: C

Question>

Given code of Test.java file:

```
public class Test {  
    public static void main(String[] args) {  
        System.out.println(new RuntimeException()); //Line n1  
        System.out.println(new RuntimeException("HELLO")); //Line n2
```

```
    System.out.println(new RuntimeException(new
RuntimeException("HELLO"))); //Line n3
}
}
```

Does above code compile successfully?

A. Yes

B. No

Answer: A

Question>

Given code of Test.java file:

```
interface ILogger {
    void log();
}

public class Test {
    public static void main(String[] args) {
        ILogger [] loggers = new ILogger[2]; //Line n1 ==JVM ==> loggers[0] =null;
        loggers[1] =null;
        for(ILogger logger : loggers)
            logger.log(); //Line n2 =====> NullPointerException
    }
}
```

What will be the result of compiling and executing Test class?

A. Line n1 causes compilation error

B. Line n2 causes compilation error

C. Exception is thrown at runtime

D. No output is displayed but program terminates successfully.

Answer: C

valid

=====

```
class MyThread1 implements Runnable{}
class MyThread2 implements Runnable{}
Runnable[] runnable = new Runnable[2];
runnable[0] =new MyThread1();
runnable[1] =new Mythread2();
```

75)22-12-22

(Functional Interface And Stream API)

76)23-12-22

```
(Stream API)
class Sample
{
    private String s;
    Sample(String s){
        this.s = s;
        System.out.println("Constructor executed...."+s);
    }
}
@FunctionalInterface
interface Interf
{
    public Sample get(String s);
}
public class Test {
    public static void main(String[] args){
        Interf i = s -> new Sample(s);
        i.get("from lambda expression... ");
        System.out.println();
        //constructor reference
        Interf i1 = Sample::new;
        i1.get("from constructor reference.... ");
    }
}
=====
Example of Method reference
=====
@FunctionalInterface
interface Interf
{
    public void m1(int i);
}
public class Test {
    //logic coded by other developer
    public void m2(int i){
        System.out.println(i*i);
        System.out.println("logic coming from method reference... ");
    }
    public static void main(String[] args){
        Interf i = x-> System.out.println(x);
        i.m1(10);
        System.out.println();
        //method reference(binding the body of m2() to abstract method
```

```

m1)
}
Interf i1 = new Test()::m2;
i1.m1(20);
}
Eg:To demonstrate the usage of forEach() to print the elements of ArrayList
=====
import java.util.*;
import java.util.function.*;
// public void forEach(java.util.function.Consumer<? super E>);
// public abstract void accept(T t)
class MyConsumer implements Consumer<String>
{
@Override
public void accept(String name){
System.out.println("accept method got called... ");
System.out.println(name);
}
}
public class Test {
public static void main(String[] args){
ArrayList<String> names = new ArrayList<String>();
names.add("sachin");
names.add("dhoni");
names.add("kohli");
names.add("dravid");
//Traditional approach
Consumer<String> consumer = new MyConsumer();
names.forEach(consumer);
System.out.println();
//lambda expression
names.forEach(name->System.out.println(name));
System.out.println();
//method reference
names.forEach(System.out::println);
}
}

```

Stream API

=====

Stream ----> Channel through which there is a free flow movement of data.

Streams

To process objects of the collection, in 1.8 version Streams concept introduced.

What is the differences between Java.util.streams and Java.io streams?

java.util streams meant for processing objects from the collection. ie, it

represents a stream of objects from the collection
but Java.io streams meant for processing binary and character data with respect to file.

i.e it represents stream of binary data or character data from the file .

hence Java.io streams and Java.util streams both are different.

What is the difference between collection and stream?

=> If we want to represent a group of individual objects as a single entity then We should go for collection.

=> If we want to process a group of objects from the collection then we should go for streams.

=> We can create a stream object to the collection by using stream() method of Collection interface. stream()

method is a default method added to the Collection in 1.8 version.

```
import java.util.*;
import java.util.stream.*;
public class Test {
    public static void main(String[] args){
        ArrayList<Integer> al = new ArrayList<Integer>();
        al.add(0);
        al.add(5);
        al.add(10);
        al.add(15);
        al.add(20);
        al.add(25);
        System.out.println(al); // [0, 5, 10, 15, 20, 25]
        // till jdk1.7v
        ArrayList<Integer> evenList = new ArrayList<Integer>();
        for ( Integer i1: al )
            if (i1%2==0)
                evenList.add(i1);
        System.out.println(evenList); // [0, 10, 20]
        // From JDK1.8V we use Streams
        // 1. Configuration ==> al.stream()
        // 2. Processing ==> filter(i->i
        %2==0).collect(Collectors.toList())
        List<Integer> streamList=al.stream().filter(i->
        %2==0).collect(Collectors.toList());
        System.out.println(streamList);
        streamList.forEach(System.out :: println);
    }
}
eg#2.
import java.util.*;
import java.util.stream.*;
```

```

public class Test {
    public static void main(String[] args){
        ArrayList<Integer> al = new ArrayList<Integer>();
        al.add(0);
        al.add(5);
        al.add(10);
        al.add(15);
        al.add(20);
        al.add(25);
        System.out.println(al);
        // till JDK1.7V
        ArrayList<Integer> doubleList = new ArrayList<Integer>();
        for ( Integer i1: al )
            doubleList.add(i1*2);
        System.out.println(doubleList);
        // from JDK1.8V
        // map-> for every object, if a new object has to be created then
        go for Map
        List<Integer> streamList = al.stream().map(obj
            >obj*2).collect(Collectors.toList());
        System.out.println(streamList);
        streamList.forEach(i-> System.out.println(i));
        System.out.println();
        streamList.forEach(System.out::println);
    }
}

```

=> Stream is an interface present in java.util.stream. Once we got the stream, by using that we can process objects of that collection.

We can process the objects in the following 2 phases

1.Configuration

2.Processing

1) Configuration:

We can configure either by using filter mechanism or by using map mechanism.

Filtering:

We can configure a filter to filter elements from the collection based on some boolean condition by using filter() method of Stream interface.

public Stream filter(Predicate<T> t)

here (Predicate<T> t) can be a boolean valued function/lambda expression

Ex:

Stream s = c.stream();

Stream s1 = s.filter(i -> i%2==0);

Hence to filter elements of collection based on some Boolean condition we should go

for filter() method.

Mapping:

If we want to create a separate new object, for every object present in the collection based on our requirement then we should go for map() method of Stream interface.

public Stream map (Function f);

It can be lambda expression also

Ex:

```
Stream s = c.stream();
```

```
Stream s1 = s.map(i-> i+10);
```

Once we performed configuration we can process objects by using several methods.

2) Processing

processing by collect() method

Processing by count()method

Processing by sorted()method

Processing by min() and max() methods

forEach() method

toArray() method

Stream.of()method

eg#1.

```
import java.util.*;
import java.util.stream.*;
public class Test {
    public static void main(String[] args){
        ArrayList<String> names = new ArrayList<String>();
        names.add("sachin");
        names.add("saurav");
        names.add("dhoni");
        names.add("dravid");
        names.add("kohli");
        names.add("raina");
        System.out.println(names);
        List<String> reslut = names.stream().filter(name
            >name.length()>5).collect(Collectors.toList());
        System.out.println(reslut.size());
        long count= names.stream().filter(name->name.length()>5).count();
        System.out.println("The no of objects whose string length > 5
            is ::"+count);
    }
}
import java.util.*;
import java.util.stream.*;
//Comparable(Predefined API for natural sorting order) -> compareTo(Object obj)
//Comparator(for userdefined class for customized sorting order)->
```

```

compare(Obj1,Obj2)
public class Test {
    public static void main(String[] args){
        ArrayList<Integer> al =new ArrayList<Integer>();
        al.add(10);
        al.add(0);
        al.add(15);
        al.add(5);
        al.add(20);
        System.out.println("Before sorting :: "+al);
        //using stream api
        List<Integer> resultList=
            al.stream().sorted().collect(Collectors.toList());
        System.out.println("After sorting :: "+resultList);
        List<Integer> customizedResult = al.stream().sorted((i1,i2)
            >i2.compareTo(i1)).collect(Collectors.toList());
        System.out.println("After sorting :: "+customizedResult);
    }
}

import java.util.*;
import java.util.stream.*;
//Comparable(Predefined API for natural sorting order) -> compareTo(Object obj)
//Comparator(for userdefined class for customized sorting order)->
compare(Obj1,Obj2)
public class Test {
    public static void main(String[] args){
        ArrayList<Integer> al =new ArrayList<Integer>();
        al.add(10);
        al.add(0);
        al.add(15);
        al.add(5);
        al.add(20);
        System.out.println("Array List is ::"+al);
        Object[] objArr = al.stream().toArray();
        for(Object obj: objArr)
            System.out.println(obj);
        System.out.println();
        Integer[] objArr1 = al.stream().toArray(Integer[]::new);
        for(Integer obj1: objArr1)
            System.out.println(obj1);
    }
}
eg
import java.util.*;
```

```

import java.util.stream.*;
public class Test {
public static void main(String[] args){
//Stream API ===> Collections(group of objects)
Stream s= Stream.of(9,99,999,9999,99999);
s.forEach(System.out::println);
System.out.println();
Double[] d = {10.0,10.1,10.2,10.3,10.4};
Stream s1= Stream.of(d);
s1.forEach(System.out::println);
}
}
collect()
=====

```

This method collects the elements from the stream and adding to the specified to the collection indicated (specified) by argument.

eg#1.

```

import java.util.*;
import java.util.stream.*;
public class Test {
public static void main(String[] args) {
ArrayList<String> names = new ArrayList<String>();
names.add("sachin");
names.add("saurav");
names.add("dhoni");
names.add("yuvi");
System.out.println(names);//[sachin,saurav,dhoni,yuvi]
//Predicate(I)
// public abstract boolean test(T);
List<String> result=names.stream().filter(name->name.length()>5).
collect(Collectors.toList());
System.out.println(result);
//Function(I)<T,R>
// public abstract R apply(T);
List<String> mapResult = names.stream().map(name->
name.toUpperCase());
collect(Collectors.toList());
System.out.println(mapResult);
}
}
count()
=====
```

This method returns number of elements present in the stream.

public long count()

```

import java.util.*;
import java.util.stream.*;
public class Test {
    public static void main(String[] args) {
        ArrayList<String> names = new ArrayList<String>();
        names.add("sachin");
        names.add("saurav");
        names.add("dhoni");
        names.add("yuvi");
        System.out.println(names); // [sachin, saurav, dhoni, yuvi]
        long count = names.stream().filter(name
            > name.length() > 5).count();
        System.out.println(count);
    }
}

```

III. Processing by sorted() method

If we sort the elements present inside stream then we should go for sorted() method.

The sorting can either default natural sorting order or customized sorting order specified by comparator.

sorted()- default natural sorting order
 sorted(Comparator c)-customized sorting order.

```

import java.util.*;
import java.util.stream.*;
public class Test {
    public static void main(String[] args) {
        ArrayList<Integer> al = new ArrayList<Integer>();
        al.add(10);
        al.add(20);
        al.add(0);
        al.add(5);
        al.add(25);
        al.add(15);
        System.out.println(al);
        List<Integer> result =
            al.stream().sorted().collect(Collectors.toList());
        System.out.println(result);
        List<Integer> customizedResult = al.stream().sorted((i1, i2) ->
            i1.compareTo(i2)).collect(Collectors.toList());
        System.out.println(customizedResult);
    }
}

```

IV. Processing by min() and max() methods

min(Comparator c)

returns minimum value according to specified comparator.
max(Comparator c)
returns maximum value according to specified comparator.
import java.util.*;
import java.util.stream.*;
public class Test {
 public static void main(String[] args) {
 ArrayList<Integer> al = new ArrayList<Integer>();
 al.add(10);
 al.add(20);
 al.add(0);
 al.add(5);
 al.add(25);
 al.add(15);
 System.out.println(al);
 Integer minValue = al.stream().min((i1,i2)->
 i1.compareTo(i2)).get();
 System.out.println(minValue);
 Integer maxValue = al.stream().max((i1,i2)->
 i1.compareTo(i2)).get();
 System.out.println(maxValue);
 }
}

V.forEach() method

This method will not return anything.

This method will take lambda expression as argument and apply that lambda expression for each element present in the stream.

```
import java.util.*;  
import java.util.stream.*;  
public class Test {  
    public static void main(String[] args) {  
        ArrayList<String> names = new ArrayList<String>();  
        names.add("AAA");  
        names.add("BBB");  
        names.add("CCC");  
        names.add("DDD");  
        names.stream().forEach(name -> System.out.println(name));  
        names.stream().forEach(System.out::println);  
    }  
}
```

VI.

toArray() method

We can use toArray() method to copy elements present in the stream into specified array

```

import java.util.*;
import java.util.stream.*;
public class Test {
    public static void main(String[] args) {
        ArrayList<Integer> al = new ArrayList<Integer>();
        al.add(0);
        al.add(10);
        al.add(5);
        al.add(20);
        al.add(15);
        System.out.println(al);
        Integer[] array = al.stream().toArray(Integer[]::new);
        for (Integer element : array)
            System.out.println(element);
    }
}

```

VII.Stream.of()method

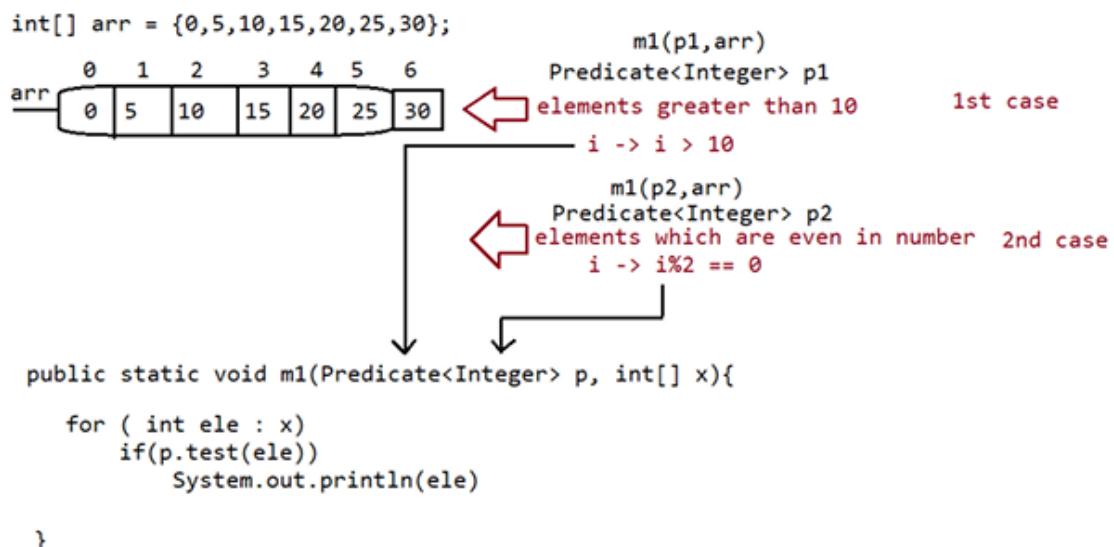
We can also apply a stream for group of values and for arrays.

Ex:

```

Stream s=Stream.of(99,999,9999,99999);
s.forEach(System.out:: println);
Double[] d={10.0,10.1,10.2,10.3};
Stream s1=Stream.of(d);
s1.forEach(System.out :: println);

```



77)02-01-23

(File Operation in JAVA io)

78)03-01-23

(File Operation)

79)04-01-23 dbt

80)04-01-23

(Serialization and Deserialization)

81)05-01-23

(Serialization and Deserialization 2)