

69 / Core Java

- * Basics of programming
- * OOPS:
 - Abstraction
 - Encapsulation
- * Constructors:
- * Interfaces and package

- * String & abstract class
- * Eclipse (IDE)
- * Arrays & collection
- * Threads
- * Exception handling
- * File I/O & Handling

70 J2EE (20 days)

JDBC :-

- Structs :-
- Servlets :-
- JSP :-

Framework (1 week)

① MVC ~~Hibernate~~

② Hibernate

API classes (1 week)

HTML

CSS

log n J

Junit/T.

→ white box tools

API - build tool.

learn by your self

- javascript
- jquery

Java 2 SE (Java for Standard edition)

Java 2 EE (Java for enterprise edition)

Java 2 ME (Java for mobile addition)

download

Disk 1.7 SE for download

JPDK development kit

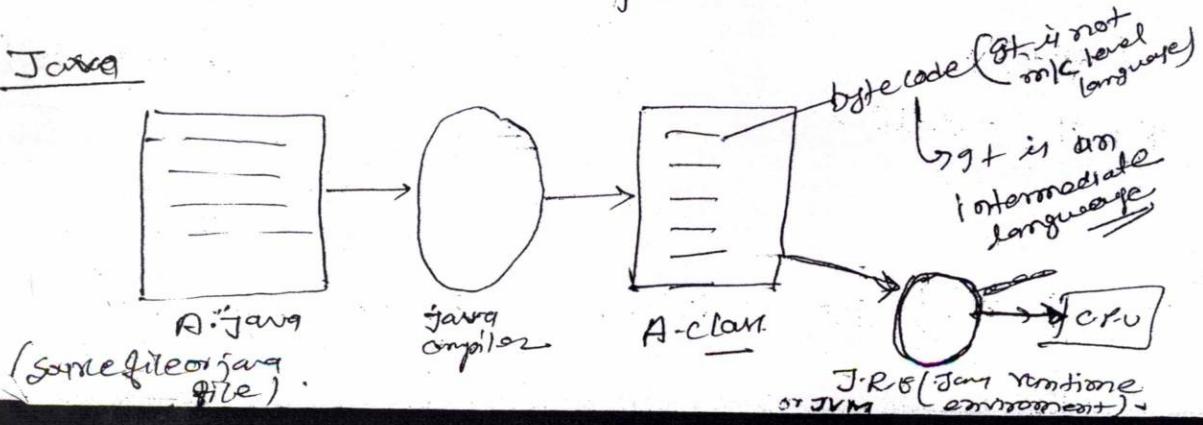
cmd
type Java

Path - [C:\program files\java\JDK1.7.0\bin] in the Environment variable

★ program is set of instruction written to achieve a task.

- * CPU understands only machine level language (low level language); which contains 0 & 1.
- * writing a program using M.L.L (machine level language) is very difficult and confusing so, developer writes a program in high level language which contains English words.
- * Since CPU understands only M.L.L, we need a translator which converts (translate) H.L.L into M.L.L.
- * Translator translate in two different ways. Some translator translate entire program into M.L.L at a shot. This process is known as compilation and the one who does compilation is known as compiler.
- * Some translator translates one line at a time. This process is known as interpretation. The one who does interpretation is known as interpreter.

Java



* To write and execute java program, there are three steps.

- (i) Development → we write java program, and save with -java : java file called as .java file
- (ii) compilation →

Using Java Compiler we have to convert source file into class file. Java Compiler compiles the source file and generates an intermediate language called byte code. Bytecode file extension is .class.

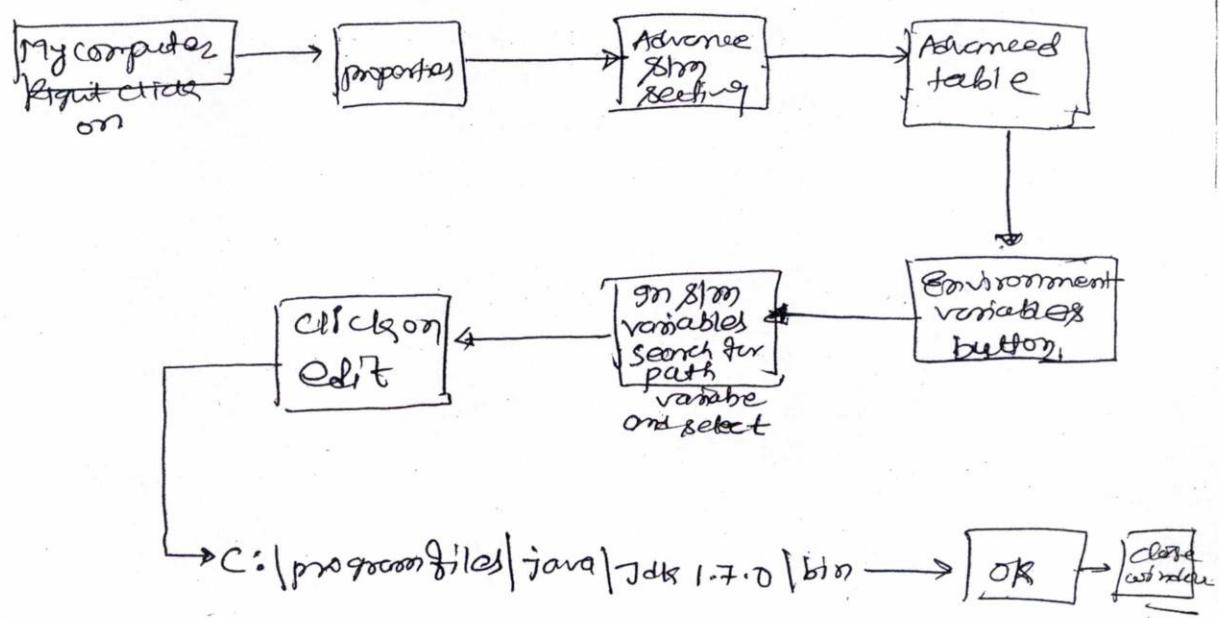
- (iii) Execution → JRE (java runtime environment) executes the byte codes or class files.

NOTE

→ To develop java program and compile, we need java compiler.

→ To execute class file, we need JRE.

Setting the path



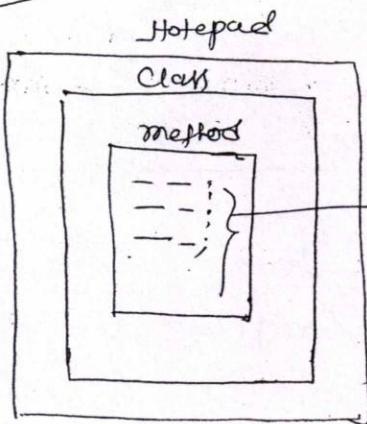
↗ Setting the environment variable path in Jdk

* to check, which version of java is installed



Date

structured class

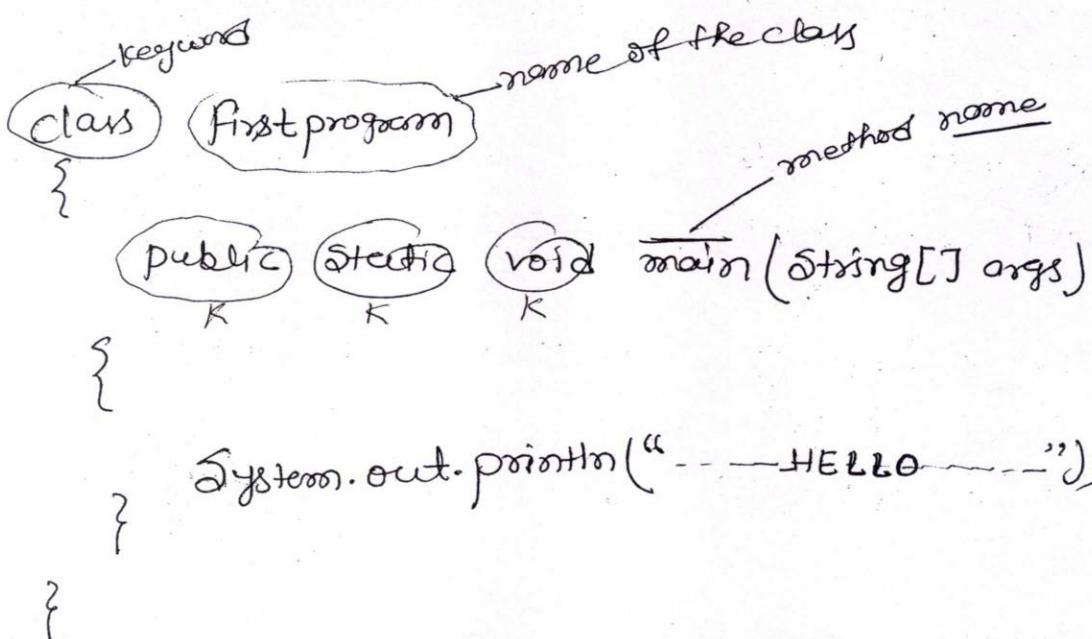


Statement

→ A Hotepad (editor),
contains n-class

→ A class ^{can} contains
n-method

→ A method can contain
n-jarg statements



* D:\ocm14\basics\firstprogram.java

* Java is case sensitive. *

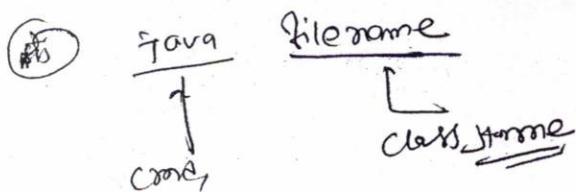
* compilation

① open cmd prompt, navigate to the folder which contains source file.

To change the drive = d:\

To change the folder = cd ocm14\basics

② javac filename.java is to compile the file



- * when we execute a class file, JRE starts execution from main's method (All the statements present in the main() one execute one after the other)

```

class AstProgram Identifier
{
    public static void main()
    {
        System.out.println("Hello");
        System.out.println("welcome to java");
    }
}

```

- * Java program is made up of keyword, identifier, operator and literals.
- * Keyword are the built in commands.
- * In Java, all keywords are in lower-case.
keyword - class, public, int, abstract, void, final etc

Identifiers

Identifiers are the names which are used to identify a block of entities. Identifiers are name given by developers or program.

- we should not use keywords as identifiers
 - Identifier must begin with alphabet, later we can use digits.
 - special characters are not allowed except _ and \$.
- Under Score

Standard naming convention ~~for class~~

⑤ for class:-

First Program

Sample Program

→ write first character of each word in uppercase

⑥ for method/variable

write first word in lowercase, and from second word onward first character in uppercase

- count
- getSize
- noOfClick
- findThe Volume

* operators

It is the symbol that is used to perform some operation.

example - , + , - , = , > etc

* Literals

- Literals are programmed data or user data.
- example string literal
integer literal etc

* Comment

→ // -----

→ /* -----

*/

* variables

- variable is a temporary place in a memory used to store temporary data.
- Based on the data type, variables are classified into two types
 - (i) primitive variables
 - (ii) reference variables
- primitive variables are the variables which contain primitive types.
- In Java, there are 8-primitive types are available.
 - (i) byte
 - (ii) short
 - (iii) int
 - (iv) long
 - (v) float
 - (vi) double
 - (vii) char
 - (viii) boolean

* Declaring a variable :-

Syntax

type variableName;

int no1;	no1	[]
double no2;	no2	[]
char ch;	no3	[]

int a=1, b=2;
int a=10

for char = '---'

int no1 = 10;

char ch = 'X';

int no1, no2, no3;

=====

4

- * String data type is derived data type
- * can we write and compile a class without main() method → yes
- * can we execute a class, which does not contain a main() method → No
because execution starts from main() method.

Date (1)
10/9/2012

operator

⇒ unary operator

$\text{++}; \text{--};$

$\text{int } i=0;$

$i++;$

$\text{s.o.p}("i=" + i); \quad // i=1$

* class Demo

{

public static void main(String[] args)

{

System.out.println("program start");

$\text{int } i=0;$

$i++;$

System.out.println("i=" + i);

$\text{int } j=10;$

$j--;$

System.out.println("j = " + j);

System.out.println("pgm ends");

? ?

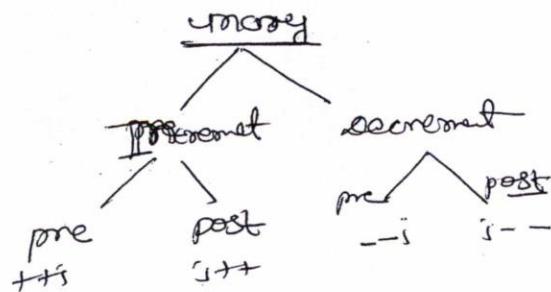
output program start

$i=1$

$j=9$

program ends

⇒



~~(++) ++~~

5

$\text{int } i=0;$

$\text{int } j=0;$

$j = ++i;$ $j = i++;$

$\text{s.o.p}(i); // 1$

$\text{s.o.p}(j); // 2$

$\text{int } i=0;$

$\text{int } j=0;$

$j = i++;$ → $j = i;$

$j = i+1;$

$\text{s.o.p}(i); // 1$

$\text{s.o.p}(j); // 2$

Class Demo7

{

public static void main(String[] args)

{

System.out.println("Program Starts"),

int i=0;

int j=0;

j = i+j; // presincrement

System.out.println("i=" + i);

{ System.out.println("j=" + j);

}

Assumptions for increment operator

int i=0;
int j=0;

j = ++i + i++; $\Rightarrow j = (\underline{++i} + \underline{i++})$ i=0

j = (0+1) + (1++) i=1

j = 1+i++;

j = 1+1++; $\boxed{j=2, i=2}$

j = 2
i = 2

Class Demo9

{

public static void main (String[] args)

{

int i=0;

int j=0;

j = ++i + i++;

System.out.println("i=" + i);

System.out.println("j=" + j);

}

Output $\Rightarrow j=2$? -
 $\Rightarrow j=2$

int i=0;

int j=0;

$$j = i++ + ++i; \quad j = i++ + ++i \quad \left. \begin{array}{l} i=0 \\ i=1 \end{array} \right\}$$

$$\begin{aligned} &= 0 + \cancel{i} \\ j &= 0 + \cancel{1} \end{aligned} \quad \left. \begin{array}{l} i=1 \\ i=2 \end{array} \right\}$$

class Demo10

{ public static void main (String[] args)

{

int i=0;

int j=0;

$$j = i++ + ++i;$$

System.out.println ("i=" + i);

System.out.println ("j=" + j);

? ?

$$\begin{aligned} &\| i=2 \\ &\| j=2 \end{aligned} \quad \left. \begin{array}{l} A \\ \equiv \end{array} \right.$$

*

int i=0;

int j=0;

$$\begin{aligned} j &= \frac{i++}{1} + \frac{+i}{2} + \frac{i++}{3} + \frac{+i}{4}; \Rightarrow \frac{i=0}{1} \\ &= 0 + 2 + 2 + 4 \end{aligned}$$

$$= \underline{i=0}, \underline{i=2}, \underline{i=3}, \underline{i=4}$$

$$\underline{j=8}$$

$$\begin{aligned} &\text{output } i=4 \\ &\text{output } j=8 \end{aligned} \quad \left. \begin{array}{l} A \\ \equiv \end{array} \right.$$

Assignment

① int i=0;
int j=0;
 $j = -i - j + i - - + - - i;$
 $= -1 + (-1) + (-3)$
 $= -5 \quad \left. \begin{array}{l} A \\ \equiv \end{array} \right.$

② int i=10;
int j=0;
 $j = ++i + --j + j - - + i + +;$
 $= 11 + 10 + 10 + 9$
 $j = 40 \quad \left. \begin{array}{l} A \\ \equiv \end{array} \right.$

11110

6

Branching Statements

① if else

② switch

if (condition)

{ — ;

} — ;

class Demo1

{

public static void main(String[] args)

{

System.out.println("Program starts");

int i = 5;

int (i >= 5);

{

System.out.println("i = " + i);

}

System.out.println("Program ends");

}

Output $\Rightarrow i = 5$

* if (condition)

{

Statement 1;

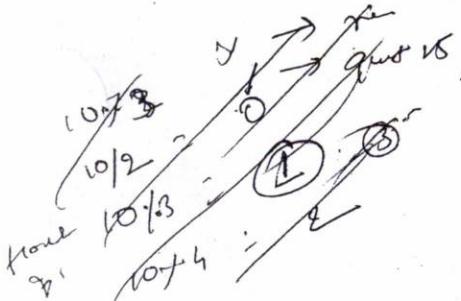
}

else (condition)

{

Statement 2;

}



class Demo2

{ public static void main (String[] args)

int s = 10,

int r = s % 2;

if (r == 0)

{
S.O.P ("No is even");

else

{
S.O.P ("No is odd");

} } .
Output No is even.

* Q. A P to check whether a person is eligible to marry.

~~public~~

class Demo3

{

 public static void main(String[] args)

{

 int age = 25;

 if (age >= 18) ~~(age <= 40)~~ {

{

 System.out.println("eligible for marriage");

}

 else

{

 System.out.println("not eligible for marriage");

}

}

My own practice

* class Myclass

{

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

 System.out.println(100);

output = 100 (no need of initialization)

~~class Myclass~~

{

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

 S.O.P(20.999);

 S.O.P(true);

 S.O.P(10+20);

 S.O.P(20 == 20); }

Output

20.999, true, 30, true

class B

③ public static void main(String[] args)

{

 int i;

 System.out.println("i");

}

1out = error (int i is not a statement).

i.e (i) is not initialized

class C

{

 public static void main
 (String[] args)

{

 int i;

 int j;

 j=i;

 System.out.println(j);

}

1out = error (variable not initialized)

class E

```
public static void main  
(String[] args)  
{  
    int p, q=10, m;  
    System.out.println(p);  
    System.out.println(q);  
    S.O.P(m);  
}  
  
output = (m, p & not initialized)
```

class F

```
public static void main  
(String[] args)  
{  
    int s;  
    s++;  
    System.out.println(s);  
}  
  
output / error (s not initialized)
```

class E

```
public static void main  
(String[] args)  
{  
    int s;  
    System.out.println(s=10);  
    System.out.println(s);  
}  
  
output = 10, 10 (we can declare local var inside S.O.P)
```

```
int i;  
S.O.P(i);  
S.O.P(i=10);  
  
// output = error (i not initialized)
```

class G

```
public static void main  
(String[] args)  
{  
    int i, j;  
    i=0;  
    j=20;  
    System.out.println(i+j);  
}  
  
1/Output = 20
```

class H

```
public static  
void main(String[] args)  
{  
    int s=10, j, k=20;  
    j=s+k  
    System.out.println(j);  
    System.out.println(j);  
    S.O.P(k);  
}  
  
Output 10, 30, 20
```

class P

```
public static void main()  
{  
    int i=0;  
    i=-; post()  
    i=++;
```

System.out.println(~~i~~)
i=0
i-- post i-- 1--
i=0
Output = 0
i=0
i=0
i=0
i=0
i=0

Output = -1*

* class F

{ public static void main
(String args)

{ int i=0; } int j=0;
i--; }
j = -i;
System.out.println(i);

} // error → undefined symbol i

class F (* declared form sir)

{ public static void main(String[] args)

* int i=10 → (if only int i
test (10, 10);
System.out ("done");

static void test (int k)

{ S.O.P (" from test;" + k);

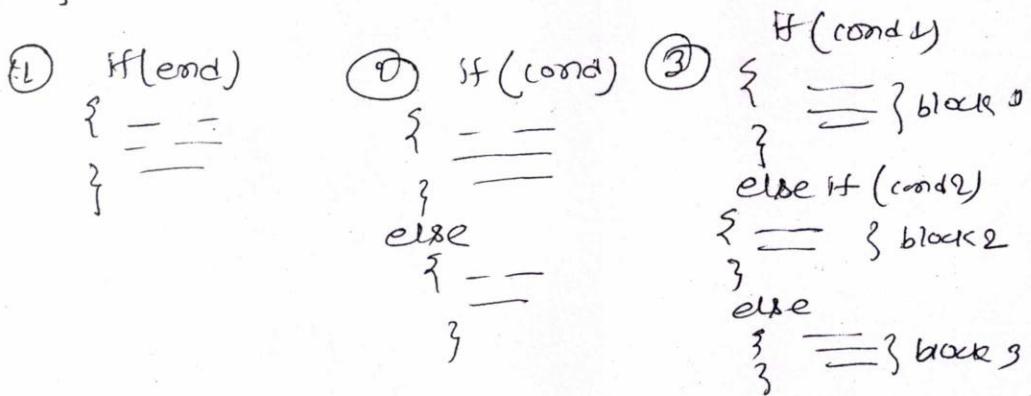
} // error - (

If int
written
here
then
also
OK
why

5

class Demo2

```
{ public static void main(String[] args)
{
    System.out.println("Program Starts");
    int i = 10;
    int r = i / 2;
    if (r == 0)
    {
        System.out.println("Number is even");
    }
    else
    {
        System.out.println("Number is odd");
    }
    System.out.println("Program ends");
}
```



* Java verifies condition 1, if cond 1 is true, block 1 will be executed, otherwise Java will verify the second condition.

If cond 2 is true, block 2 will be executed, otherwise Java verifies ~~cond 3~~ cond 3. Like this Java keeps on verifying the condition. If none of conditions are true, then else block will be executed. Any number of else if we can write and else block is not mandatory.

class Demo3

{ public static void main (String [] args)

{ char c = 'g'; ~~char c = 'g'; char c = 'y'~~

if (c == 'g')

{ System.out.println ("stop"); }

else if (c == 'y')

{ System.out.println ("HALT & PROCEED"); }

else if (c == 'g')

{ System.out.println ("PROCEED"); }

else

{ System.out.println ("Invalid color"); }

~~W.A.I.P to print grade based on percentage~~

class Demo4

{ public static void main (String [] args)

{ int i = 65;

if (i > 70)

{ System.out.println ("Grade is distinction"); }

else if (60 ≤ i < 70)

else ((i <= 60) && (i > 70))

{ System.out.println ("Grade is first class"); }

else if (50 ≤ i < 60)

{ System.out.println ("Grade is second class"); }

else if (35 ≤ i < 50)

{ System.out.println ("Third class pass"); }

9

```
else
{
    s.o.p (" fail");
}

class demo4
{
    public static void main (String [] args)
    {
        int i = 65;
        if ( i >= 70 )
        {
            System.out.println (" distinction");
        }
        else if ( !(i < 70) && (i > 60) )
        {
            System.out.println (" First class");
        }
        else if ( (i < 60) && (i >= 50) )
        {
            System.out.println (" Second class");
        }
        else if ( (i < 50) && (i > 35) )
        {
            s.o.p (" Third class");
        }
        else
        {
            s.o.p (" FAIL");
        }
    }
}
```

Switch branching statement:-

switch (condⁿ or express)

{

case values: _____;

break;

case values: _____;

_____;

break;

_____;

case values: _____;

_____;

break;

default: _____

}

class Demo5

{

public static void main(String[] args)

{

~~int~~ int i=1;

switch(i)

{

case 1:

System.out.println("SUNDAY");

break;

case 2:

System.out.println("MONDAY");

break;

case 3:

s.o.p("TUESDAY");

break;

case 4:

s.o.p("WEDNESDAY");

break;

case 5:

s.o.p("THURSDAY");

break

case 6:

s.o.p("FRIDAY");

break

case 7:

s.o.p("SATURDAY");

break

default: s.o.p("invalid number");

Output: SUNDAY

Note: If we don't use break statement until next statement it will continue execute break statement

} If we don't use break
It will continue execute next statement until find break statement.

After executing the case block, mandatory, we have to write break statement to break the execution, if we don't write break statement, then java continues to the next case and execute the statement.

In the above program,

If there is no break statement in all the cases, then, the output will be (all the the case executed (like "Monday ---- Saturday").

- * For case values, only we have to write literals. and case value should be unique.

Date
14/04/2013

(3)

$n=1-12$

1, 2, 3 - winter
4, 5, 6 - summer
7, 8, 9 - spring
10, 11, 12 - autumn

int n=3

Switch(n)

{
case 1:
case 2:

case 3: S.O.P("winter"); break;

case 4:

case 5: S.O.P("summer"); break;

↑

We can write case in one line like
case 1; case 2; case 3; case 4;

class demo6

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

int n=4

Switch(n)

case 1:
case 2:

case 3:

case 4:

S.O.P("winter");
break;

Care can write in
one line
case 1; case 2;

case 5:
case 6:
case 7:
case 8:

S.O.P("summer");
break;

case 9:
case 10:
case 11:

S.O.P("spring");
break;

case 12:

S.O.P("autumn");
break;

} default: S.O.P("Invalid num");

Output = Summer.

11

→ Use Switch, when we have more condition and when we have equality test.

Loops :-

→ 4 types of Looping /

Looping statements are used to execute the block of code repeatedly. In Java, there are four available.

for, /
while
do-while /
+ for each (Advanced for) ✓
→ Added in after 1.5 version

- ① for
 - ② while
 - ③ do while
 - ④ for each (Advocacy for)

⇒ Syntax. —

for(initialization; condition; incr/decr)

{ - - - - ; } fact statements

~~class~~ Demo1

```
{ public static void main(String args[])
{
    System.out.println("Program Starts");
    for(int i=1; i<=5; i++)
    {
        S.O.P("Java");
    }
    System.out.println("Program Ends");
}
```

output

{ poem starts
fjona
fjona
fjona
fjona
fjona
fjona
poem ends

Hole:-

- Note:-

 - only for the first time initialization parts execute.
 - In each time, java checks a condition, if condition is true, java executes the body of the loop. Other, it's stop.
 - At the end of each iterations, java executes increment or decrement part.

* Sum of 1st 20 number. (Suppose $1+2+3+ \dots + 20 = \underline{\underline{210}}$)

class remove

{ public static void main (String [] args)

{ int i; sum = 0;

for (i=1; i<=20; i++)

{ sum = sum + i; } \rightarrow (we can also write like) $\underline{\underline{\text{sum} += i}}$

System.out.println ("Sum of natural number:" + sum);

Output = 210 Ans $\frac{10}{\Sigma}$

* no. of print b/w 50 to 100

for (int i=50; i<=100; i++)

{ System.out.println (i); }

* even number between 50 & 100

for (int i=50; i<=100; i++)

{ if (i%2 == 0)

{ S.O.P (); }

Assignment

① W.A.P to display the sum of odd numbers b/w 50 to 100

② W.A.P to count number which is divisible by 3 b/w 25 to 100

③ W.A.P to print, multiplication table for any number

④ W.A.P to find factorial for a number

class factorial

{ public static void main (String [] args)

{ int fact = 1; 1x2x3x...x5

for (int i=1; i<=n; i++)

{ fact = fact * i; \Rightarrow (or fact *= i) }

} system.out.println ("factorial of n is = " + fact);

* Use of break in for loop:

class Demo4

{ public static void main (String args[])

{ for (int i=1; i<=10; i++) (for int i=1; i<=10; i++)

{ system.out.println (i);

if (i == 3)

{ break; }

}

~~good~~ for (int i=1; i<=10; i++)

{ system.out.println (i);

if (i > 5)

{ break;

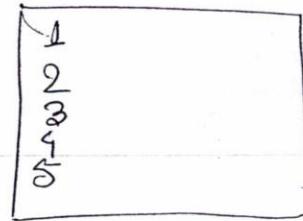
}

1	—
2	✓
3	✓
4	✓
5	✓
6	✓

..

output 1, 2, 3, 4, 5, 6

~~good~~
~~int (i=1; i<=20; i++)~~
{
 ~~if (j>5)~~
 {
 ~~break;~~
 }
}
S.O.P(i);



Output 1, 2, 3, 4, 5

~~write a program to check whether a number is prime or not~~

~~bad~~

class prime
{
 public static void main(String args[]){
 {
 int i, n=7;
 for(; i<n; i++)
 {
 if(n % i == 0)
 {
 System.out.println("number is not prime");
 break;
 }
 else
 {
 System.out.println("number is prime");
 }
 }
 }
 }
}

Here we use some reference variable
set a=1
in if (---)
 { a>=0;
 3---;
 if(a==1)
 { not prime}

$$\begin{aligned} 7 \times 1 &= 7 \\ 7 \times 3 &= 21 \\ 7 \times 4 &= 28 \\ 7 \times 5 &= 35 \\ 7 \times 6 &= 42 \\ 10 \times 2 &= 20 \end{aligned}$$

2000

→ 1000

class prime
{ public static void main(String args){

 int i, n=7
 for(; i<n; i++)
 { if(n % i == 0)
 { count++;
 }
 }
}

(to make efficient, add

 {
 if (count==0)
 {
 S.O.P("prime");
 }
 else
 {
 S.O.P("not prime");
 }
 }
}

13

Assignment program (divisible by 3 from 25 to 100)

① class demo1

```
{
    public static void main(String args[])
    {
        int i; count=0;
        for(i=25; i<=100; i++)
        {
            if(i%3==0)
            {
                count = count+1;
            }
        }
        System.out.println("total count is:" + count);
    }
}
```

② class MultiplicationTable

```
{
    public static void main(String args[])
    {
        int i; int a=0;
        int n=7;
        System.out.println("multiplicationtable of n is:" + n);
        for(i=0; i<=10; i++)
        {
            a=n*i;
            System.out.println(a);
        }
    }
}
```

S.O.P
 $n \times i = a$

③ class oddnumbersSum;

```
{
    public static void main(String args[])
    {
        int i=0;
        int sum=0;
        for(i=50; i<=100; i++)
        {
            if(i%2==1)
            {
                sum = sum+i;
            }
        }
        System.out.println("The sum is:" + sum);
    }
}
```

Date
15/04/2012

Nested for loop:-

```
for(int i=1; i<=3; i++)
{
    for(int j=1; j<=3; j++)
        S.O.P(" "+i+j);
}
```

Output :- 11 21 31
12 22 32
13 23 33.

Note :- Here, concatenation occurs,
due to starting with null character
so, it is not adding)

called null string.
Tried to concatenate, but not to add.

println - It adds new line
print - It does not add <u>line</u>

Class Demo5

```
{ public static void main(String args[])
{
    System.out.println("program starts");
    for(int i=1; i<=3; i++)
    {
        for(int j=1; j<=3; j++)
            System.out.println(" "+i+j);
    }
    System.out.println("program ends");
}
```

Output :- 11 21 31
12 22 32
13 23 33.

Class Demo6

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

```
{ S.O.P("program starts");
    for(int i=1; i<=3; i++)
    {
        System.out.print("X");
        System.out.print("\n");
    }
}
```

14
This is for
next line

```
for(int i=1; i<=3; i++)
{
    System.out.print("X");
}
```

Output :-

Program

```

class Demo
{
    public static void main(String args[])
    {
        for(int j=1; j<=4; j++)
        {
            System.out.print("X");
            System.out.print("");
        }
    }
}

```

Block

```

        for(int j=1; j<=4; j++)
        {
            System.out.print("X");
            System.out.print("");
        }

```

Repeat upper block 3 time more

====

?

====

?

Output

```

X***X
*KKK*
KKKK
*KKK*

```

so, In upper code, same block repeated for
3 times, more, for upper output for

so To avoid, we nested for loop

```

class Demo
{
    public static void main(String args[])
    {
        for(int i=1; i<=4; i++)
        {
            for(int j=1; j<=i; j++)
            {
                System.out.print("X");
                System.out.print("");
            }
        }
    }
}

```

Output

```

XXXX
XXX
XX
X

```

class demo7

{

public static void main (String args [])

{

for (int i=1; i<=4; i++)

{

for (int j=1; j<=i; j++)

{

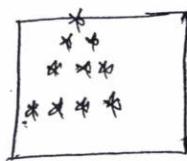
System.out.print("X");

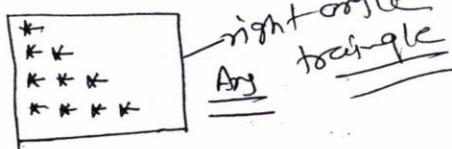
}

} System.out.println("");

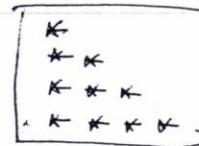
};

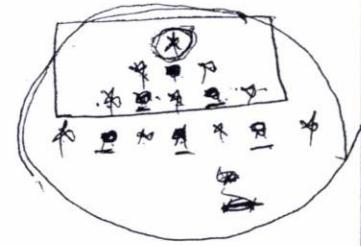
Output



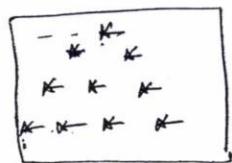


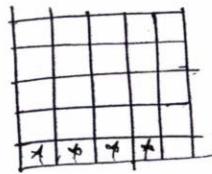
right angled
Any triangle

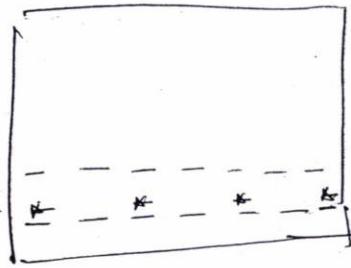
O/P


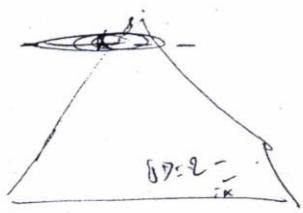


~~Assymetrical~~









B7:2

class demo7

{

public static void main (String [] args)

{

for (int i=1; i<=4; i++)

{

for (int j=1; j<=i; j++)

{

System.out.print(" ");

}

System.out.println(" ");

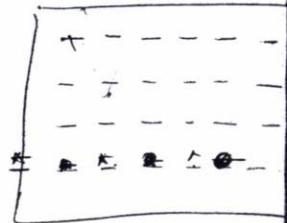
System.out.println(" ");

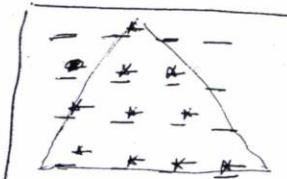
{

for (int k=1; k<=i; k++)

* * *







G10

G11

G12

G13

G14

G15

G16

G17

G18

G19

G20

G21

G22

G23

G24

G25

G26

G27

G28

G29

G30

G31

G32

G33

G34

G35

G36

G37

G38

G39

G40

G41

G42

G43

G44

G45

G46

G47

G48

G49

G50

G51

G52

G53

G54

G55

G56

G57

G58

G59

G60

G61

G62

G63

G64

G65

G66

G67

G68

G69

G70

G71

G72

G73

G74

G75

G76

G77

G78

G79

G80

G81

G82

G83

G84

G85

G86

G87

G88

G89

G90

G91

G92

G93

G94

G95

G96

G97

G98

G99

G100

* class Demo8

```
{ public static void main(String args[])
{
    for(int i=1; i<=5; i++)
    {
        for(int j=1; j<=5; j++)
        {
            System.out.print("*");
            if(i==3) {break;}
        }
        System.out.println("");
    }
}
```

Analysis
Output

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

* class Demo9

```
{ public static void main(String args[])
{
    for(int i=1; i<=5; i++)
    {
        for(int j=1; j<=5; j++)
        {
            System.out.print("*");
            if(j==3) {break;}
        }
        System.out.println("");
    }
}
```

O/P

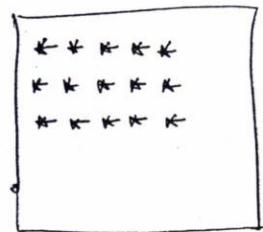
```
***  
* * *  
* * *  
* * *  
* * *
```

```

class demo7
{
    public static void main( String args[])
    {
        for( int i = 1; i <= 5; i++)
        {
            for( int j = 1; j <= 5; j++)
            {
                System.out.print("X");
            }
            System.out.println();
            if( i == 3) { break; }
        }
    }
}

```

6/8



* while loop

```

while( cond)
{
    --;
    --;
    --
}

```

Ex. int i = 1;
 while(i <= 10)
 { S.O.P(i);
 i++; }

class demo8

```

public static void main( String args[])
{
    System.out.println("program starts");
    int i = 1;
    while( i <= 10)
    {
        System.out.println("i");
        i++;
    }
}

```

O/P

$\frac{1}{2}$
 $\frac{3}{4}$
 $\frac{5}{6}$
 $\frac{7}{8}$
 $\frac{9}{10}$

16

do-while loop

```
do {  
    ____;  
    ____;  
} while(condition);
```

for example
take some example
restart or stop: (at comes 1 time
play after)

```
class demo9  
{ public static void main( String args[] )  
{ System.out.println("Program starts");  
    int i = 1;  
    do {  
        System.out.println("i = " + i);  
        i++;  
    } while(i <= 10);  
    System.out.println("Program ends");  
}
```

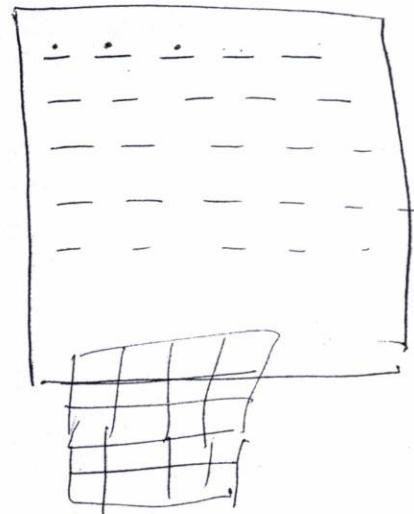
	0	1	2	3	4	5	6	7	8
0	*								
1	*	*	*	*	*	*	*	*	
2	*	*	*	*	*	*	*		
3	*	*	*	*	*	*	*		

(L-1)

L-2 + L-1

```
for(i=0; i<=n; i++)
{
    for(j=0; j<=n; j++)
        cout << " ";
    for(k=1; k<=(2*i)-1; k++)
        {
            if(k%2==0)
                cout << "*";
            else
                cout << " ";
        }
    cout << endl;
}
```

```
for(i=0; i<=n; i++)
{
    for(j=0; j<=n; j++)
        cout << " ";
    for(k=1; k<=(2*i)-1; k++)
        {
            if(k%2==0)
                cout << "*";
            else
                cout << " ";
        }
}
```



~~if(i>0) { for(j=0; j<=i; j++) a[i][j] = { '*' }; }~~

class equitriangle

```
{ public static void main(String[] args)
{
    for(i=0; i<=n; i++)
    {
        for(j=0; j<=n-i; j++)
            cout << " ";
        for(k=1; k<=(2*i)-1; k++)
        {
            if(k%2==0)
                cout << "*";
            else
                cout << " ";
        }
    }
}
```

17

equilateral triangle (Assignment)

class equilateral

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

for (int i=1; i<=4; i++)

{ for (int j=1; j<=4-i; j++)

{ System.out.print (" ");
}

for (int k=1; k<=(2*i)-1; k++)

{

if (k%2 == 1)

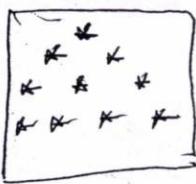
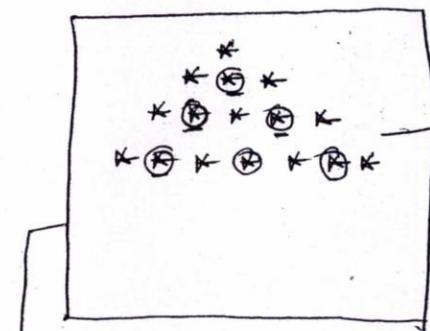
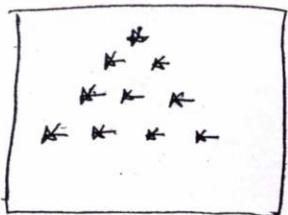
{ System.out.print ("*");
}

else

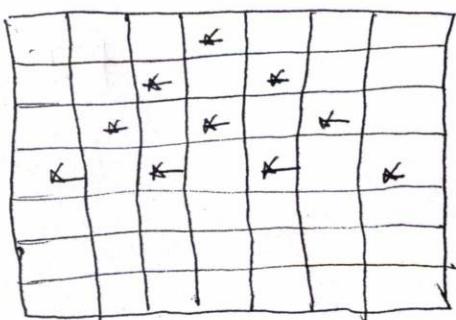
{ System.out.print (" ");
}

} System.out.println ("");

Output



first print the upper
equilateral triangle, then
add space in place of O (marked
dot).

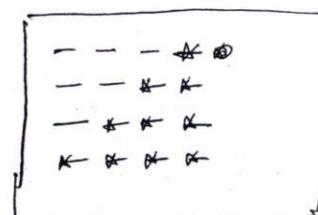
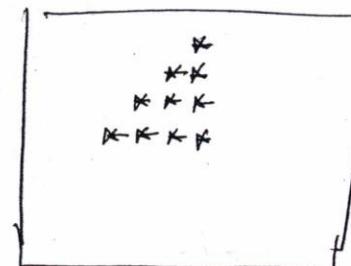


Assignment 2

```
class B
{
    public static void main(String args[])
    {
        for(int i=1; i<=4; i++)
        {
            for(int j=1; j<=4-i; j++)
            {
                System.out.print(" ");
            }
            for(int k=1; k<i; k++)
            {
                System.out.print("*");
            }
            System.out.println("");
        }
    }
}
```

JWx livecombe
written also
like
`for(int j=i;
 j<=n; j++)`

Output



Date
16/04/2013

for each (Advanced for)

It is used for only arrays & collections

Methods :-

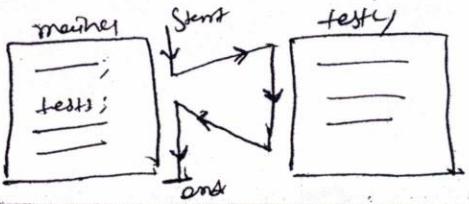
- It is functions / sub-routines.
- It is group of statement return to achieve a simple task.
- The main advantage of method is reusability.
- Once we write a method, we can call and execute that method n, number of times.
- A method can take some inputs and return some output.

class Demo1

```
{ public static void main(String[] args)
{
    System.out.println("main starts");
    add(); // invoke add method
    System.out.println("main ends");
}

static void add()
{
    System.out.println("add() Starts");
    int n1=10;
    int n2=20;
    int res=n1+n2;
    System.out.println("res=" + res);
}
System.out.println("pgm ends");
```

Output {
main Starts
add Starts
res=30
add ends
pgm ends}



class example

c++ plus plus
software

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

```
{  
    System.out.println("main starts");
```

```
    test1();
```

```
    test2();
```

```
    System.out.println("main ends");
```

```
}  
static void test1()
```

```
{  
    System.out.println("test1() starts");
```

```
    System.out.println("test1() ends");
```

```
}
```

```
static void test2()
```

```
{  
    System.out.println("test2() starts");
```

```
    System.out.println("test2() ends");
```

```
}
```

output

main starts
test1() starts
test1() ends
test2() starts
test2() ends

* we can call a method inside a method

* java allows two main method exists. in class.

example

```
static void test1()
```

```
{  
    System.out.println("test1() starts");
```

```
    test2();
```

```
    System.out.println("test1() ends");
```

19

```

class Demo3
{
    public static void main(String[] args)
    {
        System.out.println("main starts");
        add(10, 20);
        add(30, 40);
        System.out.println("main ends");
    }
}

```

```

static void add(int n1, int n2)
{
    int res = n1 + n2;
    System.out.println("Res=" + res);
}

```

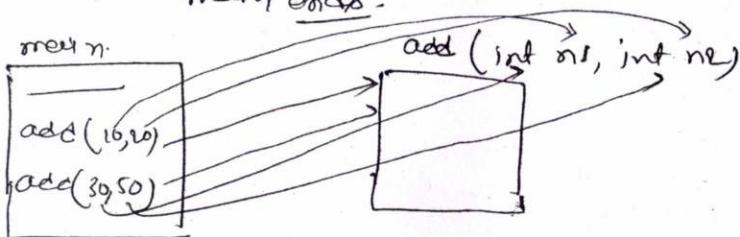
Output

main starts

res = 20

res = 70

main ends.



class Demo4

```

public static void main(String args[])
{
    System.out.println("program starts");
    int a1 = add(10, 20);
    System.out.println("a1 = " + a1);
    System.out.println(add(30, 50));
    System.out.println("program ends");
}

```

```

static int add(int n1, int n2)
{
    return n1 + n2;
}

```

Output

program starts

a1=20

80

program ends

Assignment

- ① → create a class calculate with basic operations.
- ② → create a class and create a method which should check a given number is prime or not. method should true if the no. of is prime otherwise false.

Assignment-1

class calculator

{ public static void main(String [] args)

{ System.out.println("Calculator program starts");

int num1, num2;

int a1 = add (num1, num2);

int a2 = sub (num1, num2);

int a3 = mul (num1, num2);

int a4 = div (num1, num2);

System.out.println("a1=" + a1);

System.out.println("a2=" + a2);

System.out.println("a3=" + a3);

System.out.println("a4=" + a4);

}

static int add(int num1, int num2)

{ return (num1+num2);

static int sub(int num1, int num2)

{ return (num1-num2);

static int mul(int num1, int num2)

{ return (num1*num2);

static int div(int num1, int num2)

{ return (num1/num2);

Assignment
class prime

```
{  
    public static void main(String[] args)  
    {  
        System.out.println("pgm starts");  
        int n=11;  
        System.out.println("prime(n)");  
        System.out.println("prime ends");  
    }  
}
```

static int primeno(int n)

```
{  
    int count=0;  
    if(n > 2 == 0)  
        for(int s=2; s<n; s++)  
    {  
        if(n*s==0)  
    }
```

count = count+1;

~~return (false);~~

break;

~~else~~

if(count>0)

```
{  
    return (false);  
}
```

else

```
{  
    return (true);  
}
```

}

```

class Demo4
{
    public static void main (String [] args)
    {
        S.O.P ("prm starts");
        S.O.P (checkPrime (7));
        S.O.P (checkPrime (10) + checkPrime (10));
        S.O.P ("prm ends");
    }
}

```

```
static checkPrime (int n)
```

```

{
    boolean flag true;
    for (i=2; i<n; i++)
    {
        if (n % i == 0)
        {
            flag = false;
            break;
        }
    }
    return (flag);
}

```

output

prm starts
true
is 10 prime? false
prm ends

In Java, every should contain return statement.
even if it is void type, then also should have return.

but in void method if we don't write return statement, because, defaultly Java compiler write the return statement during compilation.

but other than void, It gives 21

errors. But in void method Java, writes during compilation "return" into byte code.

Date
17/04/2018

→ methods are similar to functions one

Sub-routines-

→ method is a group of statement return, to achieve a small task.

→ Syntax

access specifiers, modifier, return type, method name (~~()~~)

{

}

type Arg1, type Arg2 ---)

- ① → Access specifiers sets the visibility of the methods.
- Access specifiers is not mandatory.
- Access specifier → public, private, protected,

② modifier

→ It is indicates, whether a method is static or non-static.

③ → return type declaration is mandatory for a method. return type indicates the type of data return by a method.

If method does not return any value, then mention void as a return type

④ In the method body, return statement is mandatory for all the methods. return keyword is used to return a value from a method.

⑤ In a method the declared return type and return ~~type~~ value type should match

example

- A method declared as int return type, this method has to return integer value only
- for void method, we have to write empty return statement,

```
void method
{
    ;
    ;
}
return;
```

- if developer does not write return statement for void methods, java implicitly writes empty return statement at the time of compilation. So, it is not mandatory to write return statement for void methods.
- we can develop a method with arguments or without arguments
- arguments are like local variables of that method.

```
static int addInt( int n1, int n2)
{
    return n1+n2;
}
```

```
static double addIntDouble( int n1, double n2)
{
    return n1+n2;
}
```

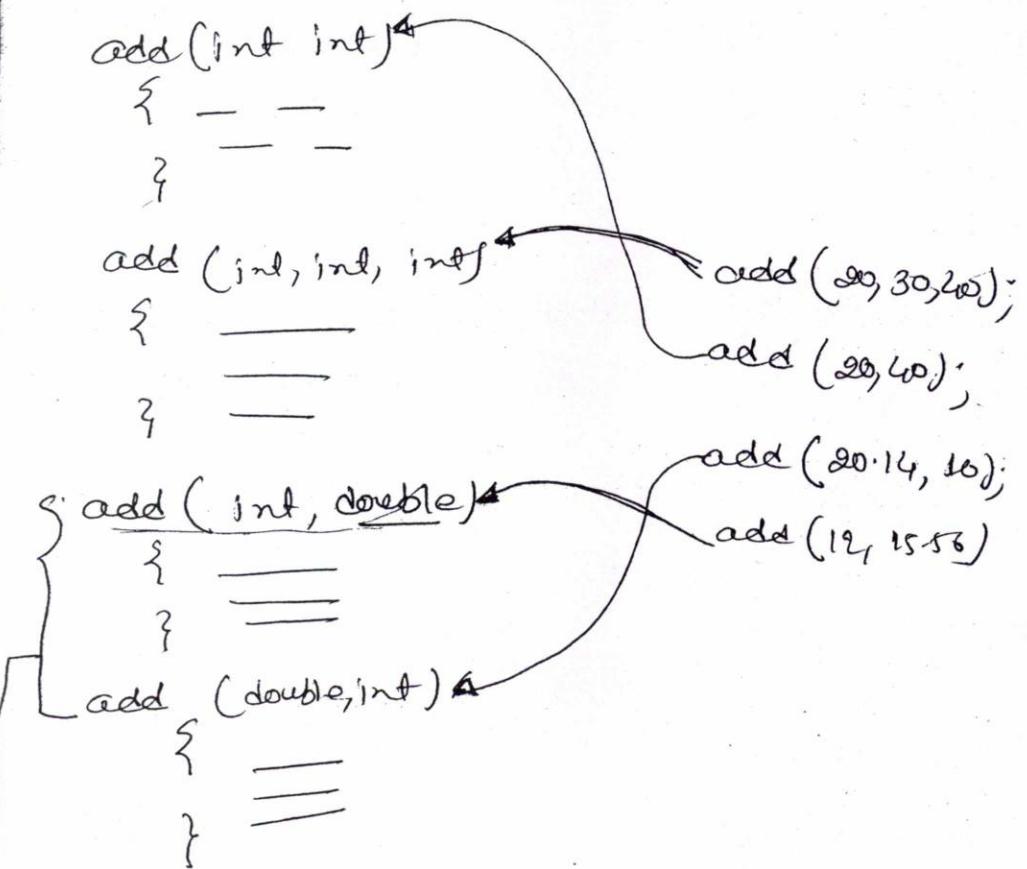
```
static double addDoubleInt( double n1, int n2)
{
    return n1+n2;
}
```

```
static double addDoubleDouble( double n1, double n2)
{
    return n1+n2;
}
```

→ ~~problem~~ problem of process oriented approach, that for addition two number, four methods are called with different method names with same example addition.

METHOD OVERLOADING

- Creating multiple methods with same name with different argument signatures is known as method overloading.
- Different in argument signatures either no. of ~~the~~ arguments should be different or type should be different.
- There is no importance of return type in overloading that means for two overloaded methods, either we can give same return type or different return type.
- When we invoke (call) overloaded method, java calls the particular method based on the arguments given.



→ Check here whether (`int, int`) and (`double, int`) is same signature or not.

rules of overloading

- Methods name to be same
- arguments signature should be different

example of overloading:

class cannot

{

 public static void main(String[] args)

{

 System.out.println("program starts");

 test1(10);

 test1();

 test1(20, 45);

 test1(30.45);

? System.out.println("program ends");

 static void test1()

{

? System.out.println("test1() with no args");

 static void test1(int a)

{

? S.O.P("test1() with one arg");

 static void test1(int a, int b)

{

? S.O.P("test1() with two args");

 static void test1(double a)

{

? S.O.P("test1() with double arg");

.

output

program starts.

test1() with one args

test1() with no arg

test1() with two arg

test1() with double arg

program ends

23

Assignment

- ① Create a class and overloaded add method
- ** \checkmark ✓ \checkmark
- class Demo7

? public static void main (String [] args)

```
{ int i=0;
  int j=0;
  i = test1 (i++) + test1 (++j);
```

System.out.println ("i+" + i);

System.out.println ("j+" + j);

```
} System.out.println ("Program ends");
```

Static int test1 (int a)

```
{ return a+1;
}
```

$i+3$
 $= 0$
 $j+4$
 $i=1$
 $\underline{0 \quad 2}$
 $\boxed{1+3=4}$
 $\underline{j+4, i+2}$

Output program starts

i=2;
j=2;
Program ends

$i=0 \& 2$
 $j=1+3=4$

→ If --
return (++a);

then

Output program starts

i=2;
j=4;
Program ends

Assignments

$$j = \text{test1}(i++) + \text{test1}(++i) + \text{test2}(--i) + \text{test2}(i--);$$

static int test1(int a)

```
{ return ++a;
}
```

static int test2(int a)

```
{ return --a;
}
```

j = 1, 2, 1, 0, 0

$$j = 1 + 3 + 0 + 0$$

$$j = 4 \quad \left\{ \begin{array}{l} \text{Output} \\ \text{Ans} \end{array} \right.$$

$$j = 0 \quad \left\{ \begin{array}{l} \text{Ans} \\ \text{Ans} \end{array} \right.$$

* class Demo8

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

```
{ System.out.println(" Pgm Starts");
    test1();
    S.O.P (" Pgm ends ");
}
```

static int test1()

```
System.out.println("inside test1()");
```

```
int a=2;
```

```
return a;
```

System.out.println("a=" + a) || unreachable statement

Output - error

unreachable statement over } busy
return statement is missing } error
over here

because

After return, there is a statement
& return statement will be last statement
in method

→ unreachable statement which are written after return statement are known as unreachable statements

- if a method has unreachable statements, java throws error during compilation
- return statement should be last line of the method

Assignment

class OverloadAdd

```
{ public static void main(String[] args)
{
    System.out.println("Program Starts");
    System.out.println("Addition1 = " + add(10, 20));
    System.out.println("Addition2 = " + add(10, 30.00));
    System.out.println("Addition3 = " + add(20, 330.33));
    System.out.println("Addition4 = " + add(3.30, 33.00));
    System.out.println("Addition5 = " + add(333.33, 333.33));
}
```

static int add(int a, int b)

```
{ return a+b;
}
```

static int add(int a, float b)

```
{ return a+b;
}
```

static int add(int a, double b)

```
{ return a+b;
}
```

static int add(float a, float b)

```
{ return a+b;
}
```

static int add(float a, double b)

```
{ return a+b;
}
```

Date
18/04/2013

Local & Global Variable :-

Class Run1

```
{ static int i=1 // Global variable  
public static void main (String [] args)
```

{

```
System.out.println ("main starts");
```

```
int b=2; // local to main
```

```
System.out.println ("i=" + i);
```

```
System.out.println ("b=" + b);
```

```
test1 (); // call test1()
```

```
System.out.println ("main ends");
```

}

```
static void test1 ()
```

{

```
System.out.println ("running test");
```

```
int c=3 // local to test1() method;
```

```
System.out.println ("i=" + i);
```

```
System.out.println ("c=" + c);
```

```
// System.out.println ("b=" + b); // error, can not find b here
```

```
System.out.println ("existing test1()");
```

{ }

Output

main starts

i=1

b=2

running test1()

i=1

c=3

existing test1()

main ends

class Run2

```
{  
    static int s; // global  
    public static void main (String [] args)  
    {  
        System.out.println ("main starts");  
        int j = 20; // local variable — need to be  
        System.out.println ("s=" + s); // initialized here  
        System.out.println ("j=" + j);  
        System.out.println ("main ends");  
    }  
}
```

output main starts

j=20.

s=20

main ends

(either static or
non static)

→ Global variable need not to be initialized.
because, compiler put default 0 to the
uninitialized global variable. but for
local variable, it is not true.

class Run3

```
{  
    static int a;  
    static double b;  
    static float c;  
    static char d;  
    static boolean e;  
    static String f;  
    public static void main (String [] args)  
    {  
        System.out.println ("main starts");  
        System.out.println ("int a=" + a);  
        System.out.println ("double b=" + b);  
        System.out.println ("float c=" + c);  
        System.out.println ("char d=" + d);  
        System.out.println ("boolean e=" + e);  
        System.out.println ("String f=" + f);  
    }  
}
```

System.out.println ("main starts");
System.out.println ("int a=" + a);
System.out.println ("double b=" + b);
System.out.println ("float c=" + c);
System.out.println ("char d=" + d);
System.out.println ("boolean e=" + e);
System.out.println ("String f=" + f);

Output

```

main starts
into
double b = 0.0
float c = 0.0
char d =
boolean e = false
String f = null
  
```

* all are given
 default value
 given by compiler
 and for char, it is
 empty means nothing is
 assigned by compiler

NOTE

- variable declared inside the methods are known as local variables.
- local variables can be accessed only within that method.
- local variables must be initialized before using them.
- method arguments are also local to that method.
- variables declared inside the class are known as global variables.
- Global variables can be accessed, anywhere in the class (i.e. in all the methods of class).
- Java assigns default values for uninitialized global variable. like int → 0, double → 0.0, boolean → false, char → , float → 0.0, String → null.

Static & non-static members

- * Variable (Global) and method declared inside the class are known as members of class or class members.

class Run3

{

```

variable1
variable2
=====
method1
method2
  
```

{

→ members of the class.

26

→ A class can contain both variable and method or only method or only variables (data members)
→ (behaviour) → (state of the object)

→ There are two types of members,

- (i) Static members
- (ii) Non static members

→ Static members are created using static keyword. Non-static members are created without using static keyword.

NOTE

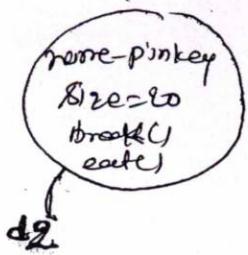
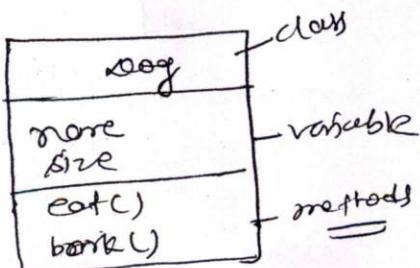
→ Since local variable are not members of the class so we can not make them static or non-static.

class & objects

→ Class is a blue-print or template for an object.

→ A class creates new data types in a program. (i.e. class is a logical construct)

→ An object is an instance of a class. An object is created by referring the class.

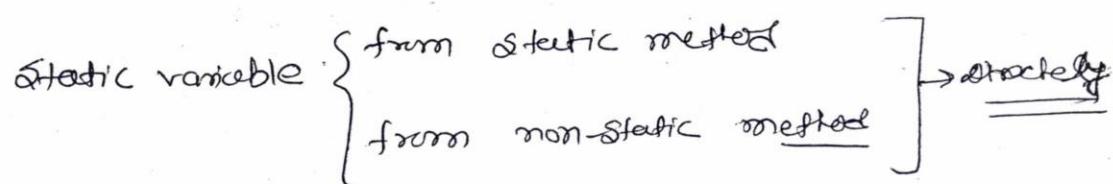


Note
19/04/2013

- Static or non-static are concept of only members.
- static or non-static are not for local variable.
- members of class is only → global variable and methods
- local variable is not member of class, we can not make it static or non-static.
- from static, we can not call non-static method and variable.

NOTE

- * within a same class, we can access static members directly from both static method and non static method.
- * non-static members can be accessed directly only from non-static method.



non static variable — only from non-static method

class Run4

{

static int i=10; // static variable

int j=20; // non-static variable

public static void main(String[] args)

{

System.out.println("i=" + i);

System.out.println("j=" + j); // compilation error.

}

* check, whether it can happen in different class or not

class Run 4

```
{  
    static int s=10; //static variable  
    int j=20; //non-static variable  
    public static void main(String args[])
```

```
{  
    System.out.println("main starts");
```

```
    System.out.println("s=" + s);
```

```
    System.out.println("j=" + j); //compilation error because j is non static  
    and main is static
```

```
} test(); //compilation error — test() is non static
```

void test()

```
{  
    System.out.println("s=" + s);  
    System.out.println("j=" + j); //compilation error
```

```
}
```

* In the static method, we can call only static variable and static method only

* In the non-static method, we can call ~~only~~ static variable, & non static method or variable

class Run 5

```
{  
    static int i=10; //global var  
    public static void main(String[] args)  
    {  
        System.out.println("program starts");  
        int int j=20;  
        System.out.println(i); //local var  
        System.out.println(Run 5.i); //global var  
    }  
}
```

output 10 20

Ques
In the
static method
we can call
static and non-
~~static~~
variable
only static means

- * we can access static members using class name
Syntax:
 $(\text{class name}).\text{member name};$
- * static member can be accessed to anywhere using
by its class name

* Non-static variable accessing

two types of variable

- primitive (int, long, double)
i.e predefined type
- reference
 - ↳ derived type

demo 9.1
Run 5 9.2

class Rmb

```
{ Static int i=20;
    int j= 20;
```

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

```
{ System.out.println ("Program starts");
```

```
S. O. P ( Rmb.i ); // 10
```

```
Rmb a1 = new Rmb();
```

```
S. O. P ( a1.j ); // 20 → accessing non-static  

variable through object
```

```
? S. O. P ( "Ends" );
```

Output

10	{
20	}

How to access non-static variables

→ Non-static variables can be accessed through reference variable

```
class name reference variable = new class name();  
reference variable • member name
```

Class A

```
{ int i=20;  
  static int j= 20;  
}
```

Class Run7

```
{  
  public static void main (String [] args)  
  {  
    System.out.println ("j=" + A.j);  
    A a1 = new A ();  
    System.out.println ("i=" + a1.i);  
  }  
}
```

Output $j=20$ $i=20$ } → A

V.VI *

Static member can be accessed before creating the object.

* main is static because, it is come first in memory first, if it is non-static then we have call this by creating object but main will start first, so it should be only static.

class B

{

 static int x;
 int y;

 public static void main(String[] args)

{

 System.out.println("program starts");

 System.out.println(B.x);

 B b1 = new B();

 System.out.println(b1.y);

 B.x = 100;

 b1.y = 200;

 System.out.println(B.x);

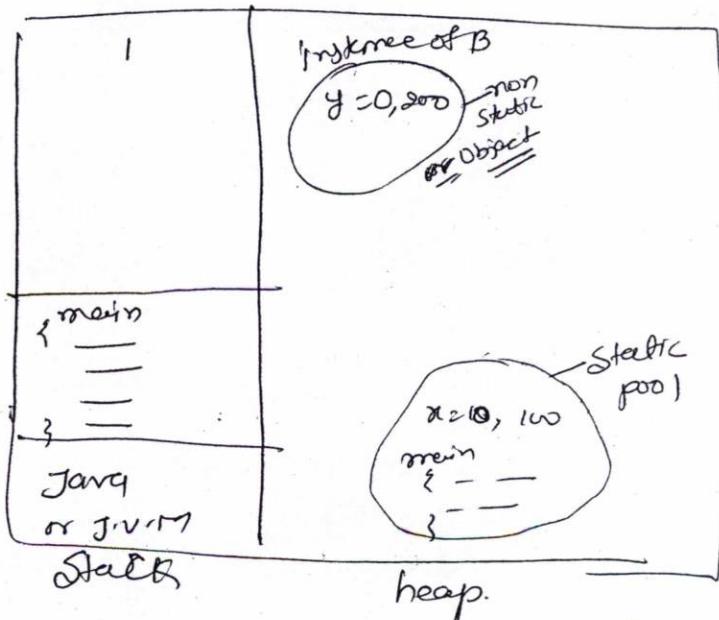
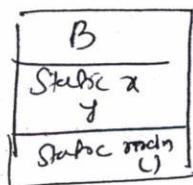
 System.out.println(b1.y);

 System.out.println("ends");

}

Output:-

0
0
100
200 } R



- i) memory allocation
- ii) Load all static members
- iii) invoke main
- iv) non-static members are loaded in heap.

→ When we execute a class, the following steps will be performed

- (i) Some piece of memory will be allocated for the program execution.
- (ii) That allocated memory will be divided into two parts. [Stack & heap]
- (iii) Stack is used for the execution purpose which implement LIFO concept.
- (iv) Heap is used for storage purpose.
- (v) Java or JVM is the first person which enter into stack & calls class loader.
- (vi) Class loader locates all static members of a class in a static pool.
- (vii) Java invoked main method.
- (viii) main() enters into stack, & ^{main} execution begins
- (ix) main() executes all the statements in sequential order.
- (x) When main() encounters object creation statement, [B b1 = new B();], then all non static members are loaded in heap and address of that instance will be stored in reference variable.
- (xi) When main() ~~over~~ execution will ends, then main(). come out the stack.
- (xii) Now java calls garbage collector, then it will come out of the stack.
- (xiii) Java clears the memory and release for use next.

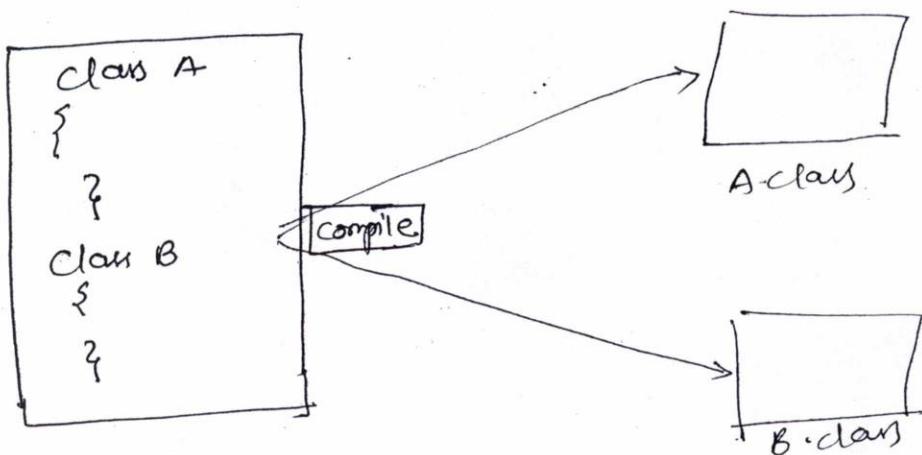
(xi) java collector clears the entire memory and
release.

Date



Date
22/04/2013

- * Compiler generates a class file of each class in a program.
- * Java allows only one public class in a file and more than one non public class.
- * Then try to give file name same as public class name.
Multiple classes in a file:-



- * we can write multiple classes in a single file if a file contain multiple non public classes, then we can give any class name as a file name.
- * Java allows only one public class in a file. If a file contains public class, then public given public class name as a file name.
So, after compilation, java creates separate class file for each classes present in source file.
- * we have to execute, the class which contains main().
- * In the entire java project, there is only one class which contains main(), and that class is used to start execution.

* Class Emp

```
{
    String name;
    int sal;
    String comName;
    void display();
}
```

void print()

```
S.O.P ("Name = " + name)
S.O.P ("Salary = " + sal);
S.O.P ("Company = " + emp.comName);
```

class Run8

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

```
System.out.println ("Pgm Starts");
```

```
→ Emp.comName = "Jspider";
```

```
Emp e1 = new Emp();
```

```
e1.name = "Prabash";
```

```
e1.sal = 20000;
```

```
→ e1.print();
```

```
S.O.P ("Name = " + name);
Emp.comName = "Jspider";
```

```
Emp e2 = new Emp();
```

```
e2.name = "Ranuhi";
```

```
e2.sal = 10000;
```

```
e2.print();
```

```
S.O.P ("Pgm ends");
```

Output

Pgm Starts

Name = Prabash

Salary = 20000

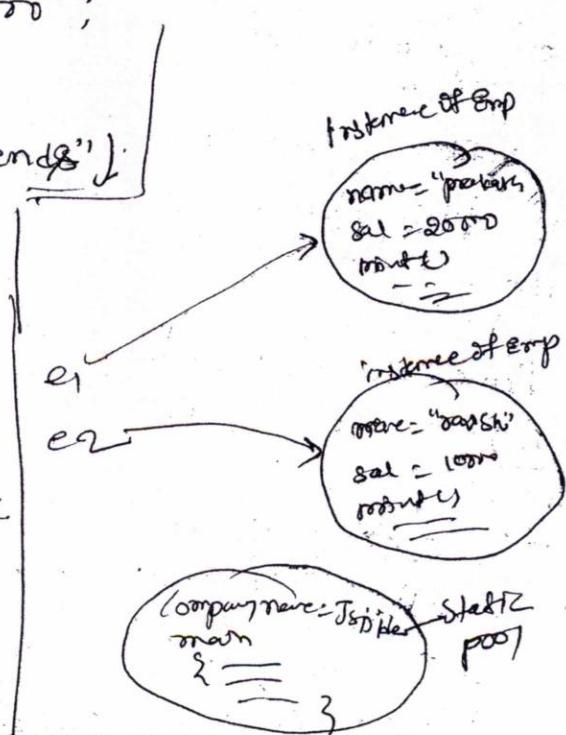
comp = Jspider

Name = Ranuhi

Salary = 10000

comp = Jspider

Pgm ends



~~What's~~ Difference b/w static & non-static members

static

- i) static members are declared using static keyword
- ii) static members are loaded into memory before creating instances of class.
- iii) static members can be accessed before creating an instance.
- iv) There is only one copy exists for all the instances.
- v) static members are accessed through class name.

Non-static

- i) get well declared without using static keyword
- ii) Non-static members are loaded into memory, each time instance created
- iii) non-static members are accessed after creating an instance
- iv) ~~there are~~ In each instance, copy of a non-static member exists
- v) non-static members are accessed through reference variable.

~~* Why main() method is static?~~

main() method should be invoked before creating an instance. (only static members will be accessed before creating an object).

Constructor

```
class Box
{
    int height;
    int width;
    int depth;
    void initialise();
}
Box b1 = new Box();
b1.height = 10;
b1.width = 20;
b1.depth = 30;
```

```
Box b2 = new Box();
b2.height = 20;
b2.width = 30;
b2.depth = 50;
```

call initialise
use this
to reduce lines

```
Class Box
{
    int height;
    int width;
    int depth;
}

void initialise(int h, int w, int d)
{
    height = h; width = w; depth = d;
}

Box b1 = new Box();
b1.initialise(10, 20, 30);
Box b2 = new Box();
```

* Constructor:-

Class A

```
{ int i;  
A() {  
}  
{ System.out.println("inside ct");  
}
```

constructor

(get invoked
automatically
when object
is created.)

Class Run1

```
{ public static void main(String[] args)  
{  
    S.o.p("pm starts");  
    A a1 = new A();  
    S.o.p(a1.i);  
    S.o.p("pm ends");  
}
```

Here automatically
constructor will
be invoked as
(A() is
invoked
automatically).

Output pm starts
inside ct
~~so~~
pm ends

Class A

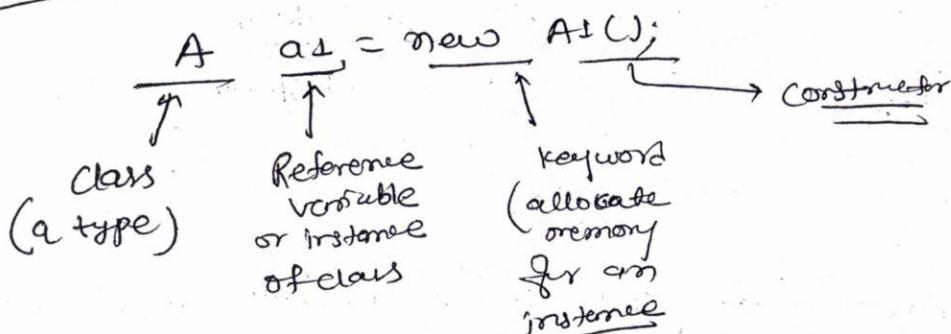
```
{ int i;  
A(int a)  
{ System.out.println("inside ct");  
    i = a;  
}
```

Class Run1

```
{ public static void main(String[] args)  
{  
    S.o.p("pm starts");  
    A a1 = new A(10);  
    S.o.p(a1.i);  
    S.o.p("pm ends");  
}
```

Output pm starts
inside ct
so
pm ends

- * constructor is always a non-static method
 - * never return, void for the constructor bc there is no return type for constructor
 - * But there can be access specifier (public, private) with constructor
 - * Constructor name should be same as class name
 - * we cannot call constructor separately, it is automatically called by java.
 - * Constructor is not member, because we can't invoke, it will automatically invoke by java when object is created
- Date 24/04/2012



- * Constructor is the special form of method, which is invoke while creating instance of a class using new operator keyword
- * Constructor has statements that executes automatically, when we create instance of a class.
- * Constructor name should always match with class name.
- * we can create constructor with arguments or without arguments.
- * Constructor without an argument are known as default constructor or no args constructor.
- * Constructor with argument is known as parameterized constructor.

* Constructor can not return any value, so Return statement is not allowed inside the constructor, and also return type declaration is not allowed (Even void is not allowed).

* Constructor should be always non-static.

* we can use access specifier for a constructor.

* Every class in java should contain a constructor (Constructor is mandatory for all the classes, without constructor we can not create an instance).

* Java writes default constructor at the time of compilation if developer has not written the constructor.

* Default constructor is not no argument, ~~no~~ Statement constructor.

class Run3
{
 int i; }
} class Run3
{
 public static void main(String[] args)
 {
 System.out.println("Program starts");
 B b1 = new B();
 b1.i = 10;
 System.out.println("b1.i=" + b1);
 System.out.println("end");
 }
}

Java will define or create a default constructor B() because there is not constructor written by programmer. So, default will be created during compile time when object is created.

Use of constructor:-

- * Constructor is used to initialize the object or instance while creating an instance

class Tree

```
{
    int height;
    //constructor
    Tree(int h)
    {
        height = h;
    }
}
```

S.O.P ("Tree object is constructed with " + height);
+ "feet height"

```
Tree()
{
    Tree()
    Tree()
    Tree()
```

class Run4

```
{
    public static void main (String [] args)
    {
        S.O.P ("Pgm starts");
        Tree t1 = new Tree(6);
        S.O.P ("height of t1" + t1.height);
        S.O.P ("-----");
        Tree t2 = new Tree(11);
        S.O.P ("height of t2" + t2.height);
        S.O.P ("ends");
    }
}
```

Here as object is created before executed
next line first constructor will be executed
first,
so see
O/P

O/P

Pgm starts

Tree object is constructed 6 feet height
height of t1 = 6

Here
constructor
executed
first

Tree object is constructed 11 feet height
height of t2 = 11
Pgm ends

34

- * non static variables are also called as instance variable
- * constructor is used to initialize instance variable while creating

* Non - static variable is called instance variable.

* Constructor is used to initialize all instance variable while create the object of class:

~~We can not call multiple constructor inside constructor in the same class, because it will enter into infinite loop~~

{
 int size

 dog(int size)

 {
 this.size = size;
 S.O.P ("dog is constructed");
 }

"this" is used here
to differentiate
from local &
~~global~~
variable & same
name).

class Run5

{
 public static void main(~~String~~ args)
 {
 S.O.P ("starts");
 ~~dog~~
 dog d1 = new dog();
 S.O.P ("size = " + d1.size);
 S.O.P ("ends");
 }
}

~~old~~
program starts

dog is constructed
size - 8
program ends

If in constructor
written as

size = size

we ~~old~~ is

size = 20;

this → means current instance :-

This keyword is used to access current instance's non-static member.

Syntax

this. non static member name

* this. size = size
↑
current instance ↑
non static variable local
variable

Constructor Overloading :

* Creating multiple constructor with different argument signature is called constructor Overloading

Class C

{

C()

{
S.o.P(" no arg ctt");
}

C(int h)

{
S.o.P(" one int arg ctt");
}

C(int a, int b)

{
S.o.P(" 2 int arg ctt");
}

{

Class Run:

{ public static void main (String [] args)

{ S.o.P(" pm starts")

} constructor with
same name
with different
argument,

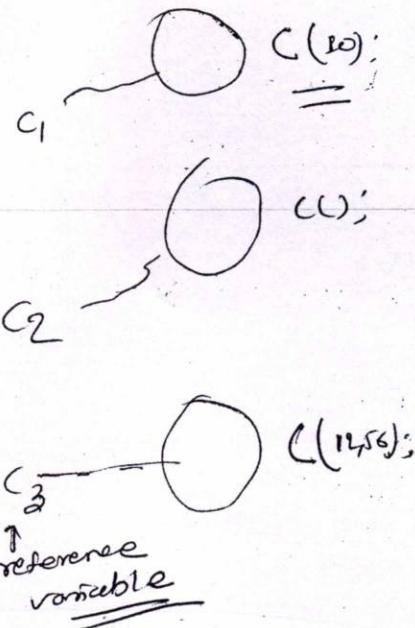
```

C C1 = new C(10);
C C2 = new C();
C C3 = new C(12, 86);
S-O-P("ends");
}
}
}

```

Op

from scratch
 one int arg Ctt
 no arg Ctt
 2 int arg Ctt
ends.



* class Cat

```
{
    String name;
    int size;
}
```

Cat()

```
{
    S-O-P ("Cat is constructed w/o name");
}
```

Cat(String name, int size)

```
{
    this.name = name;
    this.size = size;
}
```

Cat(String name)

```
{
    this.name = name;
    S-O-P ("Cat is constructed only with name");
}
```

Cat(int size)

```
{
    this.size = size;
    S-O-P ("Cat is constructed only with size");
}
```

class Run {

{ public static void main (String args) }

{

s.o.p ("start");

Cat c1 = new Cat();

Cat c2 = new Cat("Gatty", 5);

Cat c3 = new Cat("choppy");

Cat c4 = new Cat(6);

s.o.p ("ends");

}

Output

Start

Cat is constructed w/o name & size

Cat is constructed with name & size

Cat is constructed with only name

Cat is constructed only with size

Ends

Assign

① Create an employee class with name and salary as instance variable and create overloaded constructor to initialize instance variable.

② Create a tiger class with size as instance variable, create a constructor to initialize a size.

③ Create instances of a tiger, ~~no. of~~ point the no. of instances created.

class Pen

```
{  
    int height;  
    string inkcolor;
```

-Pen()

```
{  
}
```

-Pen(int height, string inkcolor)

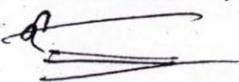
```
{  
}
```

```
    this.height = height;
```

```
    this.color = inkcolor;
```

```
{  
}
```

pen (int height)



P

pen (int height, string inkcolor)

{
 this.height = height;
 this.color = inkcolor;

Pen(Pen P) -

```
{  
}
```

```
    this.height = P.height;
```

```
    this.color = P.color;
```

```
{  
}
```

void print()

```
{  
}
```

```
    S.O.P("Height = " + height);
```

```
    S.O.P("Inkcolor = " + inkcolor);
```

```
{  
}
```

```
{  
}
```

class Run8

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

```
{  
}
```

```
    S.O.P("starts");
```

```
    Pen P1 = new Pen(10, "Black");
```

```
    P1.print();
```

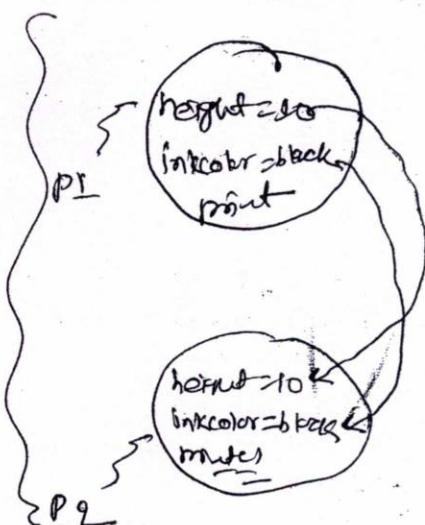
```
    S.O.P(" - - - ");
```

```
    Pen P2 = new Pen(P1);
```

```
    P2.print();
```

```
    S.O.P("ends");
```

```
?  
P3
```



Assignment

① class Employee

```
{
    String name;
    int salary;
    employee(String name, int salary)
}
```

```
{
    this.name = name;
    this.salary = salary;
}
```

Employee(Employee e)

```
{
    this.name = e.name;
    this.salary = e.salary;
}
```

void print()

```
{
    S.O.P("Salary=" + salary);
    S.O.P("name=" + name);
}
```

class Run

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

```
{
    S.O.P("Starts");
    Employee e1 = new Employee("ABC", 20000);
```

```
e1.print();
```

```
S.O.P("-----");
```

```
Employee e2 = new Employee(e1);
```

```
e2.print();
```

```
S.O.P("Ends");
```

O/P = Starts

salary = 20000

name = ABC

salary = 20000

name = ABC

Ends.

87

```

class Cat
{
    int size;

    Cat()
    {
        S.O.P("Constructor called without size");
    }

    Cat(int size)
    {
        this.size = size;
    }

    void print()
    {
        S.O.P("size = " + size);
    }

    Cat(Cat c)
    {
        this.size = c.size;
    }
}

class Run
{
    public static void main(String[] args)
    {
        S.O.P("Starts");

        Cat c1 = new Cat();
        c1.print();
        S.O.P(" --- ");

        Cat c2 = new Cat(10);
        c2.print();
        S.O.P(" --- ");

        Cat c3 = new Cat(c2);
        c3.print();
        S.O.P(" ends");
    }
}

```

O/P
Starts
size = 10;
size = 10;
ends.

```

class tiger
{
    int size; - } check whether
    String color; - non static
    String name; - but have default
    int count = 0; - value of not
    tiger (int size, String color, String name)
    {
        this.size = size;
        this.color = color;
        this.name = name;
        this.count = count + 1;
    }
}

```

tiger (tiger t)

```

{
    this.size = t.size;
    this.color = t.color;
    this.name = t.name;
    this.count = count + 1;
}

```

class Run

```

public static void main (String [] args)
{
    S.O.P ("starts");
}

```

tiger t1 = new tiger (10, "red", "TIG");

t1.point();

S.O.P ("-----");

tiger t2 = new tiger (t1);

t2.point();

S.O.P ("-----");

tiger t3 = new tiger (t2);

t3.point();

S.O.P ("ends");

by SR

class tiger

```

{ int size;
    static int count;
}

```

Tiger (int size)

```

{ this.size = size;
    Tiger.count++;
}

```

if
we don't static, then it is
non-static, and each time
it will loads into memory
and first assign default value
and then increment by each
time it will be 1 only

void point()

```

{ S.O.P ("size = " + size);
    S.O.P ("color = " + color);
    S.O.P ("name = " + name);
    S.O.P ("no of instance created is " + count);
}

```

38

non-static

member loaded
into memory each
time when

object is created
and they default
or initialized value
is stored in the
memory, so there
won't make only one
copy then make it

olp

25/10/2023

Stents

size = 10;

color = red;

name = TIG.

no. of instance created is 1

size = 10;

color = red;

name = TIG.

no. of instances created is 2

size = 10;

color = red;

name = TIG.

no. of instances created is 3

ends

program by sir, Some program :

class Tiger

{ int size;

~~static int count;~~

making it static

because to keep only
one copy, & always
when any constructor
is called ✓

 Tiger (int size)

 { this.size = size;

 } Tiger.count++;

}

class Rang

{ public static void main (String [] args)

 { S.O.P ("Start");

 Tiger t1 = new Tiger(10);

 Tiger t2 = new Tiger(8);

 Tiger t3 = new Tiger(12);

 S.O.P ("no of Tiger=" + tiger.count);

 S.O.P ("Ends");

}

olp Stents
no. of Tiger = 3
ends.

Date
25/07/2013

* using "this" keyword, we can call constructor in constructor in the same class.

Class Box

```
{  
    int height;  
    int width;  
    int depth;  
    String color;
```

Box(int h, int w, int d)

```
{  
    height = h;  
    width = w;  
    depth = d;  
    S.O.P("2nd ct");
```

Box(int h, int w, int d, String c)

```
{  
    no need  
    to write  
    again  
    [ ] {  
        height = h  
        width = w  
        depth = d  
        +HIS(h, w, d); } } → read as call to HIS  
    String color = c;  
    S.O.P("3rd ct");
```

void print()

```
{  
    S.O.P("height = " + height);  
    S.O.P("width = " + width);  
    S.O.P("depth = " + depth);  
    S.O.P("color = " + color);
```

instance of Box

height = 10
width = 8
depth = 6
color = blue

b!

Here, code is repeating,
so, instead of
repeat, we can call upper
constructor with "this".
gt will call
current constructor;

class Run10

```
{  
    public static void main (String [] args)  
    {  
        S.O.P("Starts");
```

39

Box b1 = new Box(10, 6, 8, "blue").

b1.print();

s.o.p(" ends");

}

O/P

program starts

2nd ct

3rd ct

height = 10

width = 8

depth = 6

Color = Blue

program ends

class A

{

A()

{ s.o.p("1st ct"); }

A(int a)

{

this();

s.o.p("2nd ct"); }

class B extends A

{ public static void main(String[] args)

{

s.o.p("Start");

A d1 = new A(10);

? s.o.p(" ends");

}

if we write
[s.o.p("2nd ct")
this();]

it will throw error, that
this() should be always
first statement

we can not write
this() in the 2nd line
this() should always
first statement, & we can
call only one constructor
within constructor
one time

O/P = Start
1st ct
2nd ct
program ends

POINTS

- * "this" is also used to call the constructor of the class.

this() [call to this].

- * "Call to this" should be the first statement in the constructor.

i.e super()
{
 S.O.P(" "),
 this();} } → throw
error

- * Use of "this" keyword:—

(1) "this" keyword is used to access the non-static member of the current instance.

(2) Syntax = this.member name.

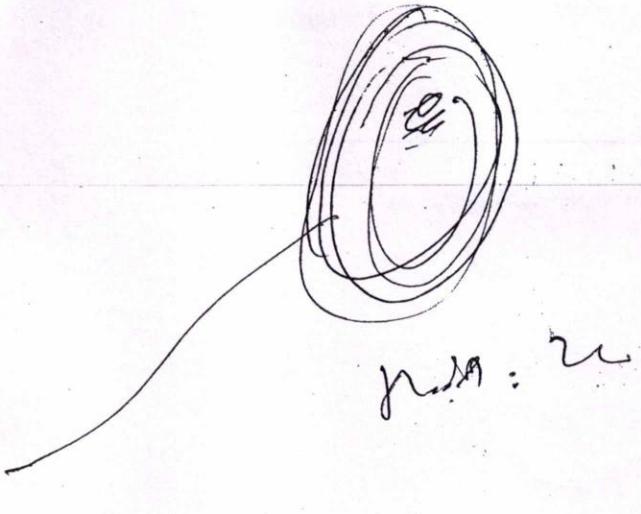
(2) "this" is also used to call the constructor of the same class.

this() [call to this].

Assignment

40

- (1) Create a student class with name, id, marks, phone no, Gender, as fields.
Create a constructor, to construct an object by taking name, id, and marks and create one more constructor to create an instance by taking all the values and create one more constructor, to create an instance with default values.



for i = 1 to n
 for j = 1 to m

Class Student

{

 Student()

System.out.println()

- * we can call only method inside the main().
- * we can not ~~can not~~ define a method inside a main() method

Assignment question

Class Student

{

```

    string name;
    string id;
    int marks;
    int phoneNo;
    string gender;
  
```

Student(string name, string id, int marks)

{

this.name = name;

this.id = id;

this.marks = marks;

System.out.println("constructor with three arguments");

}

Student(string name, string id, int marks, int phoneNo,
string gender)

{

System.out.println("System with 2 argument while calling
first in the constructor");

this(name, id, marks);

→ write here ↑

this.phoneNo = phoneNo;

this statement
will be always

this.gender = gender;

first statement

it cannot first state
→ next

System.out.println("constructor calling another constructor
with five arguments");

{

Student()

{

System.out.println("Student constructor with
default argument");

void print()

{

System.out.println("name = " + name);

S.O.P("id = " + id);

S.O.P("marks = " + marks);

S.O.P("phone no = " + phoneNo);

S.O.P("gender = " + gender);

q1

? ?

class Run

25/04/12

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

```
{ System.out.println("program starts");
```

```
Student s1 = new Student();
```

```
s1.print();
```

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

~~System.out.println()~~

```
Student s2 = new Student("varish", "4B008EC083", 70);
```

```
s2.print();
```

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

```
Student s3 = new Student("gaurav", "4B008ISO27", 80,  
9742, "male");
```

```
s3.print();
```

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

}

O/P

pgm starts

name=null

id=null

marks=0

phone no=0

gender=null

constructor with 3 arguments

name=varish

id= 4B008EC083

marks=70

phone=0

gender=null

constructor with 3 arguments

this constructor with 5 args
will call first another
constructor with 3 argument

name=gaurav

id= 4B008ISO27

marks=80

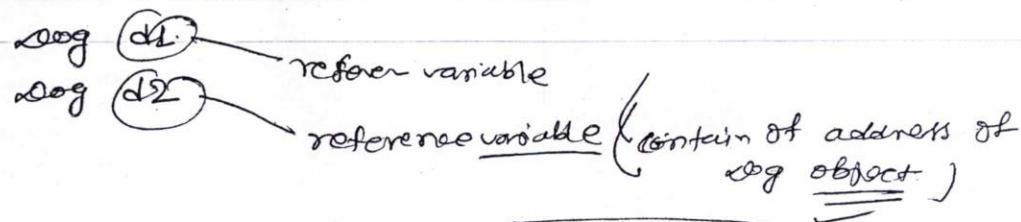
phone=9742

gender=male

pgm ends

Date
25/04/2013

reference variable contain address of the object



```
class Dog
{
    String name
    Dog(String n)
    {
        name=n;
    }
}
```

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

`Dog d1;` → declare a variable of type `Dog`

`d1 = new Dog("A");` → create an instance and assign reference into `d1`.

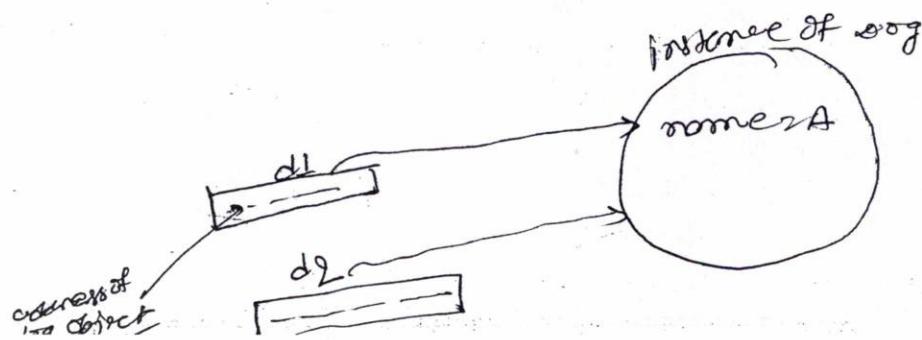
`Dog d2;`

`d2 = d1;`

→ copy `d1` contents to `d2`

i.e. `d1` contain address of `Dog` object,

so, get copy address of `Dog` object to `d2`



42

```

class dog
{
    string name;
    dog(string n)
    {
        name = n;
    }
}

```

```
class Run
```

```
? p.s. vvv (string langs)
```

```
{ dog d1;
```

```
d1 = new dog ("A");
```

```
dog d2;
```

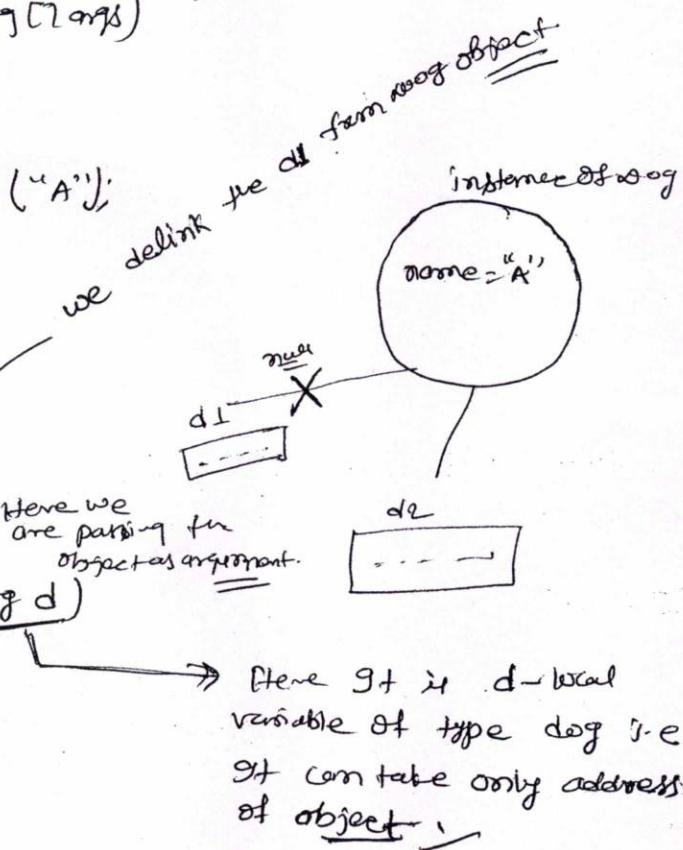
```
d2 = d1;
```

```
d1 = null
```

```
? test1(d2);
```

```
static test1(dog d)
```

```
{ d.name;
```



** If we want to delink any reference variable, then assign it null like

$d_1 = \underline{\text{null}}$

** $\underline{\text{test}}(d_2)$ → we are passing the address of object of dog that are present in d_2 . reference variable

** static test1(dog d) → argument of type dog
 that can take only reference variable that contain address of instance of dog

```

* class dog
{
    String name;
    dog (String n)
    {
        name=n;
    }
}

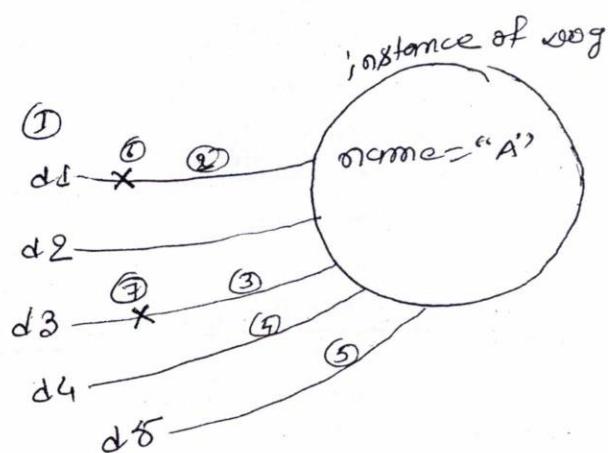
```

class Run

```

{
    p.s. m()
    {
        dog d1, d2, d3, d4, d5;
        d1 = new dog ("A");
        d2=d1;
        d3=d2;
        d4=d2;
        d5=d1;
        d1=null;
        d3=null;
    }
}

```

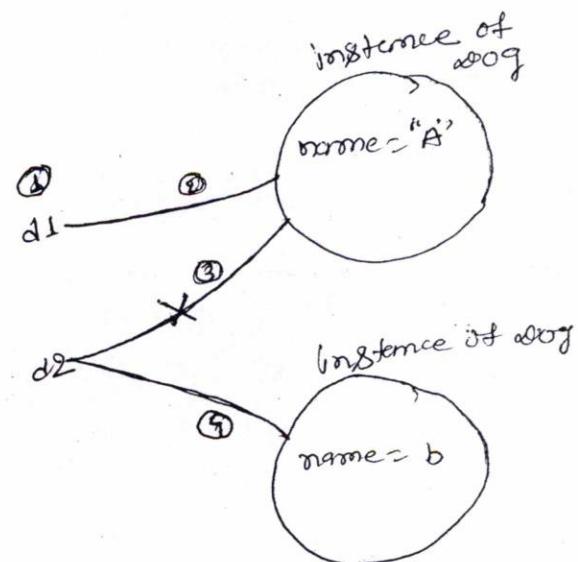


class Run

```

{
    p.s. m (---)
    {
        dog d1, d2;
        d1 = new dog ("A");
        d2=d1;
        d2 = new dog ("b");
    }
}

```

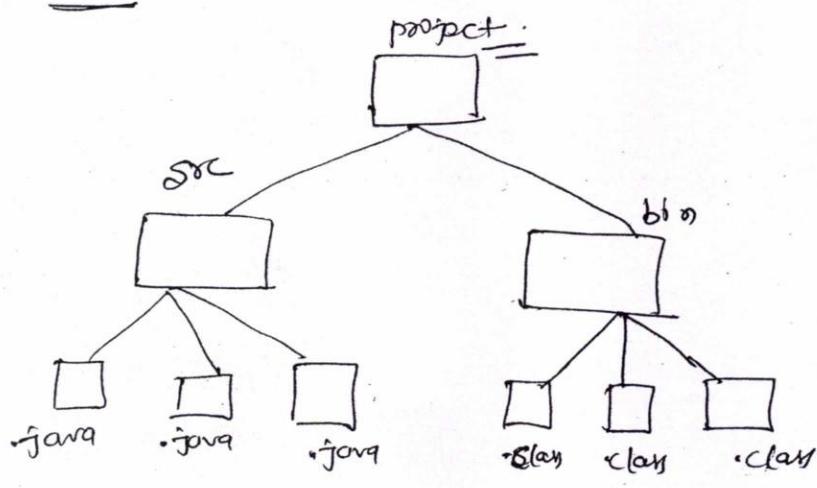


- * An instance can have n-number of reference variable.
- * To Remove the reference, assign null to reference variable. or assign other object reference.

*** If an object does not contain any reference, it is known as abandon object. So, Garbage collector cleans the abandon object.

Organizing source file and class file:-

- It is standard practice to store class files and source in a separate directory.
- Generally we store all source file in a source folder and store all class file in bin folder.



- By default, `javac` command, stores generated class file in the same directory which has source file.
- to save class file in separate directory at the time of compilation, use '-d' option.

`cd /home/blk/src > javac -d /home/blk/bin A.java`

Absolute
path

d:\ocm13\blocks\src > javac -d d:\ocm13\block\bin A.java

> java ..\bin A.java.

relative path

go to parent folder

Unary operator (v.v2 for interview)

Blocks (v.v1 of interview)

→ group of statements present in {} ?

→ two types of Block

→ static block

→ non-static blocks

static

{
—;
—;
—;
}

{

—;
—;
—;

}

static
block

non-static

block

→ Static block executes before main.

→ Static block executes first time when we use the class.

→ If multiple static blocks are exist, all static blocks are executed before main, in the order which we created.

→ Static block is also called as static initialization block. (S.I.B.).

Class A

```
{  
    public static void main(String [] args)  
    {  
        System.out.println("main starts");  
        System.out.println("main ends");  
    }  
}
```

Static

```
{  
    System.out.println("static block 1");  
}  
}
```

Static

```
{  
    System.out.println("static block 2");  
}  
}
```

Static

```
{  
    System.out.println("static block 3");  
}  
}
```

first
static
block will
be
executed

O/P:-

Static block 1
Static block 2
Static block 3
main starts
main ends

first
static block
executed

to create
javac -d ..\bin B.java
cd ..\bin
java A.java
cd ..\src
javac -d ..\bin B.java

* Non-Static block

- Non-Static block will be executed each time when we create an instance.
- If multiple block of non-static are exists, then all non-static blocks are executed, each time when we create instance in the order in which blocks are created.
- Non-Static blocks are also called instance initialization block (I.I.B)

Class B

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

```
{ System.out.println ("main starts");
```

```
B b1 = new B();
```

```
B b2 = new B();
```

```
S.O.P (" ends");
```

```
}
```

```
{
```

```
{
```

```
S.O.P (" non static block 1");
```

```
{
```

```
{
```

```
S.O.P (" non - static block 2");
```

```
}
```

O/P main starts-

non-static block 1

non - static block 2

non - static block 1

non - static block 2

main ends-

Ques

Class C

```
{ C()
```

```
{ S.O.P (" C constructor");
```

```
{ S.O.P (" non - static block");
```

Class Run

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

```
{ S.O.P (" starts");
```

```
C c1 = new C();
```

```
{ S.O.P (" ends");
```

O/P Starts
non - static block
C constructor will
be execute

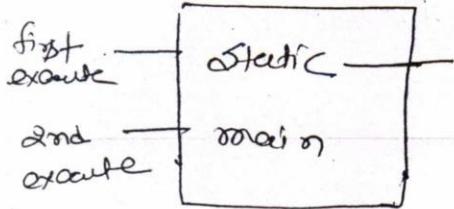
45

O/P Starts
non - static block
C constructor ends

Correct
order

non - static block
C constructor
ends

order (Remember)



Date
26/04/15

* class Run2

{

Run2()

{

System.out.println("nonRun2 cttr");

}

Static

{

System.out.println("Static block");

}

{

System.out.println("non-static block");

}

public static void main(String[] args)

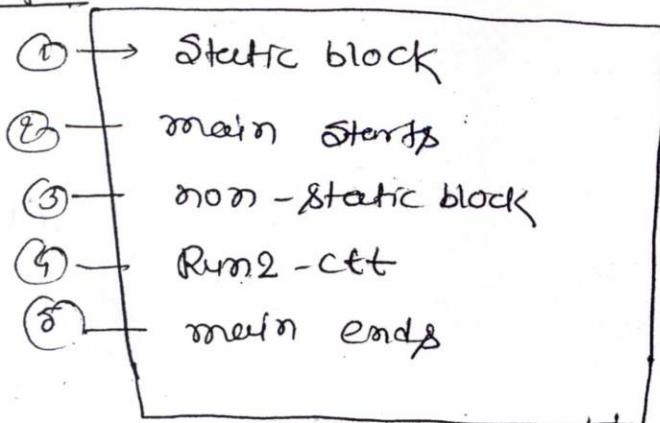
System.out.println("main starts");

Run2 r = new Run2();

System.out.println("main ends");

}

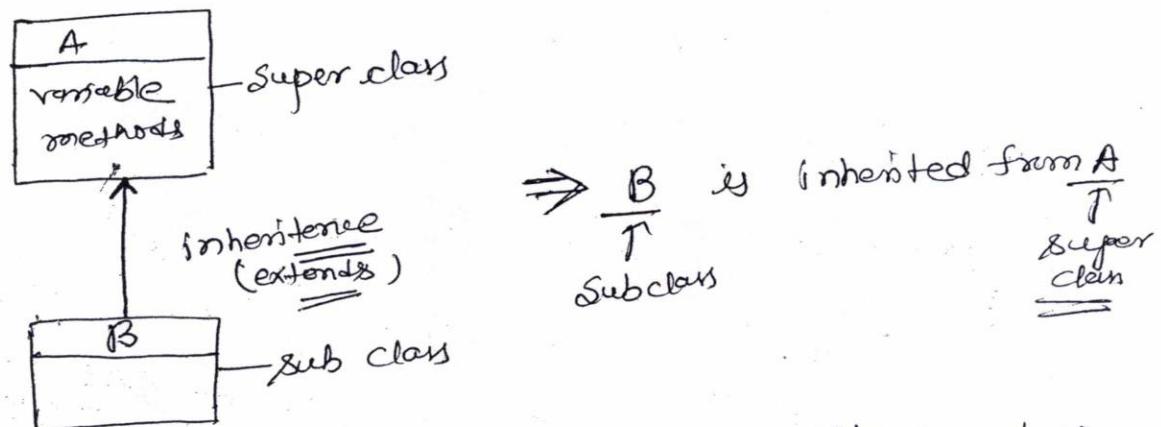
Output



Date
26/04/2013

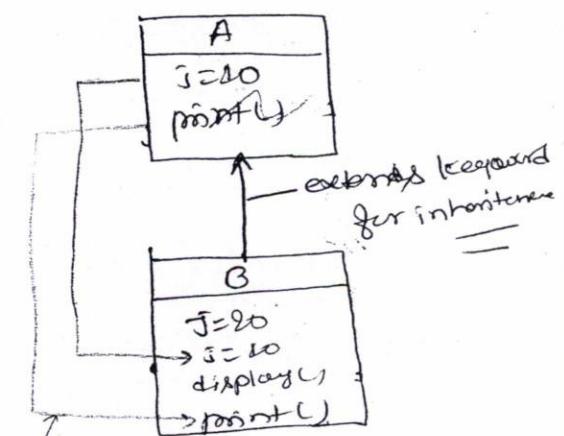
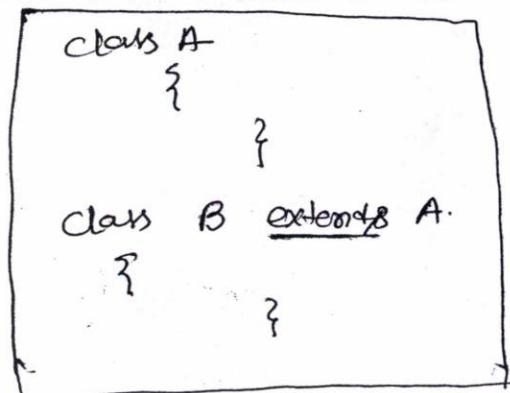
Inheritance:-

- * Inheritance is a process of acquiring members of one class to other class.
- * ~~for~~ The class from which the members are acquiring is known as super class.
- * The class to which members are acquired is known as sub-class.



generally, sub-class acquires all the members from super class or sub-class is inherited from super class.

for example, syntax for inheritance



due to inheritance,
this member are also a
member of sub-class B
but vice versa is not true.

* Class A

```
{  
    int i=10;  
    void print() { S.O.P(i); }  
}
```

Class B extends A

```
{  
    int j=20;  
    void display() {  
        S.O.P(j);  
    }  
}
```

Class Run

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

```
        S.O.P(" Pgm Starts");
```

```
        B b1 = new B();
```

```
        b1.display();
```

```
        b1.print(); // inherited method of class A, but
```

```
        S.O.P(" Pgm ends");
```

due to inherited B, we
cancel here print method
through class B. as print
method is a member
method of class A, but
due to inheritance
print method is also
the member of class B

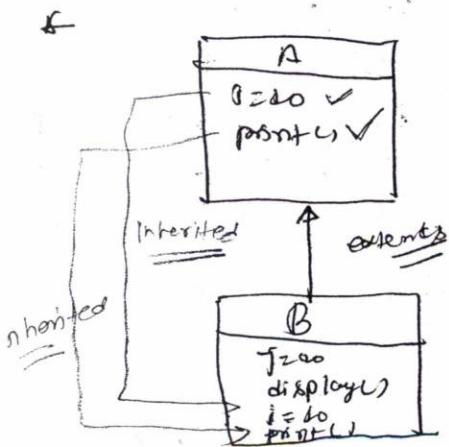
OP

Pgm Starts

20

10

Pgm ends



A a1 = new A();

a1
a1.print
only

instance of A

i=10
print()

B b1 = new B();

b1
b1.print
b1.display

instance of B
inherited
i=10
print()
j=20
display()

class C

```
{ int i=10;  
}
```

class D extends C

```
{  
    int j=20;  
}
```

class Run2

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

```
{ System.out.println ("prog starts");
```

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

```
C c1 = new C();
```

```
System.out.println (c1.i);
```

```
D d1 = new D();
```

```
System.out.println (d1.i);
```

```
System.out.println (d1.j);
```

```
System.out.println ("prog ends");
```

* check out
gt g write only
System.out(i) - then what

O/P prog starts

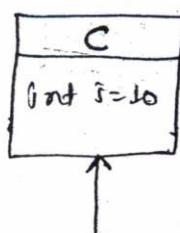
10

10

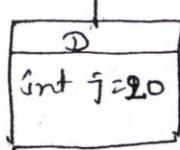
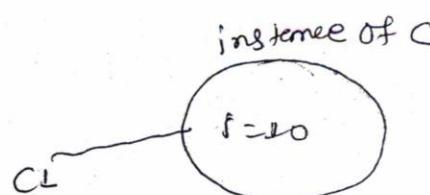
20

prog ends

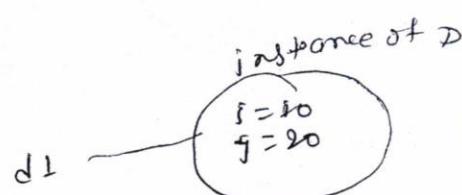
↑ over prog diagrams.



```
C c1=new C();  
System.out.println(c1.i);
```



```
D d1=new D();  
System.out.println(d1.i);  
System.out.println(d1.j);
```



47

```

+ class E
{
    void test1() {
        S.O.P("test1() of E");
    }
}

```

```

class F extends E
{
    void test2() {
        S.O.P("test2() of F");
    }
}

```

```

class G extends F
{
    void test3() {
        S.O.P("test3() of G");
    }
}

```

```

class Run3
{
    public static void main(String[] args) {
        S.O.P("Program starts");
    }
}

```

```

E e1 = new E();
e1.test1();
S.O.P("-----");

```

```

F f1 = new F();
f1.test1(); // Inherited
f1.test2();
S.O.P("-----");

```

```

G g1 = new G();
g1.test1(); // Inherited
g1.test2(); // Inherited
g1.test3();
S.O.P("Program ends");

```

O/P

Program starts

test1() of E

test1() of E

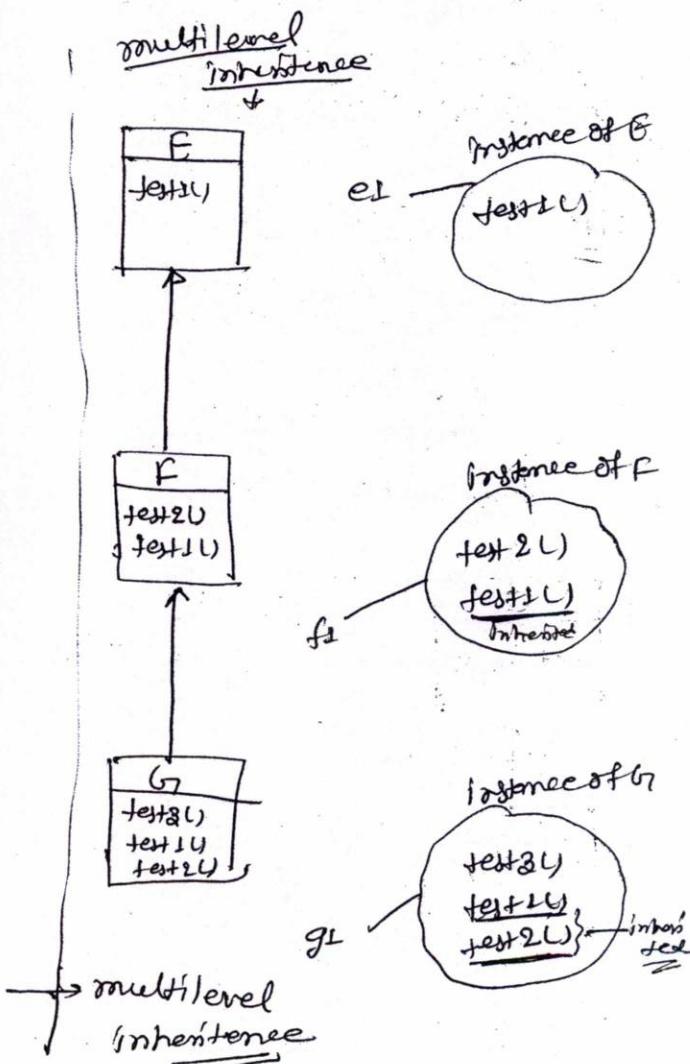
test2() of F

test1() of E

test2() of F

test3() of G

Program ends



Class H

```
{
    void test1()
    {
        System.out.println("test1 of H");
    }
}
```

Class I extends H

```
{
    void test2()
    {
        System.out.println("test2 of I");
    }
}
```

Class J extends I

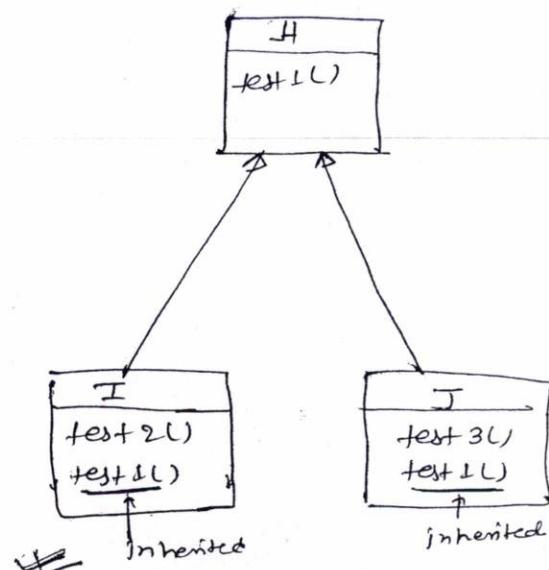
```
{
    void test3()
    {
        System.out.println("test3 of J");
    }
}
```

Class Run4

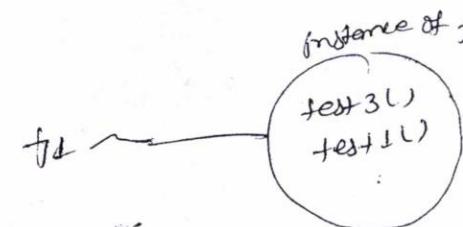
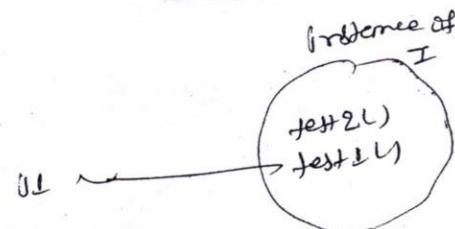
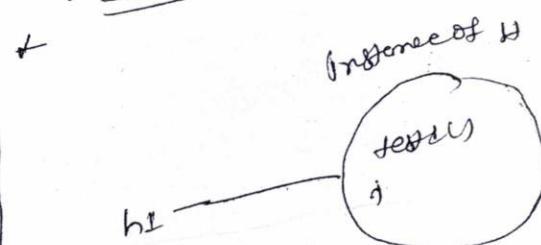
```
{
    public static void main(String args)
    {
        System.out.println("Starts");
        H h1 = new H();
        h1.test1();
        System.out.println("-----");
        I i1 = new I();
        i1.test1();
        i1.test2();
        System.out.println("-----");
        J j1 = new J();
        j1.test1();
        j1.test3();
        System.out.println("Ends");
    }
}
```

O/P

- Starts
- test1() of H
- test2() of I
- test2() of I
- test1() of J
- test3() of J
- Ends



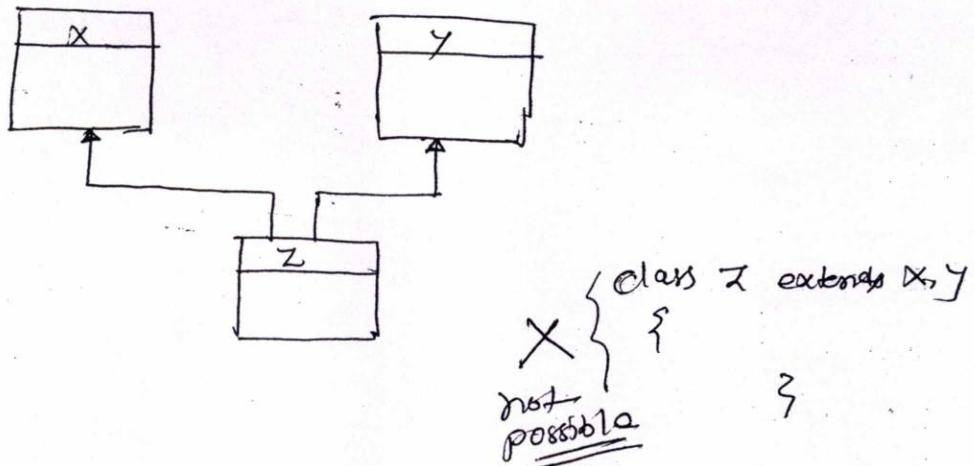
+ one super class and one or more sub-class is possible



+ but one sub-class have more than one parent so super class, it is not possible

~~V.V.1~~ Using classes multiple inheritance is not possible in Java.

A class can not extend two classes. ~~and~~
a class can not be inherited from two classes



~~KKK~~ ~~WTF~~ A super class can have any number of sub-classes, but a sub-class should have only one super class.

Constructor chain :-

class K

```
{ K() {  
    } } System.out.println("K ct") }
```

class L extends K

```
{ L() {  
    // from (your compiler)  
    // when calling superclass constructor(class K)  
    // System.out.println("L ct")  
    } }
```

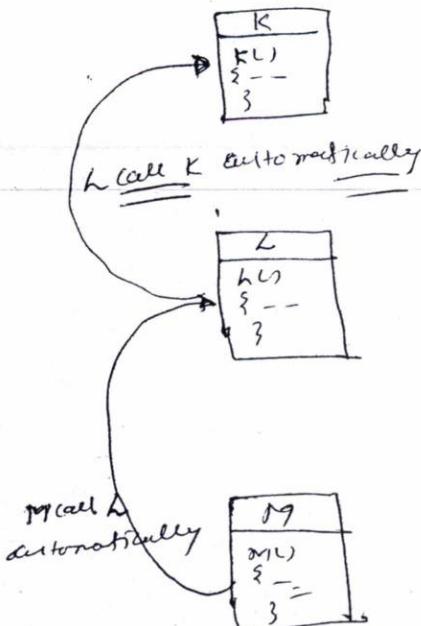
class M extends L

```
{ M() {  
    // from calling superclass constructor here  
    System.out.println("M ct")  
    } }
```

whenever a sub-class object will be created, its constructor will call that class's constructor before that its super class or parent class constructor will be called automatically.

class Run5

```
{  
    public static void main  
        (String [7 args])  
    {  
        S.O.P ("Program Starts");  
        M obj = new M();  
        S.O.P ("Program ends");  
    }  
}
```



Output

Program Starts

K ctt.

L ctt

M ctt

Program ends

~~class~~

* Static members can
not be inherited.
* Only non-static members can
be inherited.

- * In an inheritance tree, when we create an instance of sub-class, sub-class constructor called super-class constructor, and super-class constructor called父类.
Super-class constructor one go on ----, this is called constructor chain.

- * Because of constructor chain, inheritance is happening.

(iii) ~~no~~ If there is no - constructor chain, then there is no - inheritance

- * There is a keyword called which is used to call super-class constructor.

Super() → call to Super.

- * Whenever a ~~sub~~ instance of sub-class created, it will call Super() or add Super() to call the constructor of super class.

class K

{

K()

{

S.O.P("K Ctt");

}

class L extends K

{

{

Super();

{

S.O.P("L Ctt");

}

class M extends L

{

M()

{

Super();

{

S.O.P("M Ctt");

}

If a superclass have more than one constructor then through super in the sub-class, we have call any one constructor of super-class must in sub-class.

It is default constructor.

and know to call default superclass constructor if there is no other constructor called by param. So, whether write or not, it is called by param.

It is default constructor.

It is automatically called if param not called any superclass constructor.

If super class does not have any constructor, then it will have default constructor. It is called by super (constructor).

due to, super (constructor) only, multiple inheritance will not be possible, because we have to write super (constructor)

If two super() methods, which will be the first statement and it will throw error.

If we don't write super(), it will called default constructor, then it will throw error because super class have no constructor.

Here, we have to explicitly call super() with argument because in super class have constructor with parameters.

→ If super class have more than one constructor then from sub-class call any one constructor using super-

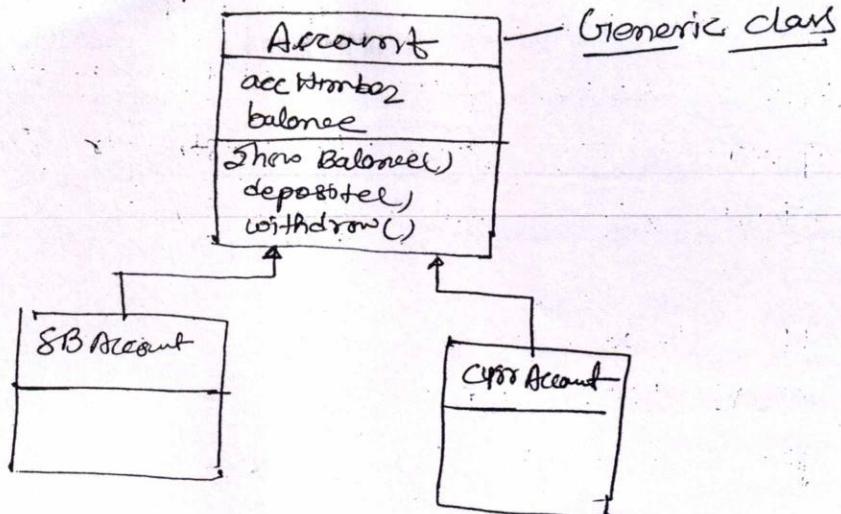
- * In an inheritance tree, if constructor does not contain super() statement, then implicitly Java calls default constructor of the super class by adding super() [call to super].
- * If there is no default constructor in super class, then explicitly we have to call parametrized constructor using super().
- * If we want to parametrized constructor of super class then also explicitly call using super().

~~Super() (call to super)~~ ~~must~~ ~~should be first~~ ~~statement~~
Super() ~~Statement~~ ~~should~~ ~~be first~~ ~~statement~~
line in the constructor

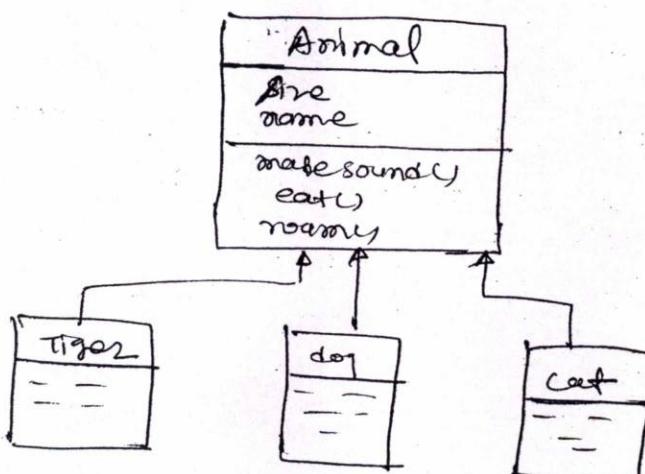
Difference b/w Super and this

- Call to this is used to call the current instance of constructor of some class.
- Call to super is used to call the constructor of super-class.

date
27/04/2013



** Inheritance is used for to reuse the code. we want to change or add some method to class, then, add or delete from Generic class and it will reflect to all sub-class. No need to change in each class.



Here size is almost wrong, makeSound() cannot be same for all, so, Here need of method overriding, so we need override the method of super-class in sub-class.

IS-A relationship

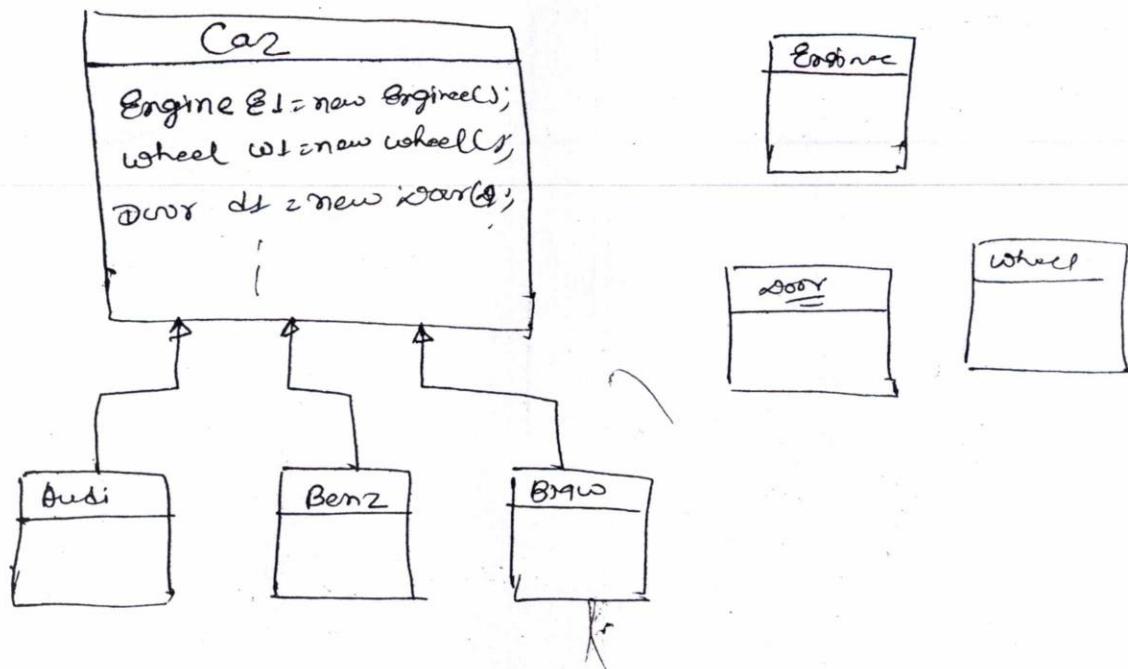
* we inherit the sub-class by seeing the IS-A relationship

→ SUBCLASS IS-A Superclass

→ dog IS-A animal

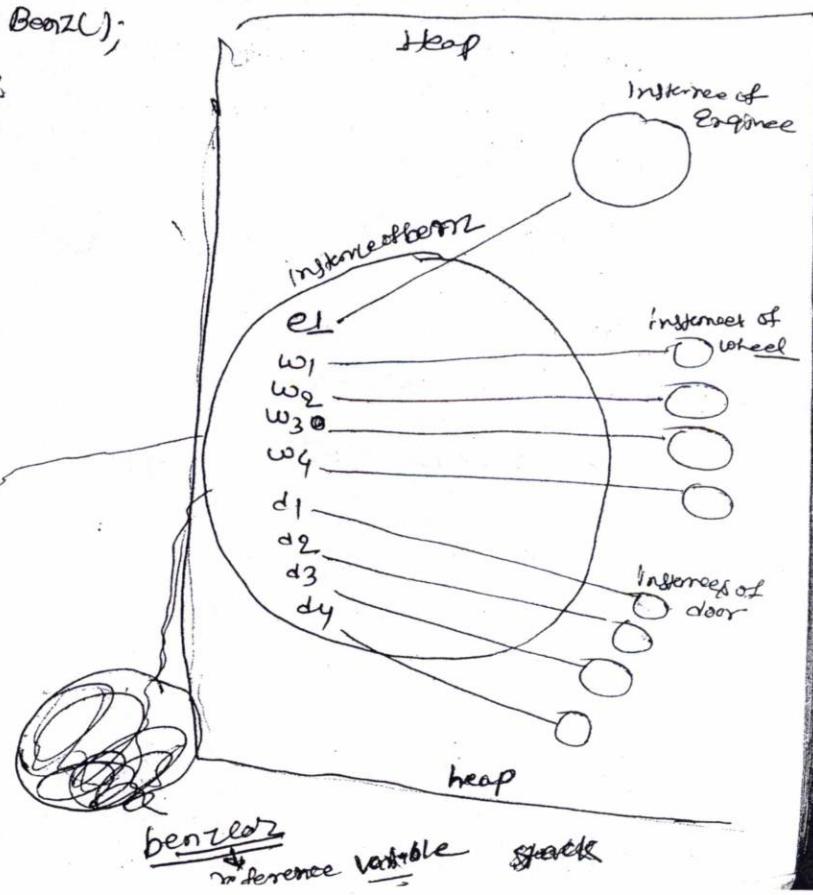
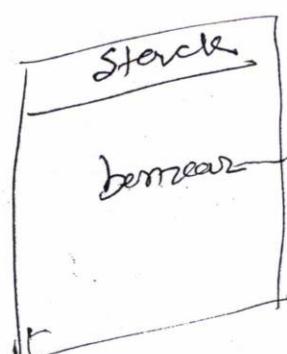
→ Tiger IS-A animal

* we puted the behaviour that is same for all subclass in Generic class but all the sub-class behave differently, like all sub-class makeSound() but all sub-class makeSound() differently. So, there is single code in super-class so, we need method overriding.



- A Car HAS-A Engine } - HAS-A relationship
- A Car HAS-A door }
- A Car HAS-A wheel }
- A Audi IS-A CAR.
- A Benz IS-A CAR. } IS-A relationship
- BMW IS-A CAR.

Benz benzcar = new Benz();
benzcar d1.open()
ref variable ref variable method



51

Class Enginee

```
{  
    void start()  
    {  
        S.O.P("Enginee is start");  
    }  
}
```

(Ans pny) *

Class wheel

```
{  
    void inflate(int i)  
    {  
        S.O.P("wheel is inflated " + i + " psi.");  
    }  
}
```

Class door

```
{  
    void open()  
    {  
        S.O.P("door is open");  
    }  
}
```

Class Car

```
{  
    Enginee e1 = new engine();  
    wheel w1 = new wheel();  
    wheel w2 = new wheel();  
    wheel w3 = new wheel();  
    wheel w4 = new wheel();  
    door d1 = new door();  
    door d2 = new door();  
    door d3 = new door();  
    door d4 = new door();  
}
```

car cl ~
car
cl. w1.inflate()
cl. w2
cl. w3
cl. w4
car start
{ openDoor()
}
↓

```
void StartCar()  
{  
    e1.start();  
}
```

Here in car class, we
call a method of engine
so, we can directly use in
main method to start
like cl.start().

otherwise, if we don't
write here we have
to call cl in main like
cl.e1.start()

```
void openfrontDoor()  
{  
    d1.open();  
}
```

some
explanation

```

class Run6
{
    public static void (String[] args)
    {
        S.O.P(" Pgm starts");
        Car c1 = new Car();
        * c1.startCar(); → Here we are indirectly
        c1.openFrontDoor();   calling through car reference
                               variable
        * c1.w1.inflate(82); → we don't have indirect
                               method in car, so we
                               have to call like
                               this
        c1.d2.open();
        } S.O.P(" Pgm ends");
    }
}

```

O/P put

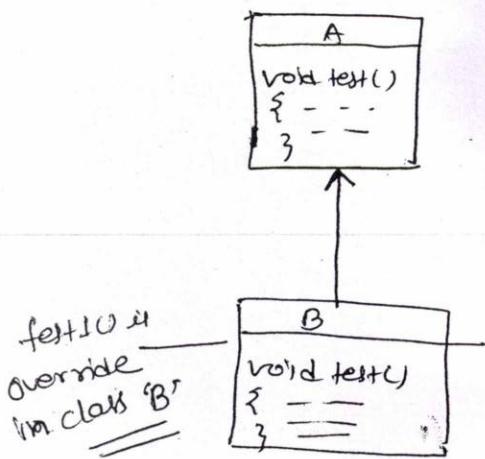
Pgm starts
 Engine is Start
 door is open
 wheel is inflated 82
 door is open
 Pgm ends.

Overloading can
 be done on
Same class
 but overriding can
 occur in different
class only

* Method Overriding :-

- Overriding is a process of inheriting a method from super to sub-class and change the implementation of inherited method according to sub-class requirement.
~~↳ known as method~~
- If I want the method, but I don't want some behaviour so override the behaviour in sub-class or change the behaviour in sub-class of same method.

52



A a1 = new A();
a1.test();

B b1 = new B();
b1.test();

rules to be followed for overriding

- There should be a inheritance.
- method name should match.
- Argument type & no. of argument should match
- Return type should match.

class A

```

{
    void test1()
    {
        System.out.println("test of class A");
    }
}
```

class B extends A

```

{
    void test1()
    {
        System.out.println("test of class B");
    }
}
```

class Run1

```

{
    public static void main(String[] args)
    {
        System.out.println("Program Starts");
        A a1 = new A();
        a1.test1();
        System.out.println("-----");
        B b1 = new B();
        b1.test1();
        System.out.println("Ends");
    }
}
```

O/P

Program Starts
test of class A

test of class B
Ends

```

class C
{
    void test1()
    {
        S.O.P("test1() of C");
    }

    void test2()
    {
        S.O.P("test2() of C");
    }

    void test3()
    {
        S.O.P("test3() of C");
    }
}

```

class D extends C

```

{
    void test2()
    {
        S.O.P("test2() of D");
    }
}

```

class E extends D

```

{
    void test1()
    {
        S.O.P("test1() of E");
    }
}

```

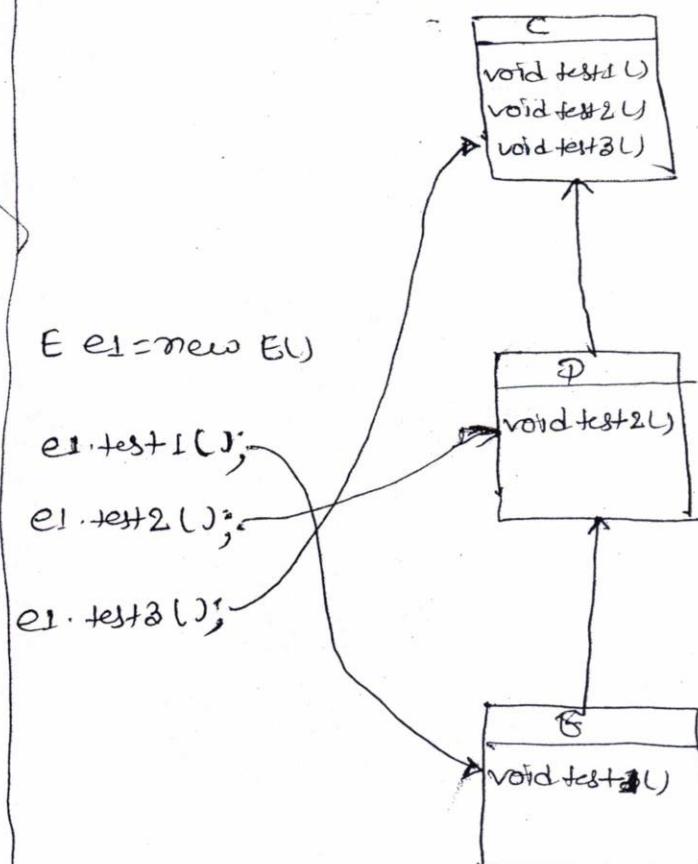
class Run2

```

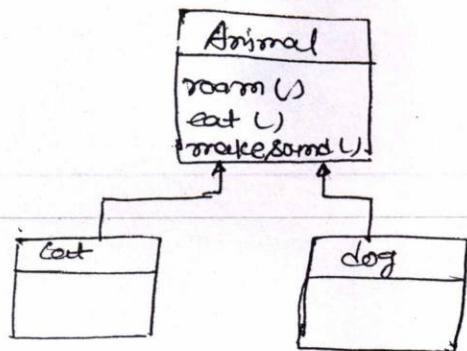
{
    public static
        void main(String[] args)
    {
        E e1 = new E();
        e1.test1();
        e1.test2();
        e1.test3();
    }
}

O/P
    test1 of E
    test2 of D
    test3 of C

```



when a method is called by subclass first it search in own class and execute otherwise search to immediate superclass and execute, --- so on ---



class Animal

{

void roarm()

{

S.O.P ("animal is rooaring");

void eat()

{

S.O.P ("Animal is eating");

void makeSound()

{

S.O.P (" Animal is making sound");

}

Class Dog extends Animal

{

void eat()

{

S.O.P (" dog is eating ");

void makeSound()

{

S.O.P (" dog is barking ");

Bow Bow

}

class Cat extends Animal

{

void eat()

{

 S.O.P("cat is eating");

}

void makeSound()

{

 S.O.P("cat is meowing");

}

}

class Pgm

{

public static void main(String[] args)

{

 S.O.P("pgm starts");

Dog dt = new Dog();

dt.eat();

dt.makeSound();

dt.roam();

Cat ct = new Cat();

ct.eat();

ct.makeSound();

ct.roam();

S.O.P("pgm ends");

?

O/P:-

pgm start

Dog is eating

Dog is barking

Animal is roaming

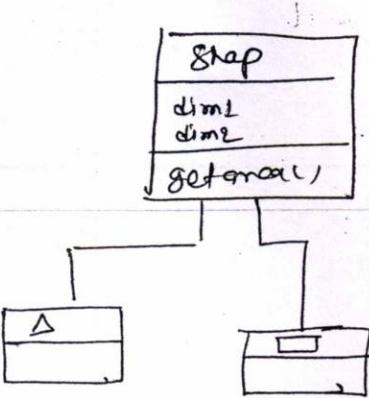
Cat is eating

Cat is meowing

Animal is roaming

pgm ends

54



Class Shape

```

{ int dim1;
  int dim2;
  Shape( int d1, int d2 )
  {
    dim1 = d1;
    dim2 = d2;
  }
  int getarea()
  {
    S.O.P( "Can not find area of "
    return -1;      Shape" );
  }
}

```

we don't want
write, then add
default constructor along
with parameterized
super class shape
shape

Class Triangle extends Shape

```

Triangle( int h, int b )
{
  Super( h, b );
  // dim1 = h;
  // dim2 = b; } — not need to call now
                                — because of super( h, b );
}

```

we have to call
super(h, b) i.e. parameterized
constructor specially
because super
class have
parameterized
constructor.

If we not call
parameterized, then
it will call default
i.e. not present in
super class &
throw error

If we don't want
write super(h, b) then
add super() (default
constructor) in super
class, then it will
execute

* Rectangle is
a keyword
check it out

Class Rectangle extends Shape

```

Rect( int s, int t )
{
  Super( s, t );
  int getArea
  { S.O.P( " " );
}
}

```

class Rect extends Shape

{
 Rect(int l, int b)

{
 } Separ(h,b);

 int getArea()

{
 S.O.P(" Area Of Rect");

} return dim1 * dim2;

class Rumb

{
 P.S.N.M (String[] args)

{
 S.O.P(" Starts");

 Triangle t1 = new Triangle(5,2);

 int areaOfA = t1.getArea();

 S.O.P(" Area of A is " + areaOfA);

 S.O.P(" - _____ ");

 Rect r1 = new Rect(10, 4);

 int areaOfB = r1.getArea();

 S.O.P(" Area of Rect is " + areaOfB);

 S.O.P(" ends ");

}

 Others

O/P

from Starts
getArea of Triangle
Area of A is 5

get Area of rect

Area of B is 40

ends

Assignment

- ① List the all the diffⁿ b/w overloading and overriding.

May 2020

8 may

at

10.20

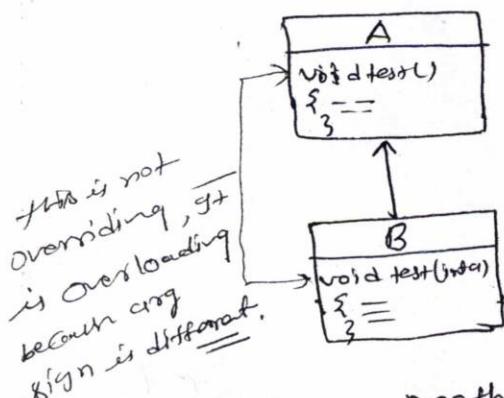
55

diff b/w overloading & overriding

Overloading

overriding

- Multiple method with same name and different argument signature.
- multiple method with same name and some argument signature
- It is compile-time polymorphism.
- It is run-time polymorphism.
- method should present in some class or sub-class → methods present in different class
- we can overload method in same class or sub-class → we can override method in sub-class
- * we can overload static method in java. → we can not override static method in java
 - ↓
 - static it out
- Overloaded method bounded using static binding and type of reference variable
- Overridden method are bounded using dynamic binding based on actual object



B b1 = new B();
 b1.test();
 b1.test(10)

ANSWER

Only static method can not be overridden in java

• Here, methods are overloaded in sub-class
 So, overloading can be happen in same class
 or in sub-class
 but overriding can happen only
 in sub-class

Date
28/09/2013

Concrete — complete
Abstract — incomplete

abstract :-

④ abstract methods :-

These are two types of methods *

(a) ~~concrete~~ concrete methods

(b) abstract methods

→ methods which has both head & body is known as concrete (complete) method

→ ~~methods~~ for ex:- void test1(int a)
 {
 ---;
 ---;
 ---;
 } → body

* methods which has only head. (without body)
is known as abstract (incomplete) methods

→ abstract methods are declared using a keyword abstract:

ex:- abstract void test2(int a);

→ If a class contains abstract method,
then that class should also be abstract class.

→ java does not allow to create an instance of abstract class.

Ex:- abstract class A

{ --- }

abstract void test2 (int a);

56

A a1 = new A(); → X
(throws error) can not create instance because
A is abstract

- An abstract class can contain concrete method
- So abstract methods should be overrided and complete in the sub-class, otherwise sub-class also become abstract.
- * If a class extends abstract class, implement all ~~the~~ abstract methods in sub-class otherwise declare sub-class also a abstract.
- * If a method is abstract in class, then we must make that class as abstract class also otherwise jvm throws error

abstract class A

```
{
    abstract void test();
}
```

class B extends A

```
{
    void test1()
    {
        System.out.println(" test1() of B");
    }
}
```

class Pmt

```
{
    public static void main(String[] args)
    {
        System.out.println(" pmt starts")
        // A a1 = new A(); → throw error bcoz A is abstract
        B b1 = new B();
        b1.test1();
    }
}
```

System.out.println(" pmt ends")

O/P [pmt starts
test1 of B
pmt ends]

* Find error in this and correct the program
abstract class C

```
{  
    abstract void test1();  
    abstract void test2();  
}
```

class A extends C
{
 void test1()
 {
 S.O.P("test1 of A");
 }
}

if was error
in class A so
we must add
abstract key
word in class A

find error
and correct

class E extends A
{
 void test2()
 {
 S.O.P("test2 of E");
 }
}

V.V.T

class Run2
{ p.s.m.v & strg }
{ E e = new E();
 e.test1();
 e.test2();
}

Ans → Add abstract at class A

because class A have one incomplete method
complete method, ~~but~~ but one abstract
method test2(). So, if any class contain
method, then class should be abstract

→ class A have ~~one~~ one complete method
and test2() is not completed in class A,
then, it will be present as abstract
method in class A because class A
is inherited from class C.

→ Here we can create an instance of
only class E, because it contain
both the complete method

57

* abstract class shape

```
{ int dim1;  
  int dim2;  
  abstract int getArea();  
}
```

class Triangle extends shape

```
{  
  Triangle(int d1, int d2)  
  {  
    dim1 = d1;  
    dim2 = d2;  
    int getArea()  
    {  
      return (dim1 * dim2) / 2;  
    }  
}
```

class Rect extends shape

```
{  
  Rect(int d1, int d2)  
  {  
    dim1 = d1;  
    dim2 = d2;  
    int getArea()  
    {  
      return dim1 * dim2;  
    }  
}
```

class Run5

```
{ public static void main(String[] args)  
{  
  System.out.println("Program starts");  
  Triangle t1 = new Triangle(10, 5);  
  System.out.println("Area of triangle is:" + t1.getArea());  
  Rect r1 = new Rect(6, 8);  
  System.out.println("Area of rectangle is:" + r1.getArea());  
  System.out.println("Program ends");  
}
```

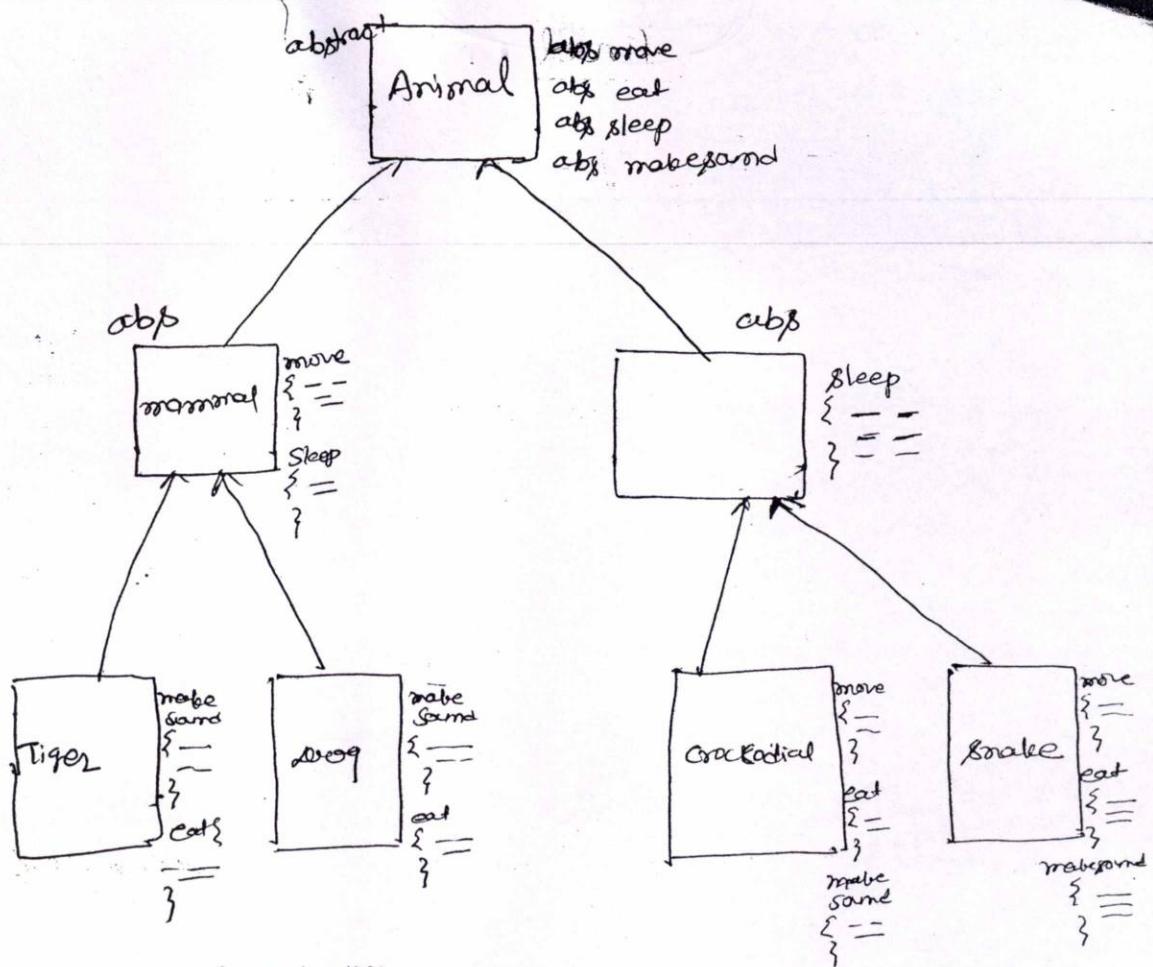
O/P

Program starts

Area of triangle is: 25

Area of rectangle is: 48

Program ends



try to implement this

abstract class F

```

{
    abstract void test1();
    abstract void test2();
}

```

abstract class G1 extends F

make it abstract

```

void test1()
{
    S.O.P("test1 of class G1");
}

```

```

void test2(int a)
{
    S.O.P("test2 (int) of class G1");
}

```

58

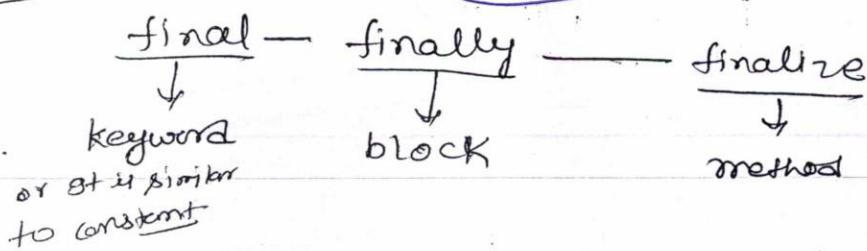
* Here we have to add abstract to class G1

because still there test2() with not congruent
is incomplete present in class G1 as abstract.

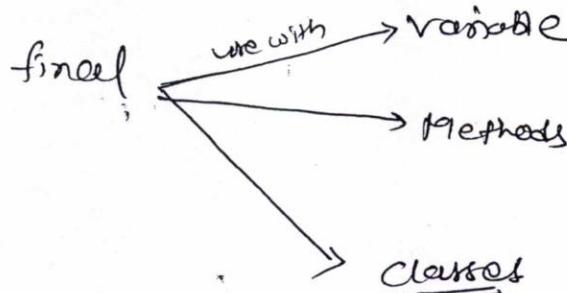
void test2 (int a) is overloaded in class G1, not
overridden.

diffn b/w

Final



for local
variable, there
is no concept
of static
and non static



- Final is similar to constants
- final variable must be declared & initialized in same line

<pre>final int i=10; ✓</pre> <pre>final int i; } X</pre>	{ general practice is write final variable in upper case <code>final double PI = 3.14;</code> <code>final int I = 10;</code>
---	--

- * final variable value can not be changed or Alter.
- * final variables must be declared and initialize in the same line.
- * final methods can not be overridden

Class H

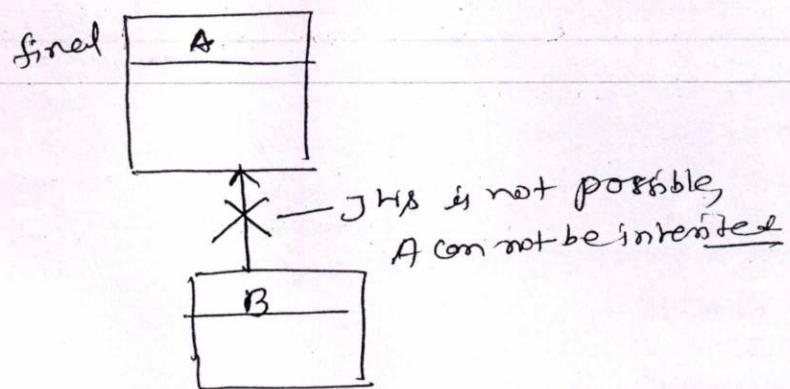
```
{
    final void testL()
    {
        S.O.P("testL() in H");
    }
}
```

Class I extends H

```
{
    void testL()
    {
        S.O.P("testL() in I");
    }
}
```

error at compile time
 // it will show
 here, because
 final method can
 not be overridden

* final classes can not be inherited.



final class A

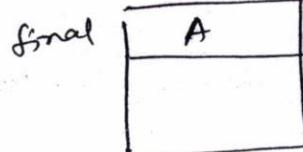
```
{  
    void test()  
    {  
        // code  
    }  
}
```

class B extends A

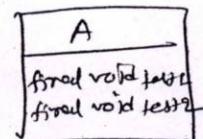
```
{  
    void test()  
    {  
        // code  
    }  
}
```

throws error.
because class A is
final so can not be
inherited

differently ~~class~~ final class and all method in
a class



and



59

- ① An abstract can contain concrete method → Yes
- * From we create an object of abstract class → No
- * If abstract class contain a static members, access those static members by using class name.
- * How do you access concrete non static method of abstract class? →

- Ans → Create a class by extending ~~an abstract class~~ abstract class.
- Override and complete all abstract methods in sub-class.
 - Create an instance of sub-class and access non-static members.

abstract class

{

~~abstract final void test();~~

}

→ not legal, i.e this
not legal combination

Static final int i = 10

why
non static
member does
not participate
in inheritance

- no meaning, because static number does not participate in inheritance.
- explicitly no need to write final
- by default all static members are final because static members never participate in inheritance.
- but it will not throw error by final

~~static abstract void test();~~

→ illegal combination

Implementation of Animal class

Abstract class Animal

{
 abstract void sleep();
 abstract void eat();
 abstract void makeSound();
 abstract void move();
}

Abstract class Mammal extends Animal

```
{  
    void move()  
    {  
        S.O.P("Mammal is moving");  
    }  
}
```

```
void sleep()  
{  
    S.O.P("Mammal is sleeping");  
}  
}
```

Abstract class Reptile extends Animal

```
{  
    void sleepy()  
    {  
        S.O.P("Reptile is sleeping");  
    }  
}
```

Class Dog extends Mammal

```
{  
    void eat()  
    {  
        S.O.P("The dog is eating");  
    }  
    void makeSound()  
    {  
        S.O.P("The dog is Bow Bow");  
    }  
}
```

Class Tiger extends Mammal

```
{  
    void eat()  
    {  
        S.O.P("The tiger is eating");  
    }  
    void makeSound()  
    {  
        S.O.P("The tiger is growling");  
    }  
}
```

class Snake extends reptile

{ void eat()

{ S.O.P ("The snake is eating");

} void makesound()

{ S.O.P ("The snake is making sound");

} void sleep()

{ S.O.P ("The snake is sleeping");

class Crocodile extends reptile

{ void eat()

{ — — }

void sleep()

{ — — ? ? }

class Run

{ public static void main (String [] args)

{ S.O.P ("Program starts");

Tiger t1 = new Tiger();

t1.eat();

t1.sleep();

t1.makesound();

t1.move();

S.O.P ("— — — — —");

Snake s1 = new Snake();

s1.move();

s1.sleep();

s1.makesound();

s1.eat();

S.O.P ("— — — — —");

Dog d1 = new Dog();

d1.move();

d1.sleep();

d1.makesound();

d1.eat();

? ?

O/P

Program starts

Tiger is eating

Mammal is sleeping

Tiger is growling

Mammal is moving

— — — — —

Snake is moving

Snake is sleeping

Snake is making sound

Snake is eating

— — — — —

Dog is moving

Mammal is sleeping

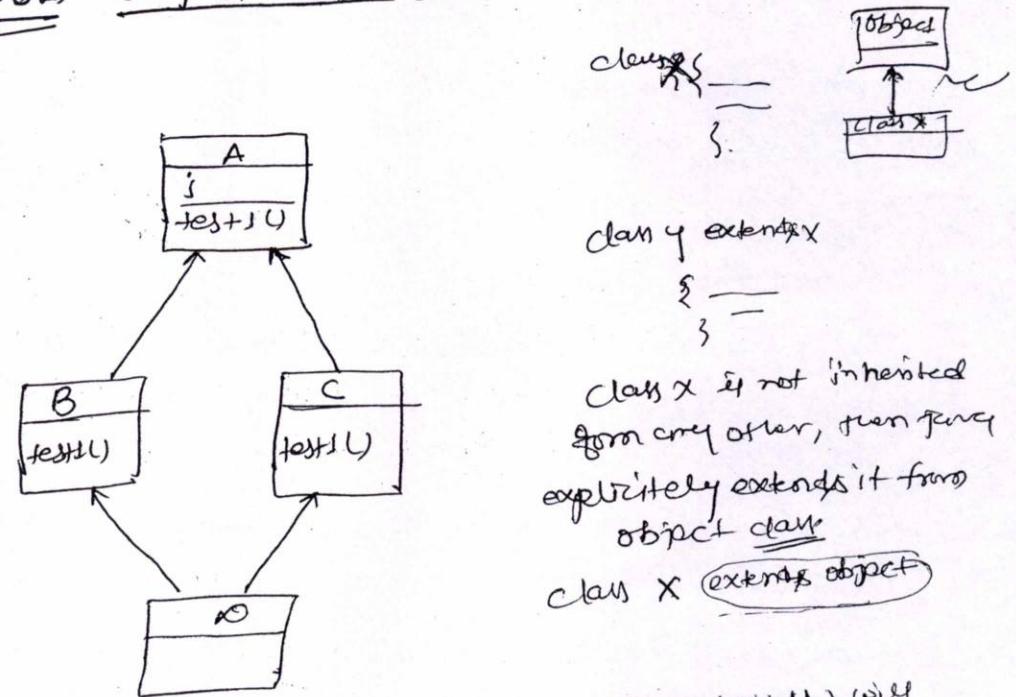
Dog is Bow Bow

Dog is eating

— — — — —

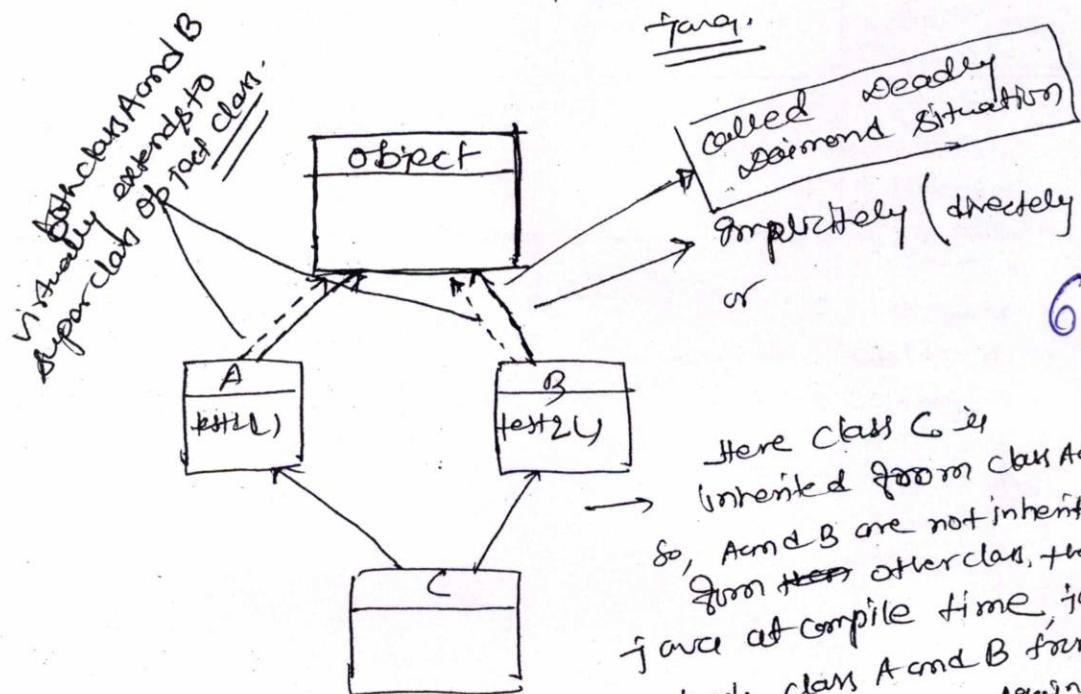
* Why multiple inheritance is not accepted

- * Super class in java is object class.
- * In java, Any class, that does not extend from any class then, it implicitly extends object class.



`Object d1 = new A();`
`d1.test()`

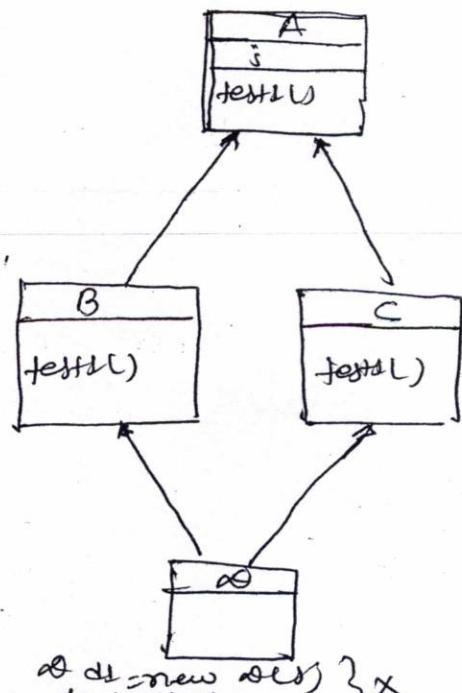
X then which `test()` will be called so, it is not allowed in java i.e. multiple inheritance is not allowed in java.



Here class C is inherited from class A and B. So, A and B are not inherited from ~~the~~ other class, then once at compile time, java extends class A and B from `Object` class, then again it leads to diamond problem. In java is not allowed.

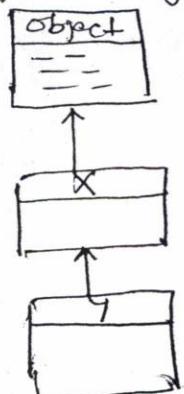
Diamond problems:-

- Class A has instance variable *i* and a method `test1()`.
- Class B and C are inherited from class A.
- Both class B and class C overrides `test1()` method.
- Class *d* is inherited from both class C & class B.
- If we call `test1()` method of *d* instance, then there is a confusion, to execute a `test1()` implementation. (`test1()` from both class B and class C are inherited to *d*).
- This situation leads to an unexpected output at run-time.
- There is some special technique is required to solve this problem, since Java is simple programming language, it does not allow multiple inheritance.



Object class:-

- * In Java, there is fundamental class called Object class.
- * Any class that be created, is implicitly inherited from Object class, if a class is not explicitly inherited from other class.

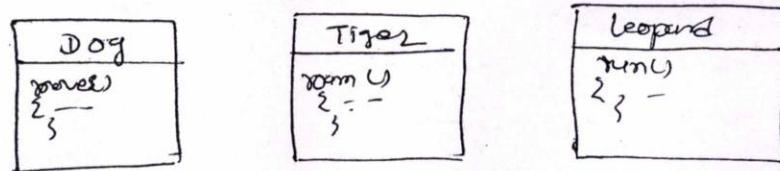


class X extends object
 {
 } → Java add it a
 compile time implicitly

class Y extends X
 {
 }?

* all classes that we create in java are directly or indirectly inherited from Object class.

advantage of inheritance



reusability
polymorphism
standardization
or consistent

- * → All have some method with different names, this is an award situation in java, inheritance provides puts some rule or standardized the design that leads to reflex effect all other classes must follow that rule or standard.

Advantage

- (i) Reusability
- (ii) polymorphism
- (iii) provide standardization or consistent, i.e. it put some rules that all classes should follow.
- (iv) provide hierarchy classification.

class name
general practice
first letter
upper and other
in lower case

- Abstract class is a class that contains at least one abstract method
- It is not fully abstract because it can contain concrete methods also
- We can not create object of abstract

class

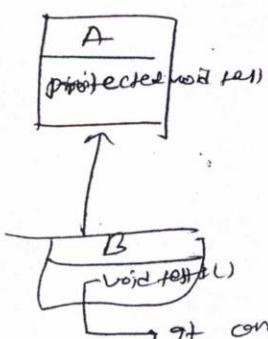
3. * Abstract classes can contain static members
or not?

four access specifier

- ① private → It can be used by current class (less visibility)
- ② default → It is used in package (It is more friendly) (~~(less)~~)
- ③ protected → Can be used by current class and any other class
- ④ public → It can be used by anyone
the intended sub-class

increasing visibility
from top to down

- A sub-class always have same or more visible access specifier than super class



→ It can be protected or public only.
It can not be private or default.

* multiple inheritance using class is
not possible but using interface is
possible

A class can inherits ~~more than~~ any no.
of interfaces.

C c₁;
C c₂ = new C();

- * diff'n b/w stateful and non-state
- * what is iterator
- * what is API
- * what is filter in J2EE
- * Exception handling questions in core J2EE

- * ~~lambdas~~ * what is constructor.
 - = abstract class, and interface
 - * Overriding & overload in diff or in
 - * oops concept: — polymorphism, interface encapsulation
abstraction
 - * final finally & finalize in diffn
 - = what ~~each~~ package
 - * dead-lock, Serialization, deserialization,
Synchronization
 - * class & object — ✓
 - * Annotation, @ override
 - * Threading — (J2EE)
 - * super & this diffn
- (Site
Developer Book
.com)

- ### J2EE
- * what is jvm —
 - * what is diff CGI and servlet why use servlet now-

- * servlet life cycle, ✓
- * do, & post not diffn —
- * How many
- * diff JSP & servlet
- * what is request dispatcher
- * Session what
- * what is Servlet Context
- * Servlet Session
- * Cookies what

How to add cookies and delete cookies

- * JSP page what life cycle of JSP
- * Java bin, what

Select Employee name,
customer name, Id, address
from customer
where amount > max(amount)

Select customer -
from customer
where

order	name	Id	amount	-

① Sales technologies

→ core func

→

② sales matrix

for testing
Babylon & pure brother

* normal testing

Accel free Software
305, pride kurmar Semate
Semapatti Bapat Rd

Pune - 411016

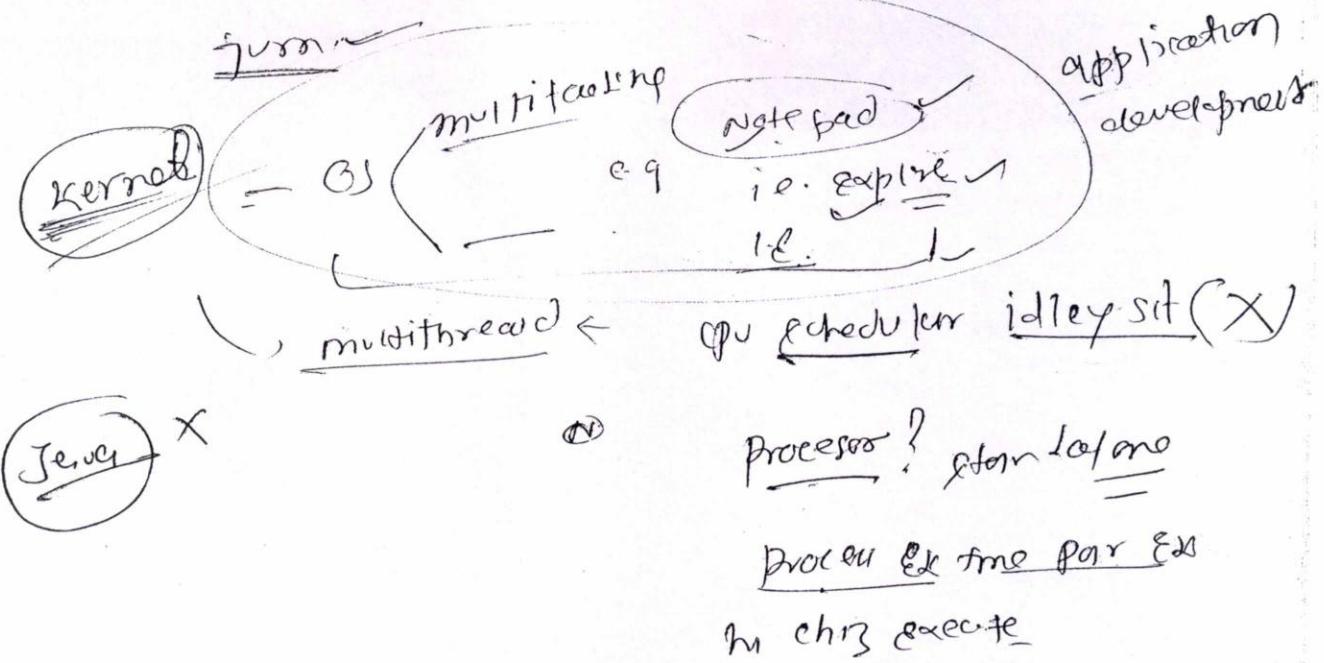
Lavond mark -
Marriott hotel, panchaloon

Karshi (Pune spider)

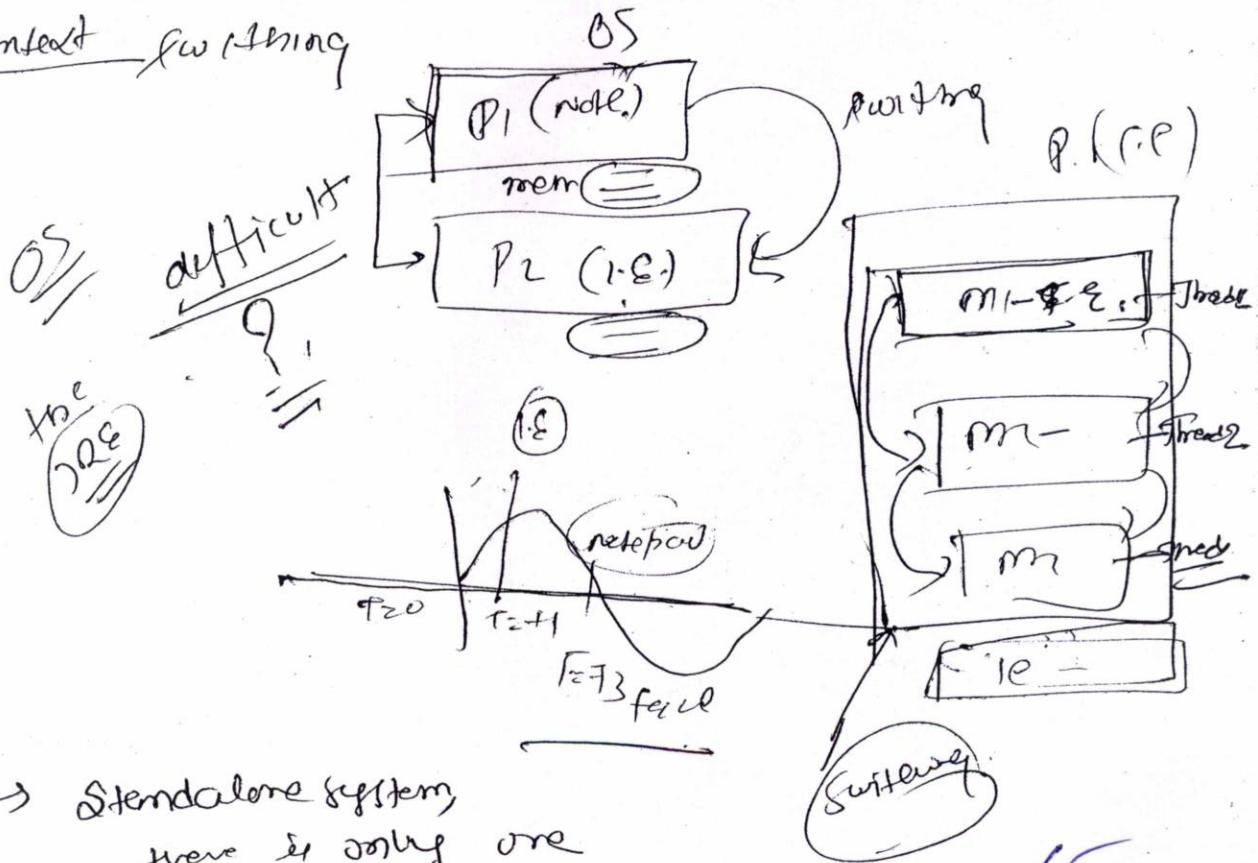
9096055556

→ JVM is specification, that is used for low memory management

→ J.R.S — Physical JVM, like implement of



Context switching



→ Standalone system, there is only one processor,

→ When we write `javac` command there
only three thread are created by
JVM

(i) main thread

(ii) Garbage collector

(iii) thread scheduler → CPU scheduler

→ main thread → CPU scheduler & G.C. thread

→ It is background

process

→ main thread is front thread

→ CPU Scheduler allocate CPU time for
main thread & G.C.

Garbage Collector

→ It is demon thread, used to clear, clean
flush memory automatically, it is called

J.R.E.

[Runtime.GC.] → request G.C.

not
mem
but
write
no

Interface

- * Interface is another type definition block in java.
- * When we are creating an interface, we have created a one more type.
- * To create interface, use a keyword called interface.

Syntax:

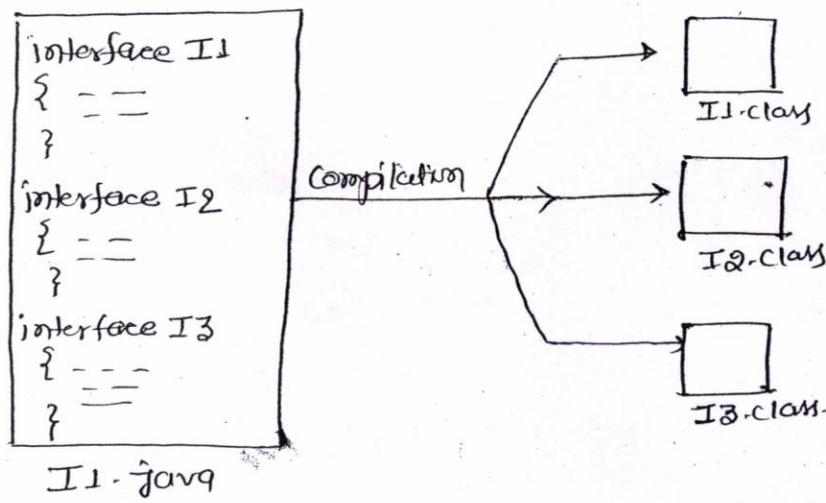
```
interface InterfaceName
{
    ---  

    ---  

    ---  

}
```

- * After compiling a file, which has interfaces, java generate separate class file for each interface.



- * Interface should not contain static methods.
- * Interface is 100% abstract class.

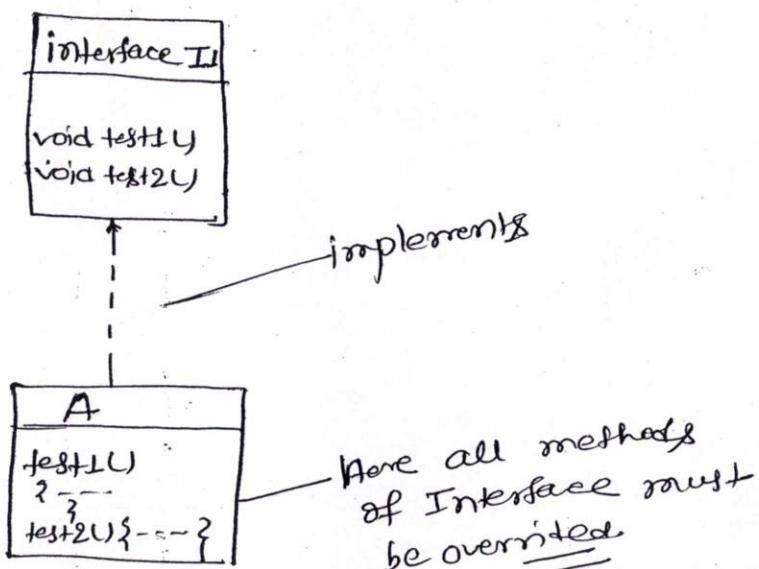
~~abstract~~ interface I1

not mandatory,
but if we
write no
error

~~Static final~~ int I = 10
~~public abstract void test1();~~
~~public abstract void test2();~~

66

- Interface is 100% abstract class.
- In interface, by default methods are public and abstract.
- Static method are not allowed in interface.
- In interface, by default all variables are static and final.
- Since the variables are final, we have to declare and initialize variables in the same line inside the interface.
- We can not create a non-static variable in interface.



class A implements II

{

}

- * We can inherit members from interface into a class.
- * To inherit members from interface into a class, use a keyword "implements" i.e. a class can implement interface.
- * Above example class A implements interface II.

** If a class implements interface, override all the methods in the sub-class otherwise mark sub-class as abstract.

→ Java does not allow to create an instance for interface because it is abstract class.

Interface II

```
{
    void test1();
    void test2();
}
```

class A implements II

```

public void test1()
{
    S.O.P("test1() of class A");
}

public void test2()
{
    S.O.P("test2() of class A");
}

```

class Run1

```

public static void main (String [] args)
{
    A a1 = new A();
    a1.test1();
    a1.test2();
}

```

O/P

test1 of class A

test2 of class A

execution sequence

→ In sub-class i.e. class A
we can not put or override
less visibility access specifier
for method compare
to super class i.e.

II.

67

Interface I2

```
{  
    void test1();  
    void test2();  
}
```

abstract class B implements I2

```
{  
    public void test1()  
    {  
        S-O-P("test1 of class B");  
    }  
}
```

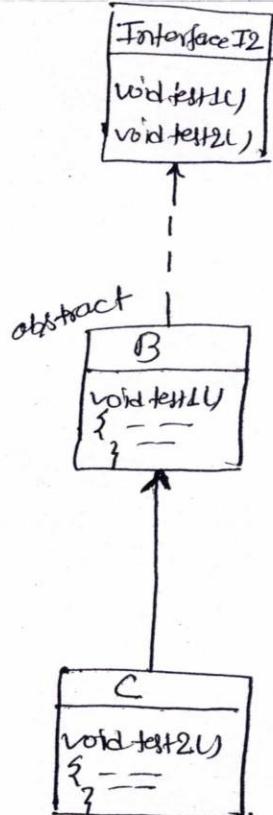
class C extends B

```
{  
    public static void test2()  
    {  
        S-O-P("test2 of class C");  
    }  
}
```

class Run2

```
{  
    public static void main(String[] args)  
    {  
        S-O-P(" program starts");  
        C c1 = new C();  
        c1.test1();  
        c1.test2();  
        S-O-P(" program ends");  
    }  
}
```

O/P program starts
test1 of class B
test2 of class C
program ends



Interface I3

```
{  
    void test1();  
}
```

interface I4 extends I3

```
{  
    void test2();  
}
```

class o extends implements I4

```
{  
    public void test1()  
    {  
        S.O.P("test1() of class o");  
    }  
  
    public void test2()  
    {  
        S.O.P("test2() of class o");  
    }  
}
```

Class Run3

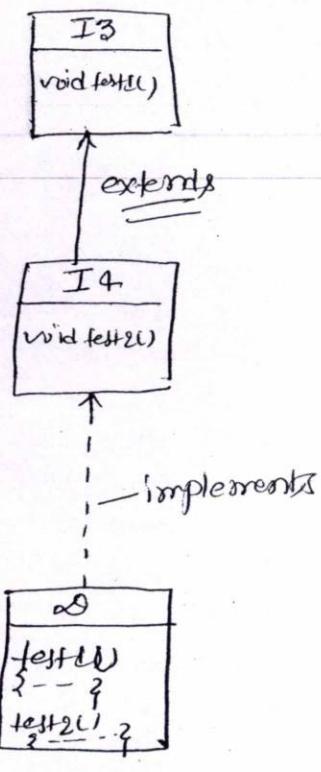
```
{  
    public static void main (String args)  
    {  
        S.O.P("Program starts");  
        o d1 = new O();  
        d1.test1();  
        d1.test2();  
        S.O.P("Program ends");  
    }  
}
```

O/P Program starts

test1() of class o

test2() of class o

Program ends.



interface I5

```
{ void test1();  
}
```

interface I6

```
{ void test2();  
}
```

class E ~~extends~~
implements I5, I6

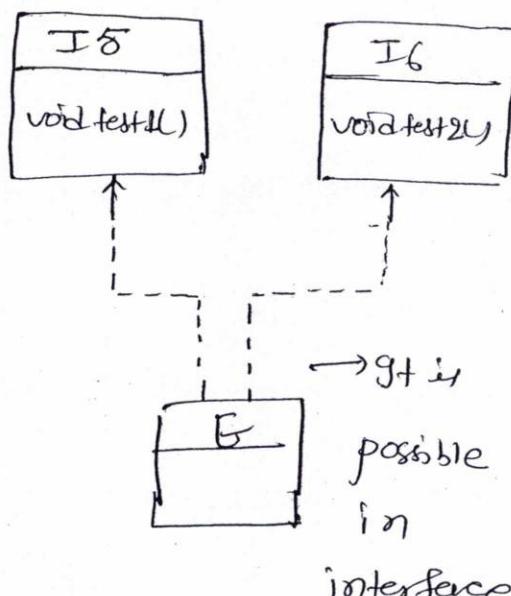
```
{  
    public void test1()  
    {  
        System.out.println("test1 of class E");  
    }  
  
    public void test2()  
    {  
        System.out.println("test2 of class E");  
    }  
}
```

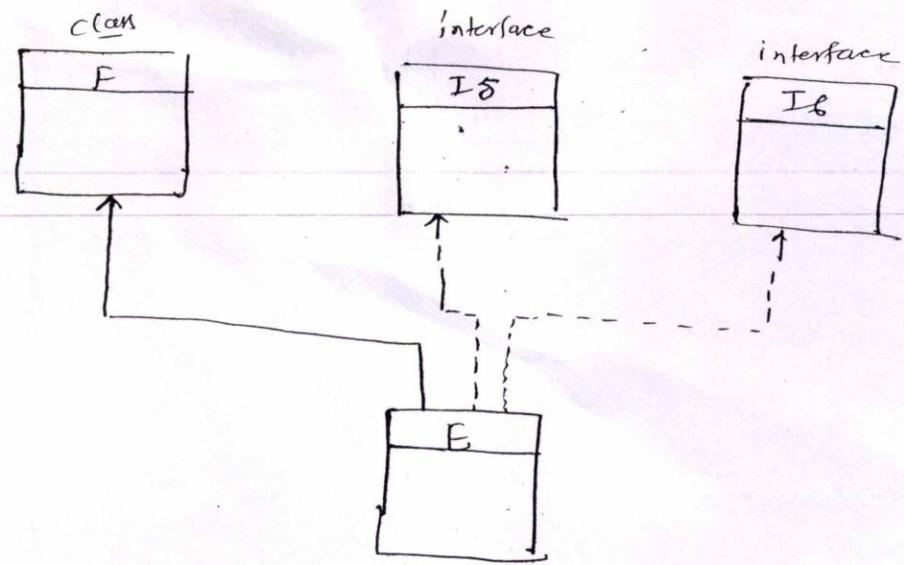
class Run5

```
{  
    public static void main  
        (String[] args)  
    {  
        System.out.println("Program starts");  
        E e1 = new E();  
        e1.test1();  
        e1.test2();  
        System.out.println("Program ends");  
    }  
}
```

Output
Program starts
test1() of E
test2() of E
Program ends.

Class → interface
Implementation
not possible





Class E extends F implements I5, I6

{

}

- * In order to write, upper, first is extends
then implements - (order must be followed)

→ Multiple inheritance is possible using interfaces.

→ (A class can implement multiple interfaces).

ex:- Class A implements I1, I2, I3

{

}

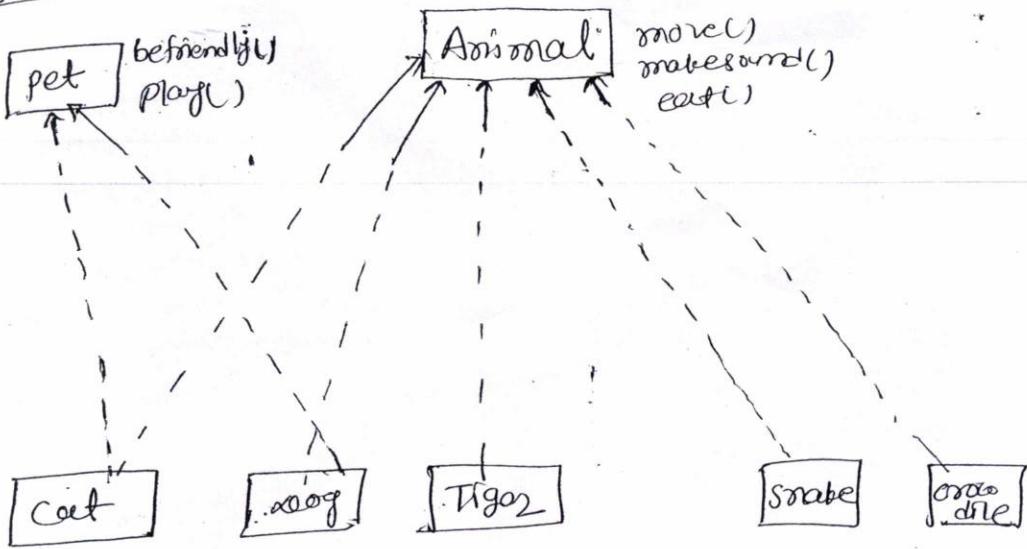
→ A class can inherit from a class and also from an interface but the [order should be extends then implements.]

ex:- Class A extends B implements I1, I2

{

}

Date
20/05/12



* Interface set the rule, that what method should sub-class contain. but implementation of that method depends on sub-class.

- * API - collection of class.
- * Interface provides clear visibility to the programmer, what methods class should contain, and it avoid to see some method with different name contain in different class (like run, move, run, ride-----)

* Interface use → (take example of mobile network → to connect police, from any telecom company, dial 180 → connect to police)

* big advantage of interface is polymorphism.
* " " " " " to provide multiple inheritance.

* Use or Advantages of Interface -

- * Using interface, we can achieve multiple inheritance.

Difference b/w abstract & interface

abstract class

- abstract class does not support multiple inheritance.
- abstract class can contain concrete method as well as abstract method.
- abstract class should be extended using ~~"extends"~~ "extends" keyword.
- variables declared in abstract class may contain non-final variable.
- methods declared in abstract class may be public, private, etc. i.e. not by default public.
- A abstract class defn begins with keyword "abstract".
- any class can extend an abstract class.
- All abstract method must have abstract access modifier.
- abstract class Shape
 - {
 - abstract void area();
 - abstract void perimeter();
 - void someMethod();
 - }

Interface

- Interface support the multiple inheritance.
- Interface contain only abstract method.
- Interface should be implemented using keyword "implements".
- Variable declared in interface by default static and final.
- Methods declared in Java interface are by default public and abstract only.
- An interface definition begins with keyword "interface".
- Only interface can extends interface.
- All methods in an interface must not have abstract access modifier.
- Interface Shape
 - {
 - void area();
 - void perimeter();
 - }

70

Type Casting & Polymorphism:-

int i = 100 → hetrogeneous statement

int i = 100.235 → hetrogeneous statement — X

→ java allow only homogeneous statement

so, to convert hetrogeneous to homogeneous through
type casting.

int i = (int) 100.235 → now it become homogeneous statement, by type casting.

Type casting - Converting one type into another type is known as type casting.

* java allow only homogeneous statement.

* If there is a hetrogeneous statement in a program, at the time of compilation, JVM throws error.

* If we want to compile successfully, we have to make hetrogeneous statement to homogeneous statement by converting type.

There are two types of Casting:-

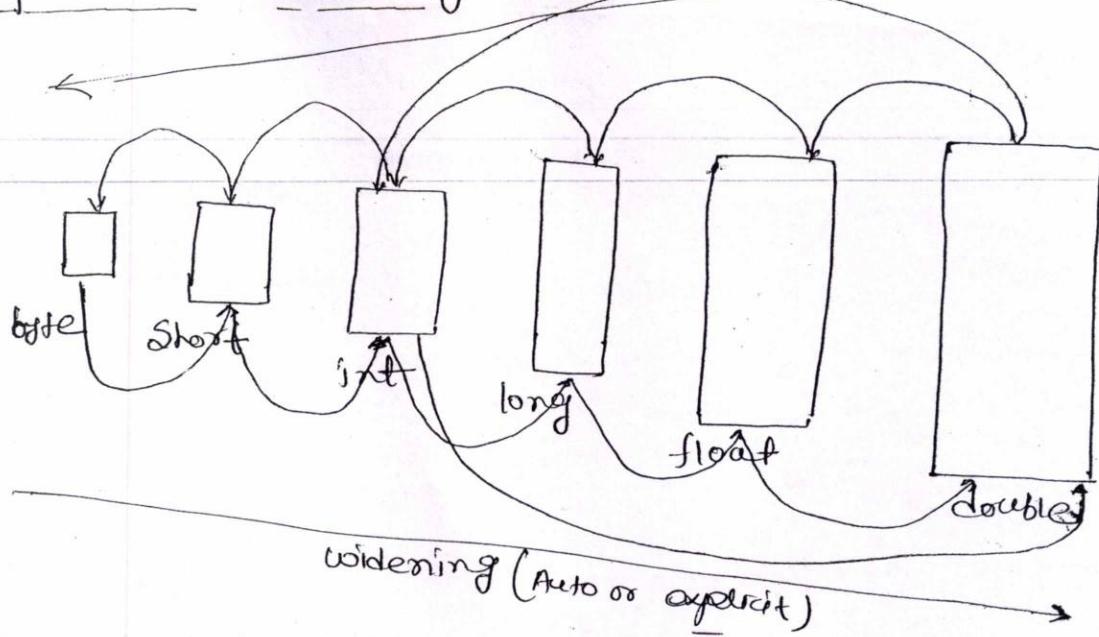
① primitive Casting-

② derived Casting -

→ Converting a primitive type to another primitive type is known as primitive casting. ex- int to double, double to short etc.

→ Converting a derived type to another derived type is known as derived casting.

* primitive Casting: — Narrowing



`double d = (double) 10` → explicit widening, either explicitly,
`double d = 10` → Auto widening.
 If we don't write, Java can do explicitly at compile time.
`int i = (int) 20.53` → explicit narrowing
`int i = 20.13 X`

- * Converting a smaller type into bigger type is known as "widening" (Assigning smaller data into a variable of bigger type).
- * We can do widening explicitly or Java can do automatically. If Java is widening, it's called auto-widening.
- * Converting bigger type into smaller type is known as "narrowing" (Assigning bigger data into a variable of smaller type).
- * Narrowing is always explicit.
- * While narrowing, we might lose some data.

`int i = (int) 20.53` — explicit narrowing.

`int i = 20.13 X` we can do auto narrowing, it will

class Run1

```
{  
    public static void main(String[] args)  
    {  
        double d1 = 20; // Auto widening  
        double d2 = (double) 30; // explicit widening  
        System.out.println("d1=" + d1);  
        System.out.println("d2=" + d2);  
        System.out.println("-----");  
        int i = (int) 39.456; // explicit narrowing  
        // int i = 39.456; - it will throw auto narrow  
        // not possible  
        System.out.println("i=" + i);  
    }  
}
```

O/P

$$\begin{array}{r} \cancel{d1=20.0} \\ \cancel{d2=30.0} \\ \hline i=39. \end{array}$$

NOTE:-

A variable of type `double`, can be used to store any data related to number.

class Run2

```
{  
    public static void main(String[] args)  
    {  
        System.out.println("main starts");  
        test1(10);  
        System.out.println("main ends");  
    }  
    static void test1(double a)  
    {  
        System.out.println("a=" + a);  
    }  
}
```

Auto widening &
cast

O/P

main starts
a=10.0
main ends

class Run2

```
{  
    public static void main (String [] args)  
    {  
        System.out.println ("main starts");  
        test1 (10);  
  
        System.out.println ("test1 ends");  
    }  
}
```



static void test1 (double a)

```
{  
    System.out.println ("a = " + a);  
}
```

~~static~~ ~~double~~ test2 (int a)
{
 a += 5;
 return a; // Auto widening
}

O/P main starts

a = 10.0

15.0

main ends

class Run3

```
{  
    public static void main (String [] args)  
    {  
        System.out.println ("main starts");  
        short s = 10;  
        test1 (s);  
        System.out.println ("main ends");  
    }  
}
```

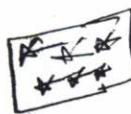
72

static void test1 (int a)

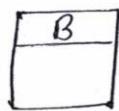
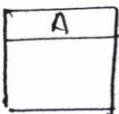
```
{  
    short s = (short) a; // explicitly we have to  
    // convert int into short  
    // to use short s in the  
    // method.  
}
```

Derived Casting :-

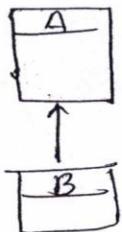
- * Creating an instance of a class, and assigning that instance into a reference variable of different types. is known as derived Casting.



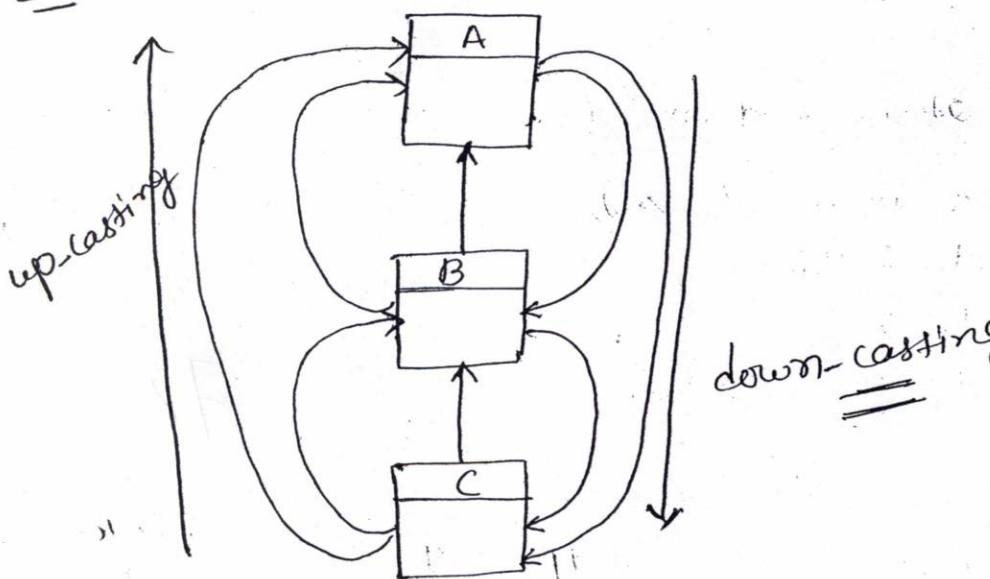
Derived Casting is possible only when there is an "IS-A" relationship between class:



A ar = (A) new B(); ~~X~~ *because there is no inheritance*



A ar = (A) new B(); ~~X~~ *0.88*



→ A a1 = (A) new C(); — explicit up casting

A a1 = new C(); → Auto up-casting

instance of C

→ C c1 = (C) new A(); X

↳ explicit down casting

→ If will not give error at compile time
but If will give error at run time.

C c2 = (C) a1; → only to down cast to
upcasted object

→ Converting sub-type into super-type is known
as up-casting.

→ we can up-cast explicitly or Java can upcast
implicitly (auto upcasting).

→ If Java implicitly doing up-casting, that is
known as auto-upcasting

A a1 = (A) new C(); → explicit upcasting ✓

A a1 = new C(); → Auto upcasting ✓

→ Converting super-type to sub-type is known as
down casting.

→ down casting is always explicit

** If we create an instance of super-class and
assign it to variable of sub-type by explicitly
down casting, then there is no error while compiling
but there is run-time error. (direct down casting
is not possible) ✓

C c1 = (C) new A(); → no compilation error

↳ i.e. get run time exception

73

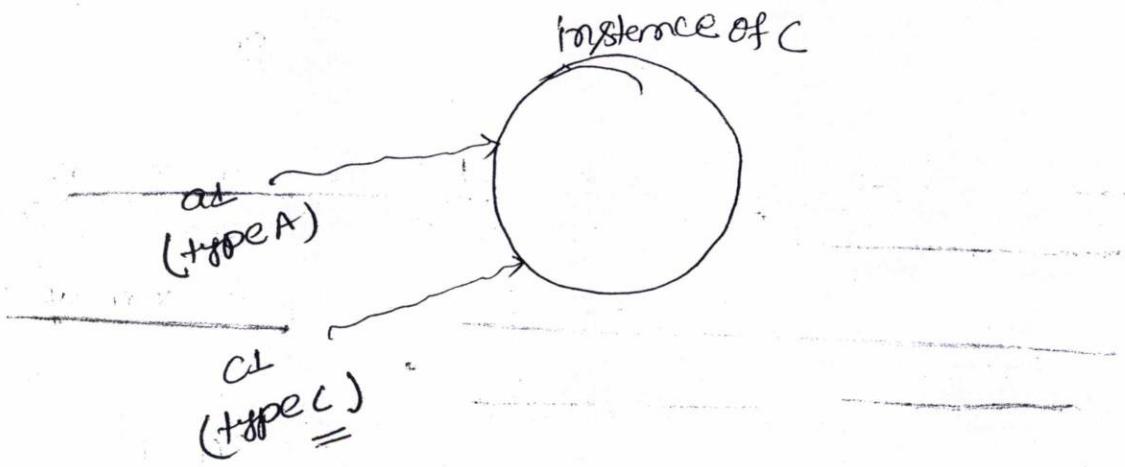
- down casting is used only to down cast
the upcasted instances or object

A a1 = new C();

→ a1 contains upcasted object

C c1 = (C)a1;

→ downcasted the upcasted object



- * If we create an instances of Sub-class and assign it to variable of Super-class is called up-casting.

- Interpretation — execution line by line

Date

3/05/13

```

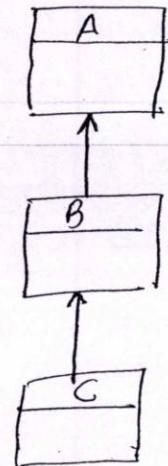
class A
{
}
  
```

```

class B extends A
{
}
  
```

```

class C extends B
{
}
  
```



```

class Run1
{
}
  
```

```

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

```

    System.out.println("program starts");
  
```

```

    A a1 = (A) new C(); // explicit upcasting
  
```

~~A a1 = new C(); // Auto-upcasting~~
 variables won't just to show that either explicit or auto can be happen.

```

    // C c1 = (C) new A(); // downcasting // Runtime errors
  
```

```

    C c1 = (C) a1; // down casting works
  
```

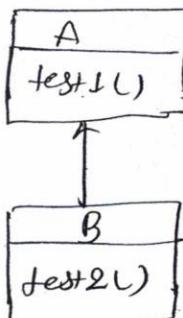
```

}
    System.out.println("program ends");
}
  
```

Q1

prog start

prog ends



```

B b1 = new B();
  
```

```

    b1.test1(); ✓
  
```

```

    b1.test2(); ✓
  
```

```

A a1 = b1 ✓
  
```

```

    a1.test1(); ✓
  
```

```

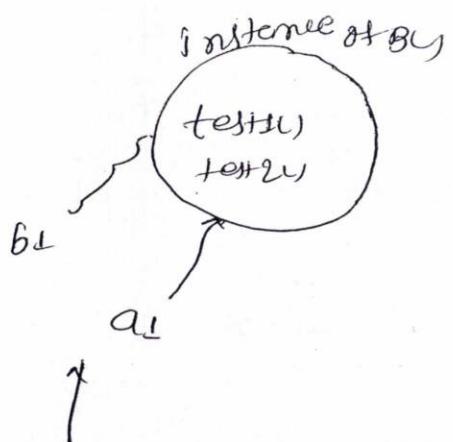
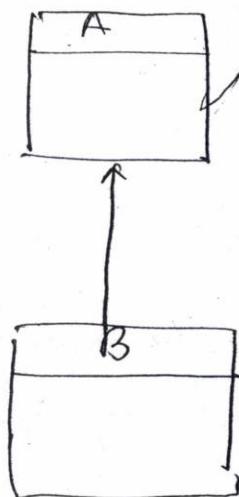
    ... more ... ✓
  
```

74

face call methods
base on only reference
no. compile

** NOTE:

→ At the time of compilation, java compiler verifies the called method are exist in the class, (verifies in the class based on reference variable type).



Here, a1 is pointing to instance of B, but it does not call the test2() method. because in the a1 is downcasted, but it still can not call test2().

B b1 = new B();
b1.test1(); ✓
b1.test2(); ✓

A a1 = b1; ✓
a1.test1(); ✓
a1.test2(); ✗ error

a1 is type of "A",
in class A, there
is not test2().

false example

dog can not behave like cat but dog can behave like animal and or mammal, when it behave like mammal or animal, dog forget to behave its own feature like Bark(), so, Here dog ~~object~~ before like after object of Animal or mammal feature.

* Polymorphism

→ A single object which takes different forms in its life cycle is known as polymorphism.
(An object behaves as a different object). That is called as polymorphism.

Class A

```
{ void test1();  
{ S.O.P("test1 of in class A");  
}
```

Class B extends A

```
{ void test2();  
{ S.O.P("test2(, in class B");  
}
```

Class Run

```
{ public static void main(String args[]);  
{ S.O.P("program starts");  
 b1.test1();  
 b1.test2();  
 A a1 = b1;  
 a1.test1();  
 } S.O.P(" program ends");  
}
```

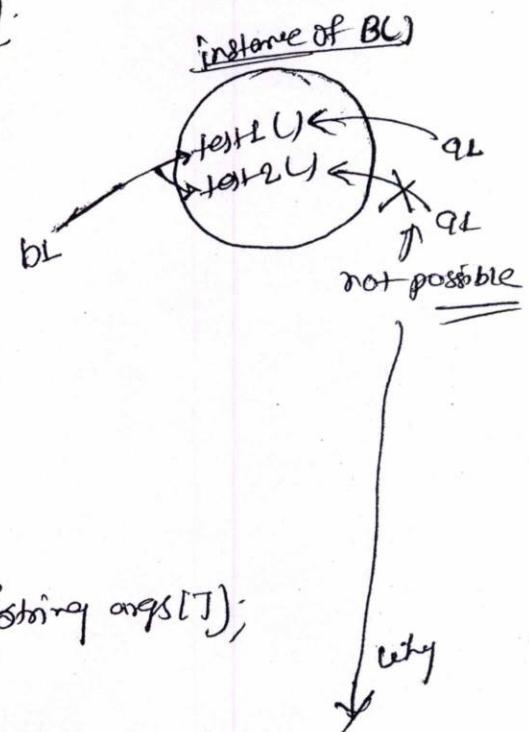
O/P program starts

test1() in class A

test2() in class B

test1() in class A

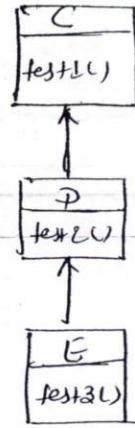
program ends.



* method can
be called based
on only based on
its reference
variable in form

class C

```
{ void test1() {  
    System.out.println("test1() of class C");  
}  
}
```

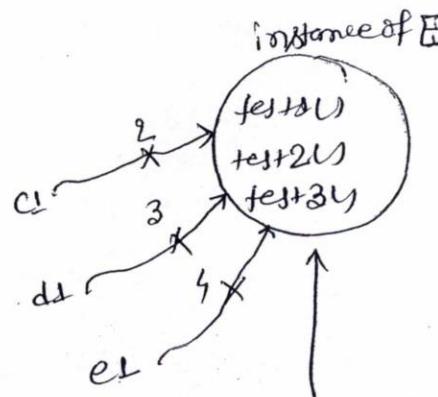


class D extends C

```
{ void test2()  
{  
    System.out.println("test2() of class D");  
}  
}
```

class E extends D

```
{ void test3()  
{  
    System.out.println("test3() of class E");  
}  
}
```



Here one
instance of E
showing different
behaviors through
different reference
variable
Example of
polymorphism

Class Pgm2

```
{ public static void main(String[] args)  
{  
    System.out.println("Program Starts");  
}
```

C cl = new E();

cl.test1(); } }

Here only test1()
can be called through cl

System.out.println("-----");

D dd = (D) cl;

dd=null;

dd.test1(); } }

dd.test2(); } }

Here only two
method can be called through
dd

System.out.println("-----");

E el = (E) dd;

el=null;

el.test1(); } }

el.test2(); } }

el.test3(); } }

Here all method
can be called through
el.

System.out.println("Program ends");

O/P

pgm starts

test1() of G

test1() of G

test2() of G

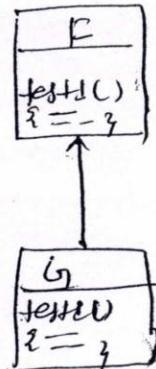
test1() of G

test2() of G

test3() of E

pgm ends

V.V.I



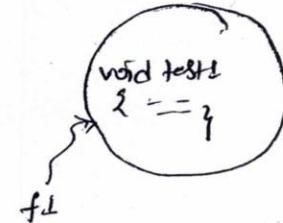
class F

```
{
    void test1()
    {
        System.out.println("test1() of F");
    }
}
```

instance of E

class G extends F

```
{
    void test1()
    {
        System.out.println("test1() of G");
    }
}
```



class Run5

```
{
    public static void main(String[] args)
    {
        System.out.println("pgm Starts");
    }
}
```

f1 = new G();

f1.test1();

} System.out.println("pgm ends");

g is also called
late binding
or dynamic
binding or
run time
polymorphism

76

at compile, java checks
but in class F, void test1() is
present or not, means java
checks method name & signature.
java does not implementation
code. So, then, at run time
implementation of G will be
checked.

O/P

```
{
    pgm Starts
    test1 of (G)*
    pgm ends.
```

interface Animal

```
{ void move();  
void makesound();  
void eat();  
}
```

class Dog implements Animal

```
{ public void move()  
{ S.O.P("dog is walking");  
}  
  
public void makesound()  
{ S.O.P("Bow Bow");  
}  
  
public void eat()  
{ S.O.P("dog is eating");  
}
```

Animal
move();
makesound();
eat();

Dog
move()
makesound()
eat()

Cat
move()
makesound()
eat()

class Cat implements Animal

```
{ public void move()  
{ S.O.P("cat is walking");  
}  
  
public void makesound()  
{ S.O.P("MEOW MEOW");  
}  
  
public void eat()  
{ S.O.P("cat is eating by  
closing eyes");  
}
```

Class Run 5

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

```
    System.out.println ("program starts");
```

```
A a1 = new Dog();  
* a1.move();  
a1.makesound();  
a1.eat();
```

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

```
A a2 = new Cat();  
a2.move();  
a2.makesound();  
a2.eat();
```

```
System.out.println ("program ends");
```

```
}
```

Output

program starts

Dog is walking

Bow Bow

Dog is eating

Cat is walking

MIAOW MIAOW

Cat is eating by
closing eye

program ends

77

Note:
we can call static method
by reference variable, although
method call be decided at
run time, it will slow the
process. It is standard practice
to call static method through class
name. Execution will be faster.

Agree
4/05/13

BINDING

- Static Binding or early binding
- happen at compile time
- also called compile-time polymorphism.

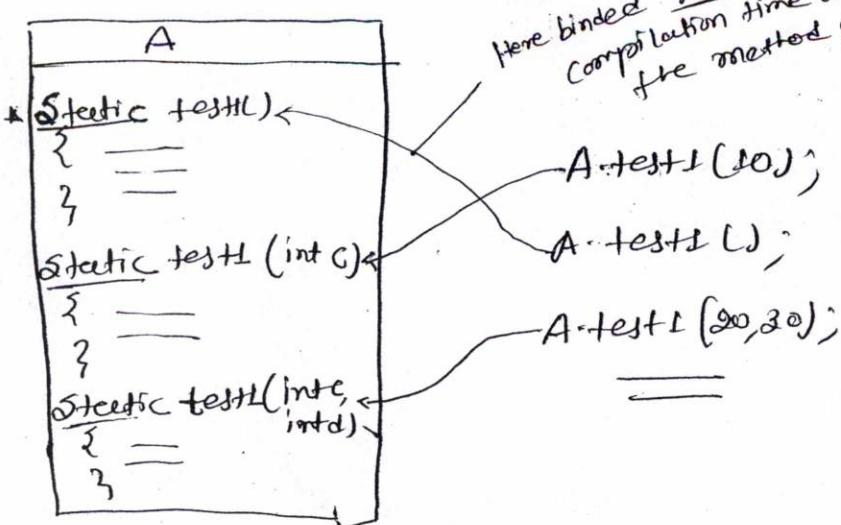
- Dynamic Binding or late binding
- happen at Runtime
- Java resolves which body will be executed for method, decide at run time or execution time
- also called run-time polymorphism.

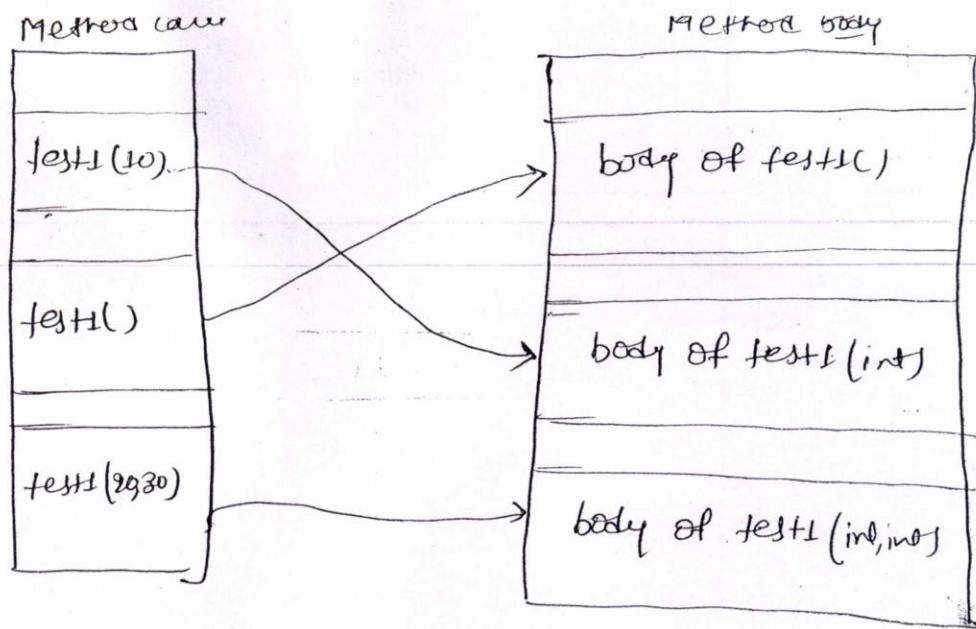
Method Binding

- There are two types of method binding.
 - i) Static binding or early binding.
 - ii) Dynamic " or Late binding.

Early Binding

- Here, Java decides which method body to execute for a particular method call at the compilation time.
- Java resolves the method call during compilation only. It is called early binding or static binding.
- All static methods calls are bound at compilation time.





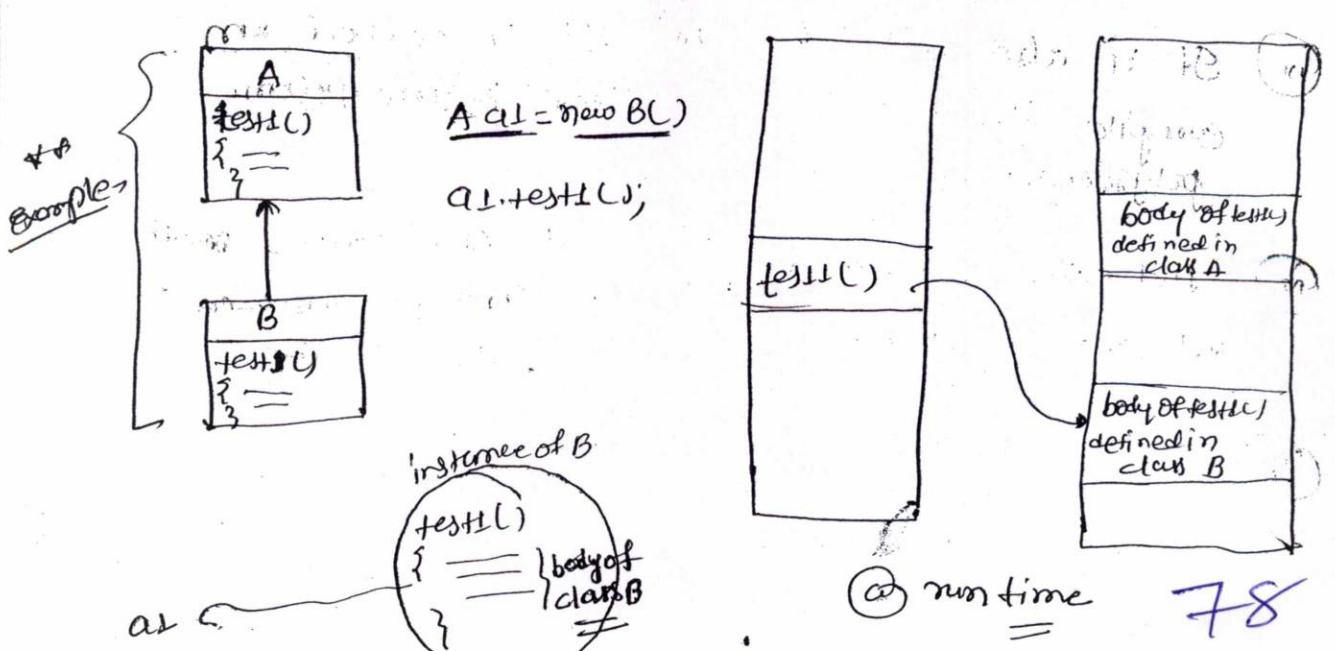
At compile time:-

* Dynamic or Late Binding

→ Here Java decides, which method body to execute for a particular method call at run-time.

→ Java resolves a method call during run time is known as late binding.

→ * All instance method call → late binding
 i.e. all non-static method call decide at run time.



* Late Binding are also called run-time polymorphism.

* Run-time polymorphism is the process in which java resolves call to overidden method during run-time.

* Create an instance of sub-class and upcast to super-class, when we call overidden method, method body will be executed based on actual instances created; not based on the reference variable.

* Java decide call to overridden method at run-time called run-time polymorphism.

Difference b/w static & late binding

(i) It is also early binding

(ii) Resolves ^{called} method body at compile time

(iii) It is also called compile time polymorphism

(iv) All static method call are early binding

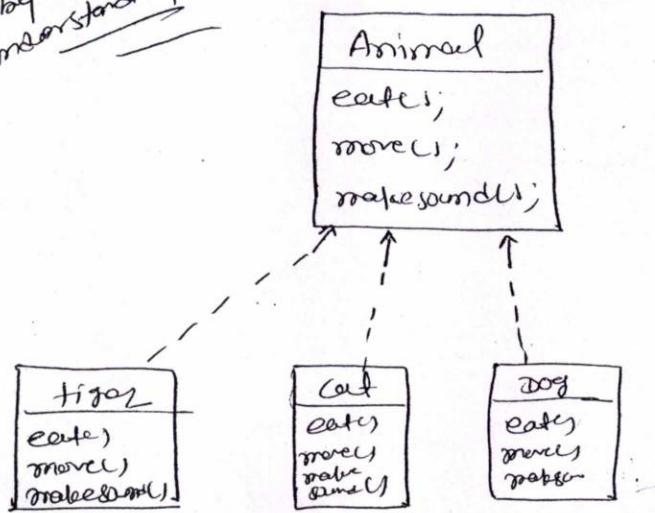
(i) It is also called dynamic or late binding.

(ii) resolve call to overridden method at run-time

(iii) It is called run time polymorphism.

(iv) All non-static method call through instances call late binding.

~~written by me
for understanding~~



if we want here, create 10 dog
object then we

```
Dog d1 = new Dog();  
d1.eats();  
d1.move();  
d1.makesound();
```

```
Dog d2 = new Dog();  
d2.eats();  
d2.move();  
d2.makesound();
```

```
Dog d3 = new Dog();  
=====
```

```
=====
```

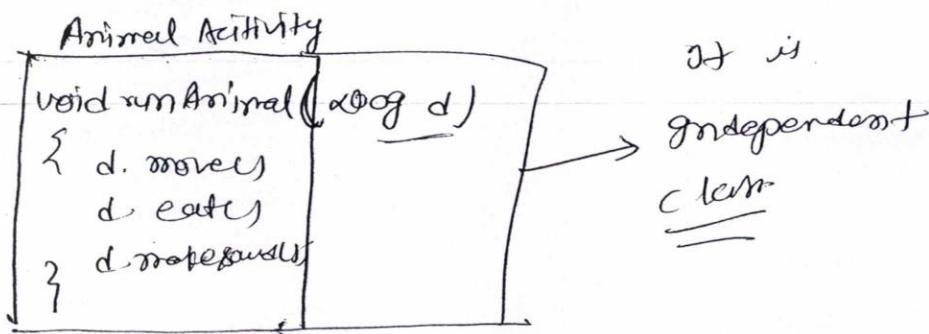
```
=====
```

```
Dog d10 = new Dog();  
=====
```

Here problem, if we want to create 10 dog or 10 cat then object then we have to separately create 10 object and call each method to dog object... there is some line or code is repeating. There is no reusability of code.

79

make a separate class or base class which
picks the object



Now Animal activity is ~~one~~ a new Animal Activity

ac. runAnimal (new dog) → It reduces code

or we are passing one, we are passing

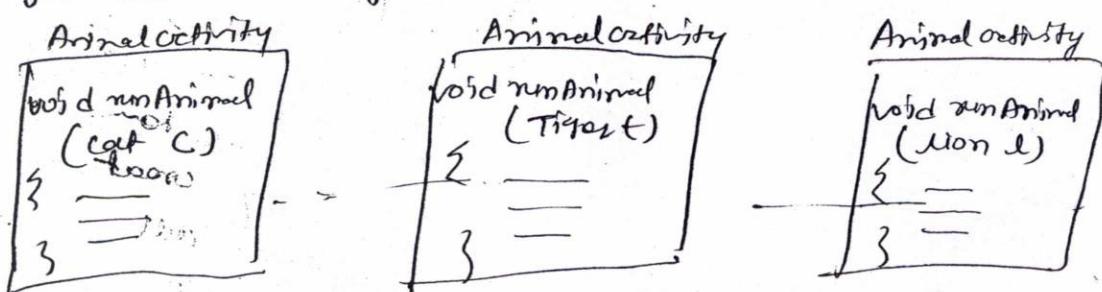
a dog object to that class, it will automatically call all the methods of dog,

here it is reusability of code

dog d = new dog();
ac. runAnimal (d);

On this method, argument is dog type,
so, we can not pass cat type argument
or lion — then we have create

for each a separate and independent class like



Again here, if we have 10 different type of animal, then we have to create 10 different independent classes so, call each object through separate class

So, instead of this — create a general

Animal Activity

```

void runAnimal(Animal a)
{
    a.move()
    a.makesound()
    a.eats()
}

```

Animal Activity ac = new Animal Activity

ac.runAnimal(new dog);

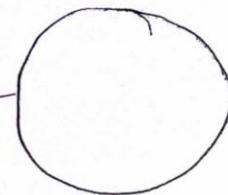
ac.runAnimal(new cat);

ac.runAnimal(new tiger);

—
—
—
—
—

Here instance
of dog is
created, so refer
variable passed to
runAnimal method; due
to late binding (move,
makesound, and eat)
method of dog will be
called
instance of dog called

a1



Now Here we have created a general class Animal Activity which has Animal arguments. It is independent class when we want a dog object, create an instance of dog and pass to Animal Activity runAnimal method.

due to late binding, first instance of dog is created, then when we pass dog instance to runAnimal method then it will call method related to dog object due to late binding. This is called run time polymorphism it provide lots of flexibility and less code to write in the program.

We can easily add any Animal like elephant ant, elephant, class in future, just passing that that animal to runAnimal method it will work fine.

80

~~interface~~ Animal

```
{  
    void move();  
    void makeSound();  
    void eat();  
}
```

class Dog implements Animal

```
{  
    public void move()  
    {  
        System.out.println("Dog is walking");  
    }  
}
```

```
remember  
remember  
public void makeSound()  
{  
    System.out.println("Bow Bow");  
}
```

```
remember  
public void eat()  
{  
    System.out.println("Dog is eating");  
}  
}
```

class Cat implements Animal

```
{  
    public void move()  
    {  
        System.out.println("Cat is walking");  
    }  
}
```

```
public void makeSound()  
{  
    System.out.println("Meow Meow");  
}
```

```
public void eat()  
{  
    System.out.println("Cat is eating");  
}
```

class AnimalActivity
 {
 Animal Activity

 void runAnimal(Animal a)
 {
 a.makesound();
 a.moves();
 a.eats();
 }
 }

class RunI

{ p.s.v.m(string[7] args)

{ S.O.P("Pgm Starts");

Animal Activity ac = new AnimalActivity();

ac.runAnimal(new dog());

S.O.P(" --- ");

ac.runAnimal(new cat());

S.O.P(" --- ");

ac.runAnimal(new dog());

S.O.P(" --- ");

ac.runAnimal(new cat());

S.O.P(" Pgm ends ");

} }
S.O.P

Pgm Starts.

Bow Bow

Dog is walking

Dog is eating

MEOW MEOW

Cat is walking

Cat is eating

Bow Bow

Dog is walking

Dog is eating

MEOW MEOW

Cat is walking

Cat is eating

JEP is also an
example of run time
polymorphism.

81

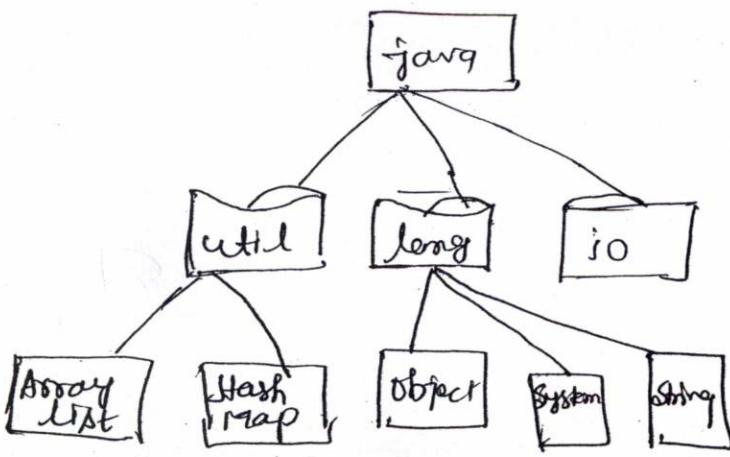
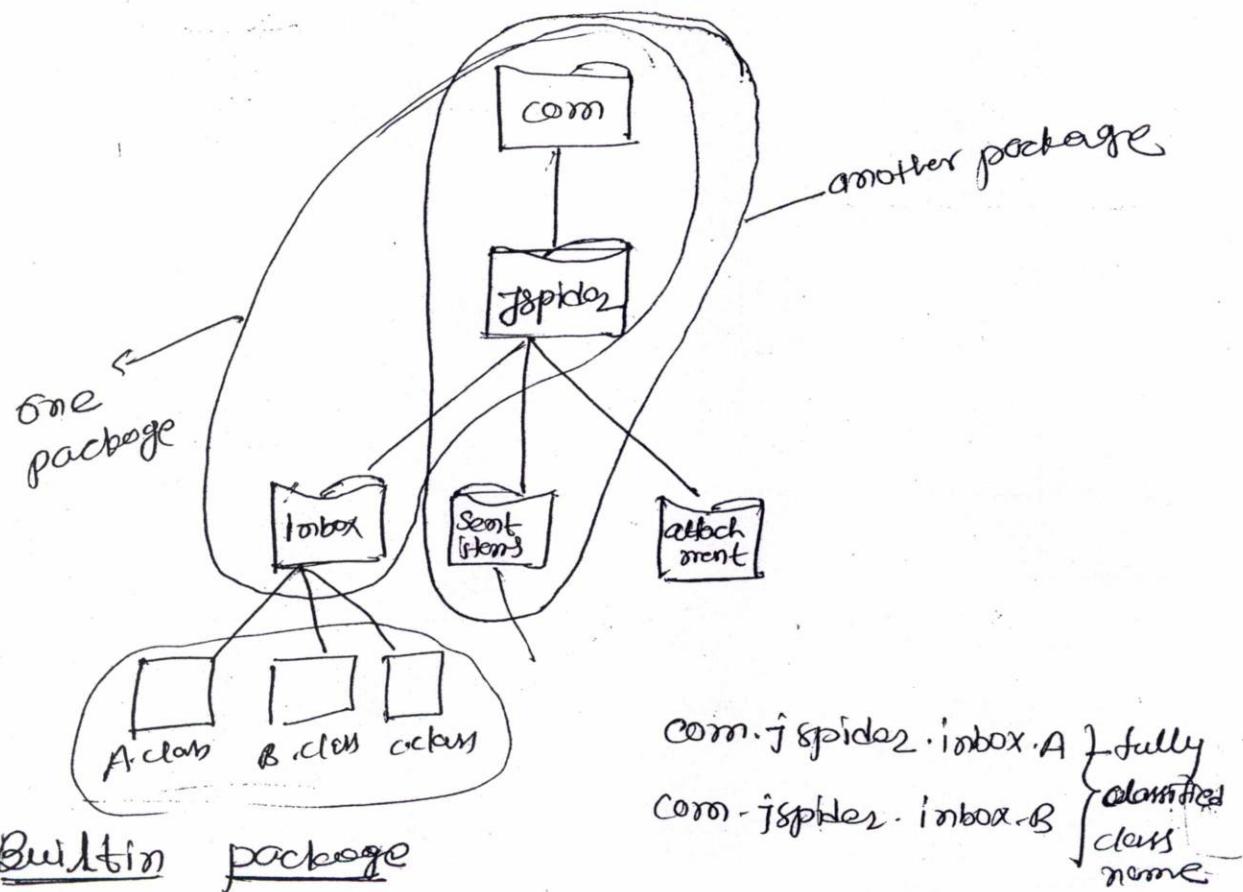
~~date
Monday
6/05/2013~~

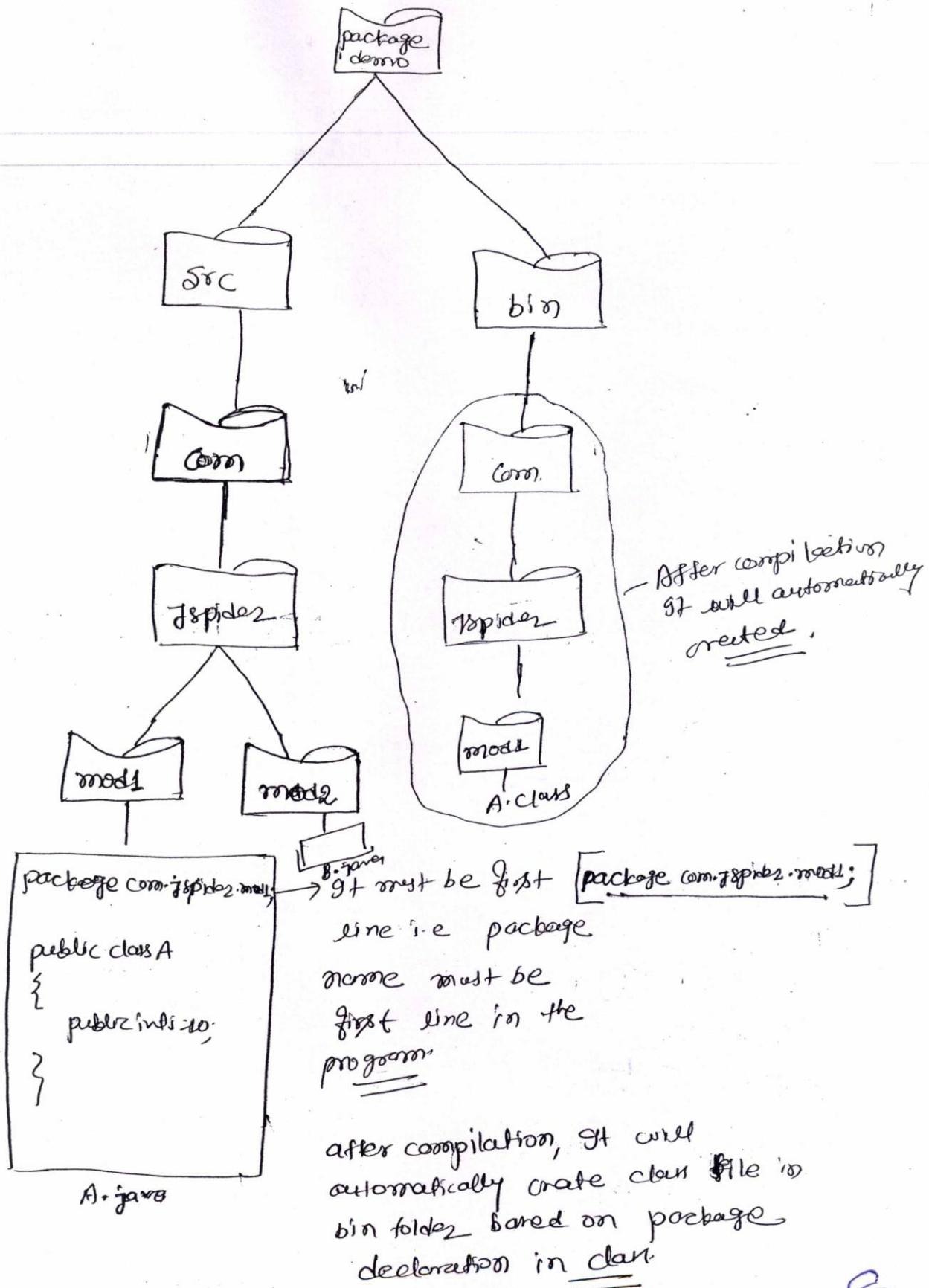
package:

- * package is a collection of classes. packages are used to avoid naming collisions.
- * Every class that we create should present inside a package.
- * Generally in projects, we start package name with the reverse order of domain name.

ex:-

www.jspider.com





Compilation

navigate to folder which contains the source files

ex- cd ~~src~~/com/jspider/mod1 > javac -d D:\com\b1\packageDemo\bin

javac D:\com\b1\packageDemo\bin\A.java.

A.java ←

During compilation given only target folder path, Java creates a package based on given class

82

To execute

D:\UOMS13\packagedemo\bin:>

java com.jspider.mod1.A ↴

for execution, go the target folder and
then give fully qualified class name that
com.jspider.mod1.A ↴

Way to set class path

→ To access a class present in different package, we have to set package path in a variable called classpath.

* set the path in environment variable of
class path new variable

• • •

→ To use a class which is present in different package, either use fully qualified class name or import a class using import keyword.

```
package com.jspider.mod2;  
import com.jspider.mod1.A;
```

class B

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

```
{ System.out.println("Program Starts"); }
```

at home
(com.jspider.mod1)
if we don't import
package then specify
package in class

```
A a1 = new A();
```

```
System.out.println(a1.i);  
S.O.P ("Program ends");
```

if we don't write
import package

- package statement should be the first line in the java file.
- only one package statement is allowed in the file.
- write import statement after package declaration.
- multiple import statements are allowed in java.
- To import all the classes present in the package, use "*".
 ex- import com.jspdo.model.*;
- Standard practice, write package name in lowerCase.
- In a java file, we write, only package declaration and import statement, outside the class.

* Access specifiers

- There are four levels of access.

(1) public access — public keyword

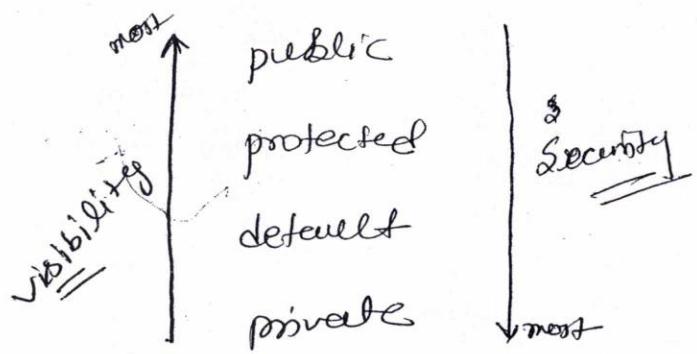
(2) protected " — protected

(3) package "/default" — no keyword
access/friendly access

(4) private access — private

- private members can not be accessed outside the class.
- default members can be accessed in all classes belongs to some package. 83
- default members can not be accessed outside the package.
- protected members are accessed to all the classes belongs to some package & also in the classes of different package through Inheritance.
- public members can be accessed in any class (in)

- for a simple class, only `gt` will be default or `public`.
- for a inner class (class inside class), `gt` can be used all access specifier.



Inside class	outside class within package	outside the package
public	Yes	Yes
protected	Yes	Yes (through inheritance)
default	Yes	No
private	Yes	No

Date
7/05/13

I^DE - Integrated Development Environment

- used for web applications

Netbeans - Client server application mostly used

versions of eclipse

Juno - current version

Indigo

Helios

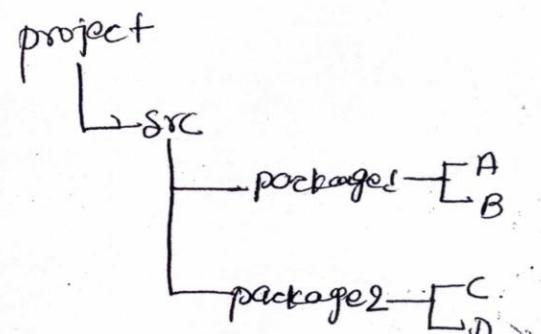
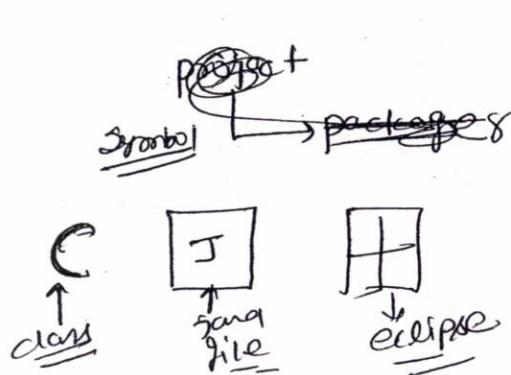
Galileo

Ganymede

Europa

- Download Eclipse rar file & extract (no installation is required for eclipse)
- when we open eclipse, select the work-space, work-space is the folder, which is used to store all java program written through eclipse.

How to create project



Step 1 → create a project

(i) file → new → project

(ii) java project → project name → finish.
creating a package:-

84

(i) in package explorer, right click on src → new package → package name → finish

(ii) under package → right click on package → new → class → class name → select the options do you want → finish

How to change the font

Windows → preferences → General → Appearance

→ colors & fonts → expand java on right side →

java editor text font → edit → change size and
etalic or bold → Finish.

→ Control spacebar → used to complete the statement

→ Eclipse is auto-completed. As soon as we write the code, it is auto completed by eclipse, and give the error or --

→  → red color — indicate static member

→  ^{any hollow triangle} — indicate variable symbol

→  — solid symbol — method

→  — green solid circle — public method (non-static)

→  — green hollow circle — public variable (non-static)

→  — blue color triangle — default access specifier

→  — yellow color rhombus — protected

* Encapsulation *

- * Encapsulation is a process of creating private variable & providing an access through public methods.
- * A public method, which is used to set private variable, is known as setter. And a public method which is used to get the private variable value is known as getter.

package com.jspiders.encapsulation;

```
public class Dog
{
    private int size;
    private String name;
}

public void setSize(int s)
{
    size = s;
}

public void getName()
{
    return name;
}

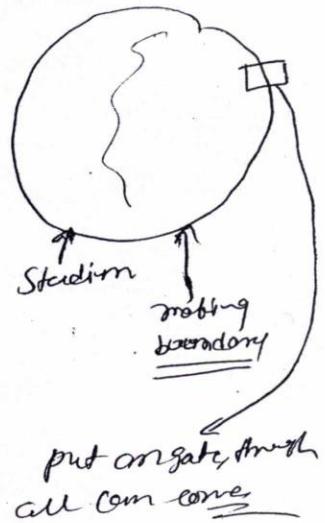
public void setName(String n)
{
    name = n;
}

public String getname()
{
    return name;
}
```

void ~~class~~ Run

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

General example



85

```
S.O.P("Poon starts");
```

```
dog d = new dog();
```

```
d.setSize(10);
```

```
d.getSize();
```

```
d.setName("Bulldog");
```

```
S.O.P("Name is " + d.getName());
```

```
S.O.P("Size is " + d.getSize());
```

}

~~If~~ If we want to access private variable outside
of class, then, through public method we can
use constructor, outside the class

```
public class A
```

{

```
private A()
```

{

Add static here

```
public static A getInstance()
```

{

return

```
A a1 = new A();
```

{

either write

```
return a1;
```

return new A();

}

After making static
variable

```
A a1 = A.getInstance();
```

—

- we can make constructor private
- but we can make instance of class through
make public method in that class
- If for the method, if we don't put static, then
method is non static, so, to call in other class, class A
method, we have to make instance of class, so, it
is not possible, so make method static then

package com.jspider.encapsulation;

class A

```
{  
    int i=10; }————— non-static  
    {  
        private A();  
        {  
            S.O.P("A's const");  
        }  
    }
```

public static A getInstance()

```
{  
    S.O.P("executing getInstance");  
    return new A();  
}
```

public class Run2

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

```
{  
    S.O.P("pgm starts");  
    A a1 = A.getInstance();
```

```
S.O.P(a1.i);  
S.O.P("pgm ends");
```

O/P
pgm starts

executing getInstance)

A's const

10

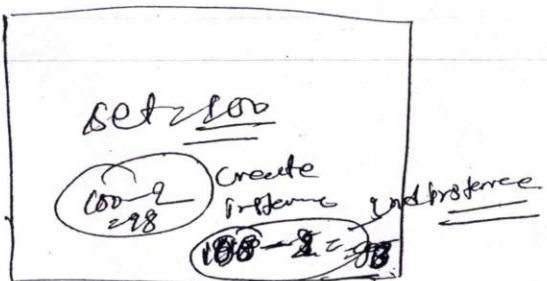
pgm ends

86

- Can we make constructor private — ref.
- How can we create instance of class — through public static method
- In other class, we can create an instance of that class — ref.

Singleton design pattern:

* singleton class



↑
take example of
flight book ticket

each time, a user enter to book
a flight ticket, then each time it separate
instance will create, each time available set
100 will show. and book the ticket, so, there
is huge problem so, avoid the problem, make
a singleton class - i.e everyone should only
one instance. Generally, we want use a
only single instance by all class. Singleton
class is used for generally shared resource.

Singleton class

- A class which has only one instance throughout the entire program execution
- To create singleton class, make constructor private & create a public static method which returns reference of an instance, if instance is already created, otherwise create an instance and return the reference.

for example

```
class Dog
{
    private static Dog d = null;
    private Dog()
    {
    }

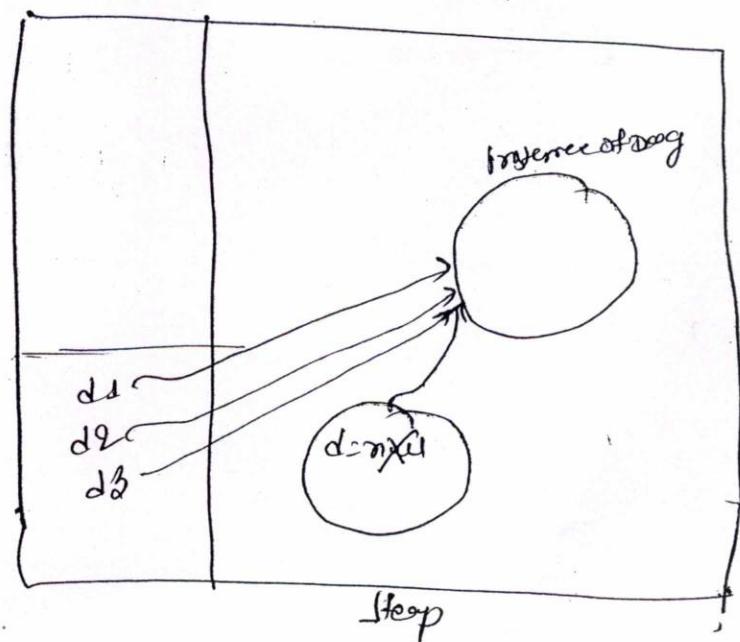
    public static Dog getInstance()
    {
        if (d == null)
        {
            d = new Dog();
        }
        return d;
    }
}
```

Class Run2

2 —

```
Dog d1 = Dog.getInstance();
Dog d2 = Dog.getInstance();
Dog d3 = Dog.getInstance();
```

3 —



87

program by us

Du
8/17

package com.ignited.singleton;

class Dog{

```
private static Dog d = null;  
static count = 0;  
private Dog()  
{  
    count++;  
}
```

public static Dog getInstances()

```
{  
    if(d == null)  
    {  
        d = new Dog();  
    }  
    return d;  
}
```

class Run2

```
{  
    public static void main(String args[]){  
        System.out.println("Count = " + Dog.count);  
        Dog d1 = Dog.getInstances();  
        Dog d2 = Dog.getInstances();  
        Dog d3 = Dog.getInstances();  
        System.out.println("Count = " + Dog.count);  
    }  
}
```

Output

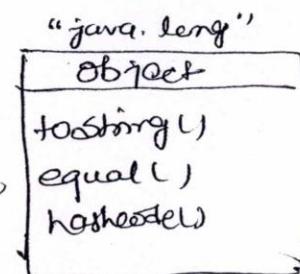
Count=1;

Count=1;

Date
8/05/13

Object class:-

- * Object class is the super-most class in the java, which is available in "java.lang" package.
- * "java.lang" package is the default package, that means all the classes present in java.lang or implicitly imported or automatically.
- * A class that we create in java, is directly or indirectly inherited object class members.
- * Following are the important methods available in Object class:
 - toString()
 - equals()
 - hashCode()



- * Object class is concrete class. (if not, then we have to implement it in each class and put abstract.)

toString() method

- toString() is a public method, which returns "String representation of an current instance".

Signature of toString()

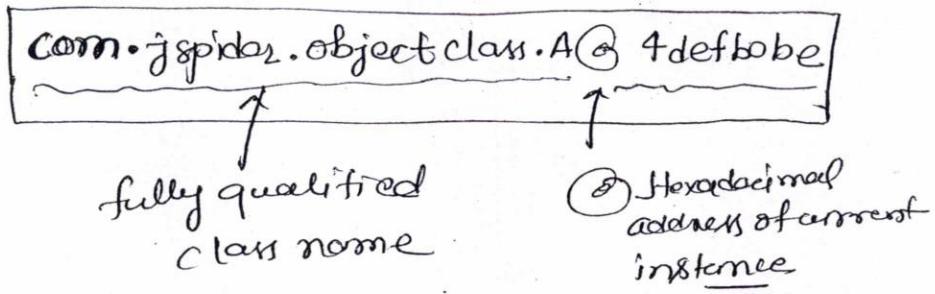
```
public String toString()  
{  
    —————  
    —————  
    —————
```

88

- * All the methods present in Object class,

- what string representation contains?
- string representation of an instance
contains a fully qualified class and @Hexadecimal address.

String representation of instance



```
package com.jspider.objectclass;
class A
```

```
{
```

```
public class Run {
```

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

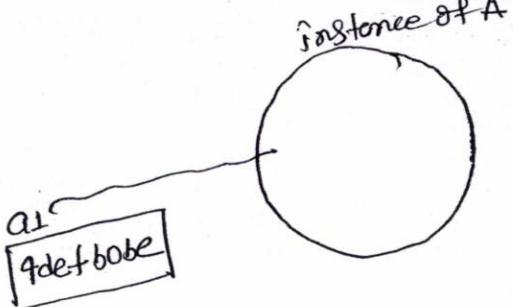
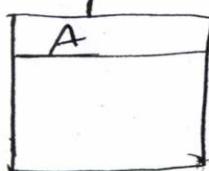
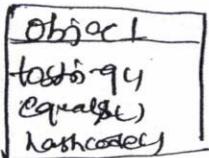
```
{ A a1 = new A();
```

```
    S.O.P(a1.toString());
```

```
}
```

```
O/P
```

com.jspider.objectclassA@4defbobe



package com.jspiders.objects;

Class A

```

    {
        {
            public class Run1 {
    
```

P.S. m.v (String[1cm])

{

A a1 = new A();

// A.tostring();

// S.O.P (a1.tostring());

S.O.P (a1);

}

O/P

com.jspiders.object.A@7096aa32

Instead of tostring() method pointing, we are pointing only reference variable (a1 → that contain only address of current instance (i.e. a1 above), but can give O/P, i.e. still fully classified class @ address of instance
So, Here what is happening.

that when we call or point reference variable, give implicitly or by default calling tostring() method, such that it is giving O/P fully classified @address, and tostring() always returns "String representation of instance".

* When we point, reference variable, Java implicitly calls or invokes tostring() method. So, when we point reference variable, we will get "String representation of an instance".

* { S.O.P (a1.tostring()); // explicit call to tostring() method
S.O.P (a1); // implicit call to tostring() method by java

interview question

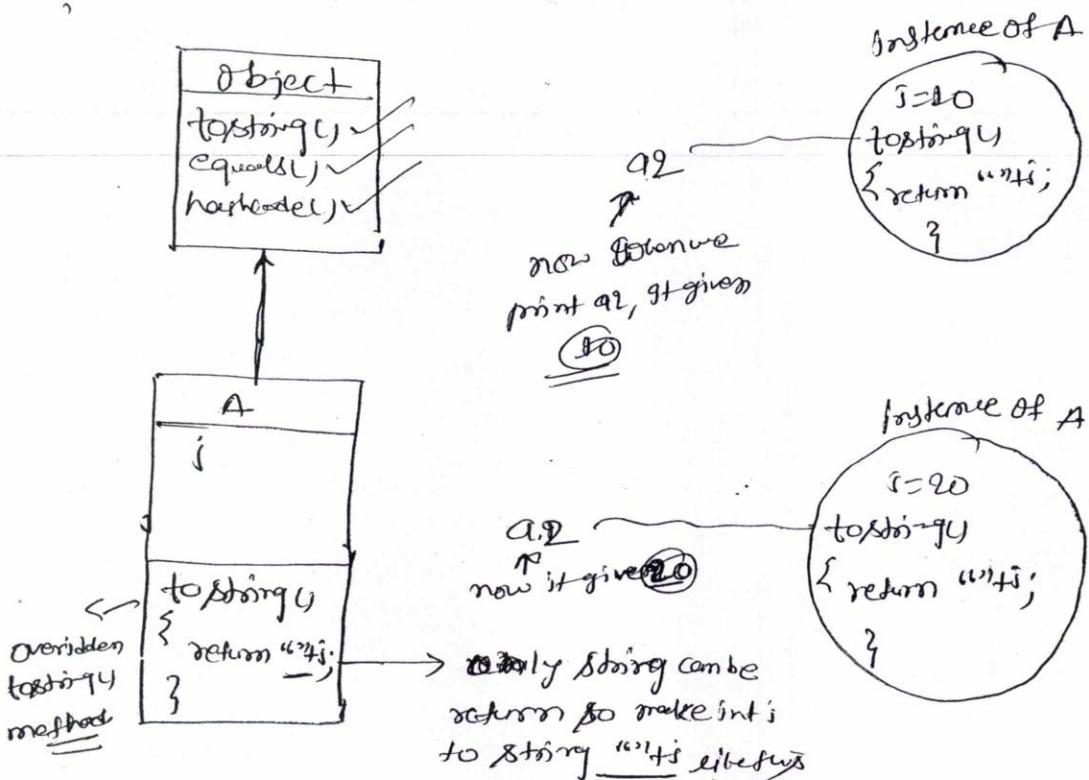
* How do you print member value through reference variable

Ans. through overriding the tostring() method

* Sometimes, we print reference variable (a1), we do not want "String representation", instead of we want member value.

* To print member values when reference variable

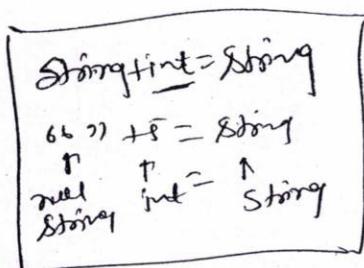
89



package com.jspider.Objectclass;

class A

```
{
    int i;
    A(int i)
    {
        this.i = i;
    }
}
```



public String toString()

{ S.O.P("Executing toString() of class A (i.e. overridden method here);") ; }

{ return " "+i; } → return type is String, & make int as String type conversion

public class RunI {

 P.S. main(String[] args)

 { A a1 = new A(10);

 S.O.P(a1);

 A a2 = new A(20);

 S.O.P(a2); }

O/P

Executing toString() of class A

(10)

Executing toString() of class A

(20)

O/P

& core

package com.jspidol.objectclass

class B

```
{  
    int i;  
    int j;  
    B(int i, int j)  
    {  
        this.i = i;  
        this.j = j;  
    }
```

public String toString()

```
{  
    {  
        return i + "-" + j; or return "i=" + i + "-" + j + "=" + j;  
    }  
}
```

}

public class Run2

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

```
{  
    B b1 = new B(10, 20); →
```

```
B b2 = new B(100, 200);
```

```
B b3 = new B(15, 40);
```

```
S.O.P(b1); } → Here overridden method toString()  
S.O.P(b2); } will be called and in overridden  
S.O.P(b3); } method, what we want we can do  
like we want to print i value not to  
return string representation  
}
```

Output

```
b1 = i=10 - j=20  
b2 = i=100 - j=200  
b3 = i=15 - j=40
```

toString(), will automatically
invoked only when we use
print statement,
but not at other
statement

90

* Can we create instance of object class math

QUESTION

Create employee class with name & salary field; override the toString method to name of the employee.

* Equal() methods:-

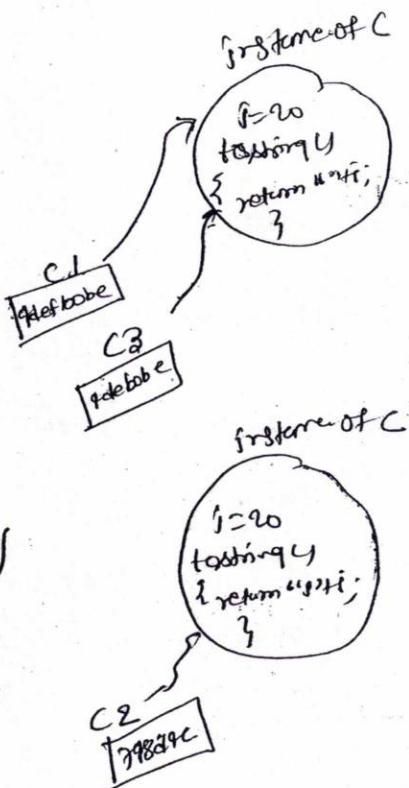
package com.jspiders.Objectclass
class C

```
{ int i;
  C(int s)
  {
    this.i = s;
  }
  public String toString()
  {
    return "i"+s;
  }
```

public class Run3{

```
  public static void main(String args)
  {
    System.out.println("C1=" + c1);
    C C1 = new C(20);
    C C2 = new C(20);
    System.out.println("C1=" + c1);
    System.out.println("C2=" + c2);
    System.out.println("C1==C2" + (c1 == c2)); // compare C1 & C2 address
    C C3 = c1;
    System.out.println("C1==C3" + (c1 == c3));
    System.out.println("end");
  }
```

SOP(a1) $a_1=10 \quad a_1=10$
SOP(a2) $a_2=10$
automatically
toString() is invoked
& print value
SOP(a1==a2) is true
here, toString() not
invoked, it will compare
address of a1 & a2 and
return false instead of
true



O/P program start
 C1=20
 C2=20
C1==C2 false
C1==C3 true

Condition
Opposite result
Condition
giving true

NOTE

- toString() will be implicitly invoked only when we are printing reference variable; (otherwise in other, we have to call explicitly).
- When we compare two reference variable using "==", address of instances will be compared

package com.jspdoz.Objectclass;

class D

{ int i;

 D(int i)

 { this.i = i;

 public boolean equals(Object x)

 {
 int a = this.i;
 int b = x.i;
 boolean res = (a == b);
 return res; or return (a == b);
 }

all code replaced by
return this.i == ((D)x.i);
we have
down casting
=

public class Run

{
 p. s. v. m (String[] args)
 {
 S.O.P("Program Starts");
 }

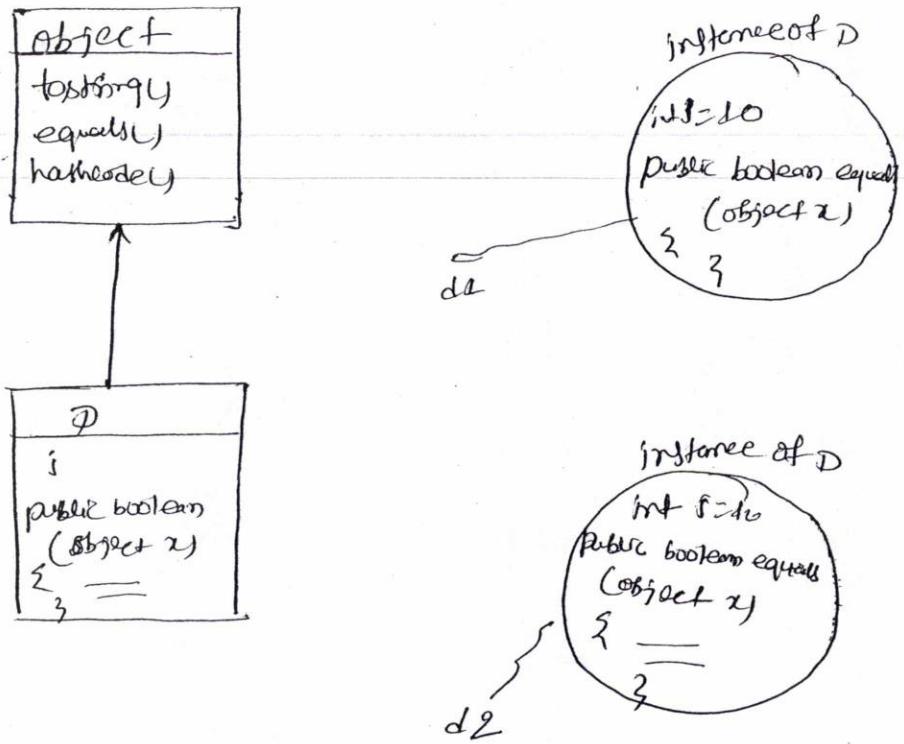
 D d1 = new D(10);

 D d2 = new D(10);

 {
 System.out.println(d1.equals(d2));
 }

int d = D x
d = D x
d =

91



Syndax of Equals() method

public boolean equals (object x)

```
{
    ———;
    ———;
    ———;
```

To compare two object
based on number value
not based on address
of object

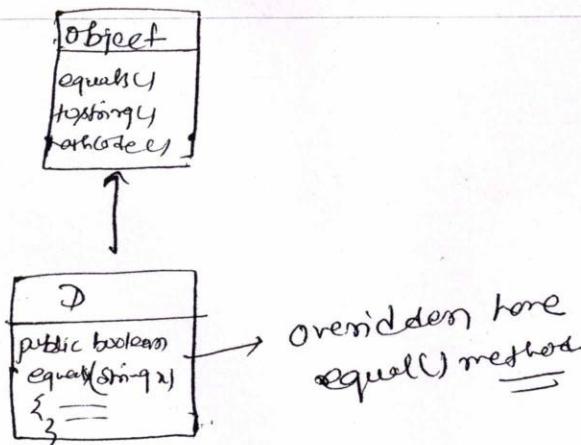
Equals() method

- equals() is a public method, which takes an argument of type Object.
equals() methods return boolean true, if the current instance and the given instance are same. otherwise
it return false

By default equals method compare two instances based on the address.

at $i_1 = 125$ $i_2 = 125$ at.equals (g2)
and also here because, it

→ to compare two instances based on the member value, override equals() method



public boolean equals(Object x)

{

 int a = this.i; → current instance's value
 D d = (D)x; → downcast + upcasted instances to D
 int b = d.i; → given instance's value.

 return (a == b);

current
instance's

given
instance's
=

→ in single line we can write

return this.i == ((D)x).i

instance of D

i = 10;
 public boolean equals
 (Object x)
 {
 =
 }

d2

instance of D

int i = 10
 public boolean
 equals(Object x)
 {
 =
 }

d2

92

```
class D  
{  
    int i;  
    D(int i)  
    {  
        this.i = i;  
    }  
}
```

```
public boolean equals(Object x)  
{
```

```
    return this.i == ((D)x).i;  
}
```

==

```
}
```

```
public  
class Run4
```

```
{  
    public void __()  
    {
```

```
        D d1 = new D(40);
```

```
        D d2 = new D(10);
```

```
        System.out.println("Comparing d1 & d2 using equals()");
```

```
        System.out.println(d1.equals(d2));
```

```
        System.out.println("Comparing d1 & d2 using ==");
```

```
        System.out.println(d1 == d2);
```

/

```
}
```

```
public  
class  
{  
    public  
    {  
        System.out.println("Comparing d1 & d2 using equals()");  
        System.out.println(d1.equals(d2));  
        System.out.println("Comparing d1 & d2 using ==");  
        System.out.println(d1 == d2);  
    }  
}
```

O/P Comparing d1 & d2 using equals()
true

Comparing d1 & d2 using ==
false

Assignment

Class Employee

```
{  
    String EmpName;  
    int salary;  
  
    Employee(String EmpName, int salary)  
    {  
        this.EmpName = EmpName;  
        this.salary = salary;  
    }  
  
    public String toString()  
    {  
        return "Salary = " + salary;  
    }  
    return "EmployeeName" + EmpName;  
}
```

public class Run1

```
{  
    public static void main(String args[]){  
        System.out.println("program starts");  
        Employee e1 = new Employee("Ravish", 10000);  
        Employee e2 = new Employee("Prakash", 20000);  
        System.out.println(e1);  
        System.out.println(e2);  
    }  
    System.out.println("ends");  
}
```

O/P program starts

EmployeeName = Ravish

EmployeeName = Prakash

ends.

93

Date
9/05/2012

package com.jspider.objectclass;

class Dog

{ int size

 Dog(int size)

 { this.size = size;
 }

 public String toString()

 { return "" + size;
 }

 public boolean equals(Object x)

 { return this.size == (Dog)x.size;

int a = Dog.size
int b = ((Dog)x).size
res = (a == b)
return res;

=

or

Dog d = (Dog)x;
return ~~size ==~~
~~this.size ==~~
~~d.size~~

or

class Run!

{ public static void main(String[] args)

{

 System.out.println("program starts");

 Dog d1 = new Dog(20);

 Dog d2 = new Dog(20);

 Dog d3 = new Dog(15);

 System.out.println(d1);

 System.out.println(d2);

 System.out.println(d2.equals(d3));

 System.out.println(d1.equals(d2));

 System.out.println(d1.equals(d3));

}

O/P

program starts

20
20
15
false

- as equal methods
- equals() method is a public method, that are available in Object class
- It takes an argument of object type and compare the current instances and given instances & return boolean value
- by default, It compare the two instances based on address of instance variable
- But If we want to equals() method, to compare the instances based on member value, then override the equals method in program

Structure public boolean equals (Object x)

- take the previous example to show

V.V.D
Question

If in a Employee class,
we have both variable
one is String name
another is Integer age

How can we compare both

Singly

If one consider
like age

One more
overridable
name
then how will
I write equals
method

Employee

Employee

{ =

{ employee

{ name

{ equals

equals

{
 books = manager.equals((Employee)x).age 94
 books = manager.equals((Employee)x).name
 ret .books

class E

{
 int i;
 int j;

E(int i, int j)

{
 this.i = i;
 this.j = j;

}

public String toString()

{
 return "" + i + "-" + j;
}

public boolean equals(Object x)

{
 E e = (E) x;
 boolean r1 = (this.i == e.i);
 boolean r2 = (this.j == e.j);
 return r1 & & r2;
}

*or write as one line
return (this.i == e.i)
&& (this.j == e.j)*

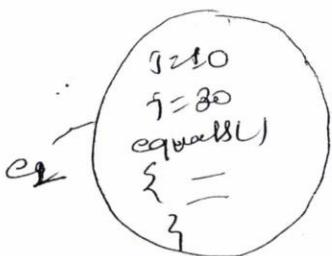
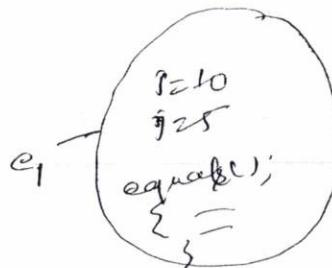
class Run1

{
 P.S.V.m(String[] args)

{
 E e1 = new E(10, 5);
 E e2 = new E(10, 30);
 S.O.P(e1.equals(e2));
}

}

Output false



```
class dog
{
    int size;
    dog(int size)
    {
        this.size = size;
    }
}
```

```
public boolean equals(Object x)
{
    dog d = (dog)x;
    return this.size == d.size;
}
```

```
Dog d1 = new Dog(5)
Dog d2 = new Dog(3)
```

if we call here d1.equals(c1); what will happen?
It will give run-time error, not compile time error. because we are trying to downcast for cat object into dog type

Dog d = (Dog)x
It will type of cat) downcast
It is type of dog (can not be happen)

It will give run-time error. To avoid this, after passing the objc to equals method, before downcasting, we have to check, the x is instance of Dog or not, so we can do it by "instanceof" method.

Instanceof

InstanceOf method

```

package com.jspider.objectclass

class F
{
}

class G
{
}

public class Run7 {
    public static void main (String[] args) {
        F f1 = new F();
        System.out.println ("f1 instances of F"); // true;

        Object x = new F();
        System.out.println ("x instances of F"); // true;

        Object x1 = new G();
        System.out.println ("x1 instances of F"); // false;

        if (x1 instanceof F)
            System.out.println ("true");
        else
            System.out.println ("false");
    }
}

```

Difference b/w (==) and equals()

- (==) checks equal, compare two instances based on address.
- equals() method, compare two instances based on the members value, but for that we have to override the equals() method.

```

class Dog
{
    int size;
    Dog(int size)
    {
        this.size = size;
    }
    public String toString()
    {
        return " "+size;
    }
}

```

```

public boolean equals(Object x)
{
    if(x instanceof Dog)
    {
        Dog d = (Dog)x;
        return this.size == d.size;
    }
    else
        else —— don't forget to write
    {
        System.out.println("Invalid composition");
    }
}

```

**

If we not use instanceof then if dog size=5 and cat size=5, and if we compare then it will give false. that is we don't want. Instead of that we want to print that this is an Invalid composition, to do that by instance of rather

```

public class Run5
{
    public void main(String[] args)
    {
    }
}

```

Dog d1 = new Dog();

Cat c1 = new Cat();

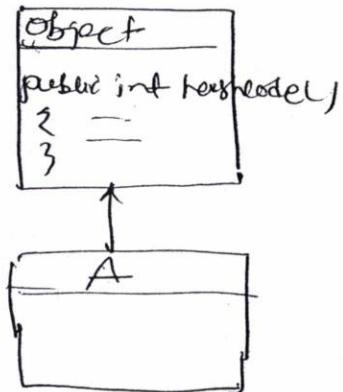
d1.equals(c1);

96

O/P Invalid composition

HashCode():-

hashCode() is a public method, which returns an integer number based on the address of the instance.



```
package com.jspider.objectcls;  
class H  
{  
}  
?  
public class P extends H  
{  
    public void main (String [ ] args)  
    {  
        H h1 = new H();  
        System.out.println (h1.hashCode());  
    }  
}
```

Output → 123456 → Some integers number,
based on address of instance.

+ If we print reference variable of any type indirectly
+ any calls the toString() method
and get object address.
but in String class, toString is overridden by Java developer.

String class :-

- String class available in "java.lang" pack
- Since, "java.lang" package is the default package so, no need to import String class.
- In String class, `toString()` is overridden to return string
- In String class `equals()` method is overridden two string based on member value or String content

package com.jspider.Stringclass;

```
public class Run{
```

```
p.s. v. m (String[] args)
```

```
{
```

```
String s1 = new String ("Java");
```

```
String s2 = new String ("java");
```

```
s.o.p ("s1=" + s1); // Java
```

```
s.o.p ("s2=" + s2); // java
```

```
s.o.p ("Comparing s1 & s2 using equals");
```

```
s.o.p (s1.equals(s2)); // true
```

```
}
```

```
3
```

Output: Java

java

Comparing s1 & s2 using equals

true;

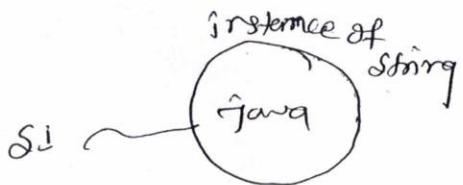
98

- String class is a final class.

String is a special class in java. we can create instance in two ways.

1st way (i) using new operator

String s1 = new String("java");



2nd way (ii) By Assigning String literals
String s1 = "developer";

String s2 = "developer"



both the ways, we can create instance

class Emp

```
{  
    String name;  
}  
Emp(String name);  
{  
    this.name=name;  
}
```

it is not primitive
type it is
reference variable

public boolean equals (Object x)

```
{  
    return this.name==((Emp)x).name; X  
}
```

got false not comparing names
it is still comparing
addresses of memory because
name is reference variable
of string class

80

class Emp

{
 String name;

 Emp (String name)

{
 this.name = name;
}

public boolean equals (Object x)

{
 Emp e = (Emp)x;

 String s1 = this.name;

 String s2 = ~~this.name~~;

 return s1.equals(s2);

}

QUESTION

* Whenever we compare, the primitive type, we can use ($==$) or equals method. But when we compare two instances, then always

use equal() methods.

if this.name == e.name, compare the address of name containing java, Here it is not compare string contents.

ANSWER

In String class, the equals () method is

Overridden to compare based on content so,

when s1 contain address & s2 contain address of

java. Then when call equal() method of s1 and

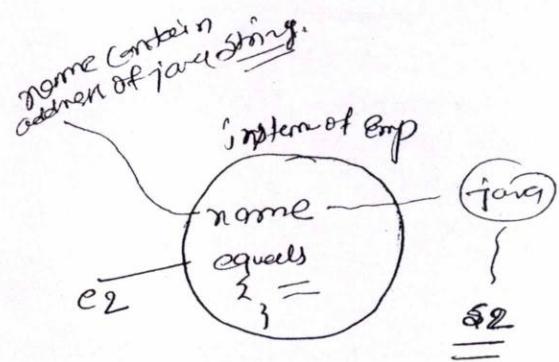
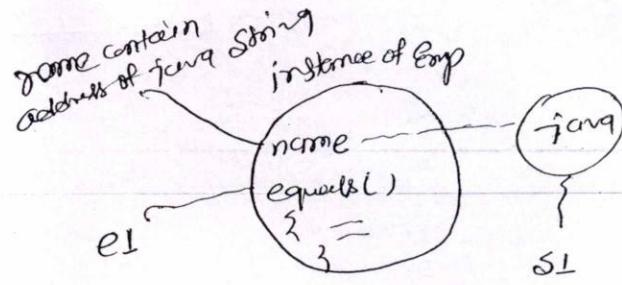
s2, means, we call equals() method of String

class s1 and s2, so then, we give comparison

result based on content. and give the

result,

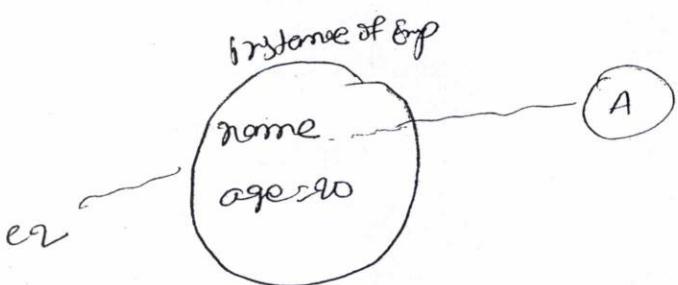
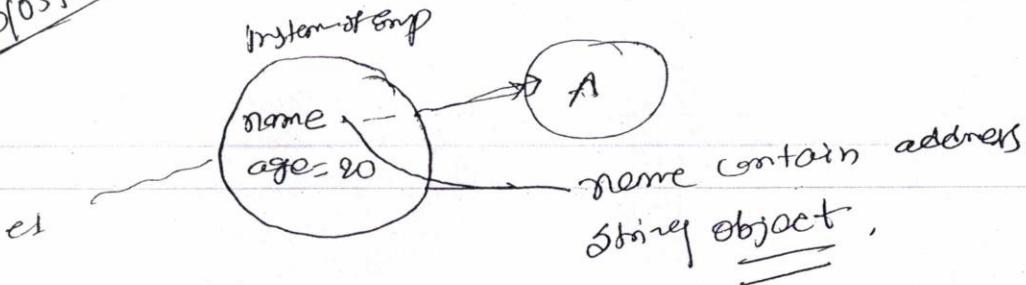
99



on single line

return this.name.equals(e.name)

Ques
10/05/2012



public boolean equals(Object x)

{
Emp e = (Emp)x;

boolean b1 = this.name.equals(e.name);

~~boolean b2 = this.age.equals()~~ X → we can not

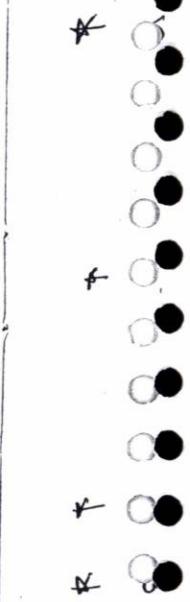
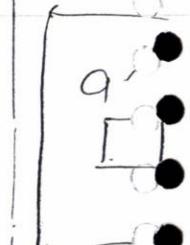
boolean b2 = (this.age == e.age); call equals
return b1 & b2; methods for primitive
 type because there is
 no equals() method
 in primitive type

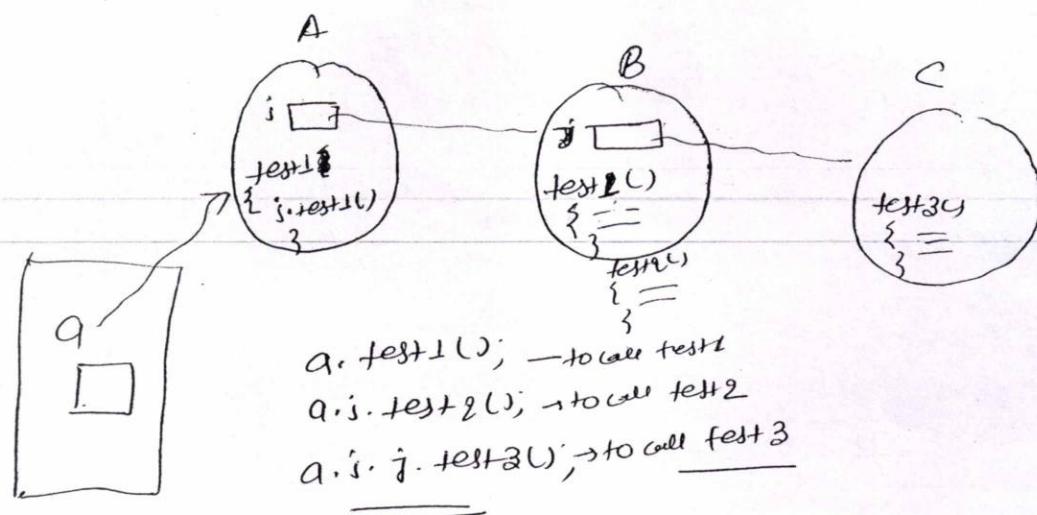
we can not call here

boolean b2 = ~~this.age.equals(e.age)~~ X

↑
we can not call equals methods for
primitive type, only for object equals()
method can be called. There is no
equals method present in integer
and age is not instance

but we can call equals()
methods for String because s1 is
an instance of String class and equals()
method present in String class





* String is an immutable class.

immutable — can not change

mutable — can change

- + Once you created an String, you can not alter the String content only new instances created for new String content
- + Duplicate instances is not allowed for String class
- * When we create a duplicate String instance, then it first checks where is existing instance, and put one more reference there

points of String

- String is a class that are available in java.lang package
- we can create instance in two ways in the String class
- ~~String~~ String is a immutable class, once we create instance we can not change the Content of that instance
- Equals() & hashCode() is overridden in String class, to return string & ~~boole~~ value based on comparison of content of string.

Immutable

String class is immutable that means once

String instances is created, value can not be altered

If I try to alter String reference variable, new instance will be created instead of altering the existing one.

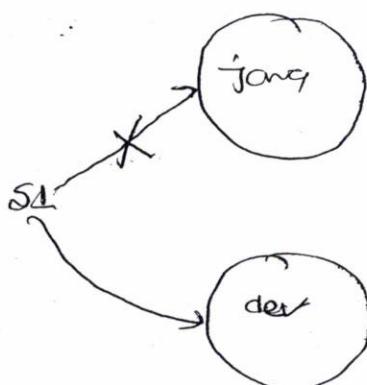
String s1 = "java"

s1 = "dev"

Here, in first line, s1 instance is created and java is loaded, then in 2nd line instead of assigning dev to java, it create one more instance

and put s1 reference to dev in other instance. and now

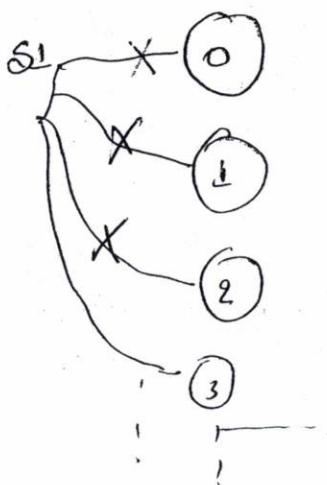
(java) is abandon object, after sometimes garbage collect clear the java object.



String s1

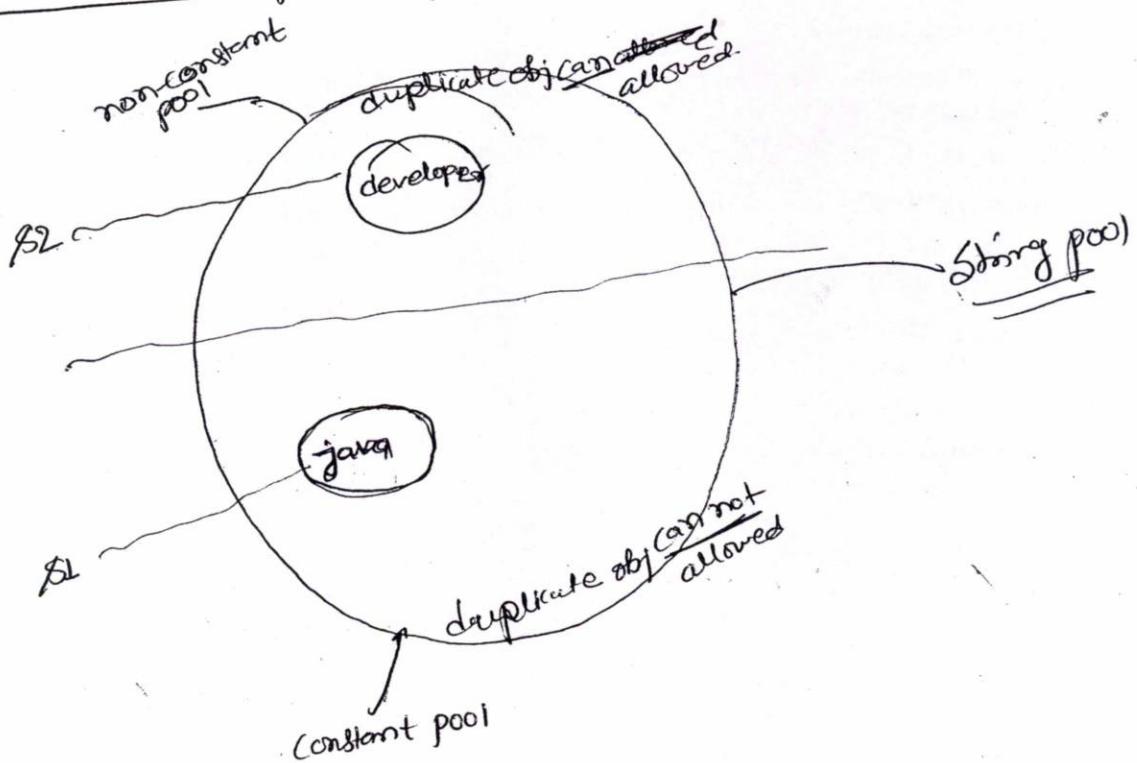
for(s=0; s<=10, i++)

{
 return "ts";
}



String pool

- All string instances are stored in string pool.
- String pool is divided into string constant pool and string non-constant pool.
- Duplicate instances are not allowed in string constant pool.
- String instances one created by assigning string literals are saved in string constant pool.
ex- String s1 = "java".
- String instances which are created using new operator are saved in string non-constant pool.
ex- String s2 = new String("developer");
- Duplicate instances are allowed in string non-constant pool.



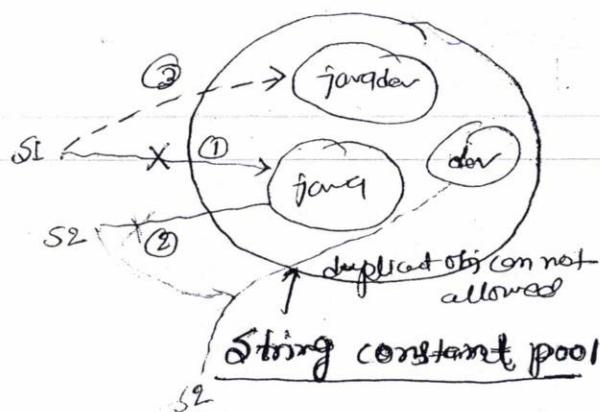
String constant pool

① String s1 = "java"

② String s2 = "java"

③ s1 = "javadoc"

Initially there is one string
 $s1 = s2$ gives true



String non-constant pool

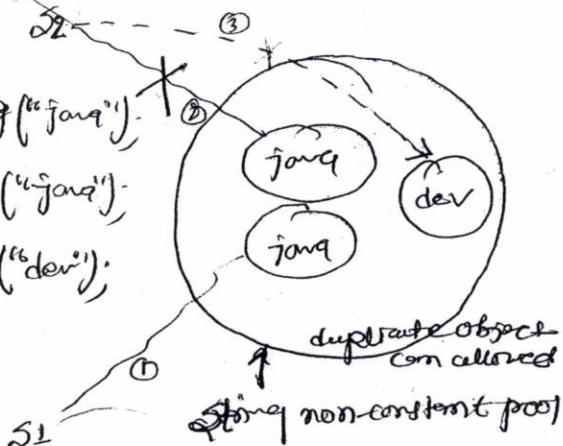
① String s1 = new String("java");

② String s2 = new String("java");

③ String s2 = new String("dev");

Here $s1 == s2$ gives

false here



→ Here if we created $s2 = \text{dev}$, then one more instance will created in constant pool and ~~java~~ dev in that instant and $s2$ will dereferenced from java in constant pool and point to dev instances

Methods available in String class

→ length() — length of the string

→ charAt(index) — return the char in a given index $\underline{\underline{\text{return}}} \underline{\underline{\text{index}}} \underline{\underline{\text{char}}}$

charAt(2) — return 3rd character $\rightarrow \underline{\underline{\text{return}}} \underline{\underline{\text{V}}}$

~~JAVA DEVELOPER~~

→ indexOf(char) — return of index of first occurrence of given character.

by default indexOf(char) return position of

Method available in String:-

JAVA DEVELOPER
 0 1 2 3 4 5 6 7 8 9 10 11 12

check for
 index of char
 St = developer
 S1 = javadeveloper
 S2 = "dev"
 S3 = "deo"

- * length() → return lengths of strings
 $S1.length = \underline{13}$

- * charAt(index) → return char in given index

$S1.charAt(2) \rightarrow$ return 3rd char = V

- * indexOf(char) → return index of given character

By default indexOf(char) returns position of first char

$S1.indexOf('a') \rightarrow 1$

$S1.indexOf('dev') \rightarrow 4$

 for 2nd occurrence, or 3rd? —

another method
 contains()
 indexOf(char, index)
 substring()
 split()
 replace()

$S1.indexOf('a', 2);$ → Start search a from 3rd character
 = first indicate, from where we start the search.

- * If character not exists, then it returns (-1)

```
package com.jspider.String;
public class Run2 {
    public static void main (String[] args)
```

String S1 = "javadeveloper";

S.O.P ("String S1=" + S1);

S.O.P ("String length=" + S1.length());

S.O.P ("char @ 2;" + S1.charAt(2));

S.O.P ("position of 'a';" + S1.indexOf('a'));

S.O.P ("position of dev;" + S1.indexOf("dev"));

S.O.P ("position of 'd' after 2;" + S1.indexOf('d', 2));

Output
 javadeveloper
 String length = 13
 Char @ 2: V

position of dev = 4
 position of ~~the~~ 'a' after 2; 3

102

If we want to search 2nd occurrence of e.
from above we can write as

st1.indexOf('e');

int p = st1.indexOf('e');

int q = st1.indexOf('e', p+1);

or int q = st1.indexOf('e', st1.indexOf('e')+1);

—

package com.jspiders.stringlab;

public class Run3 {

public static void main(String[] args)

{ String st1 = "javadevelopers";

int len = ~~st1.length();~~

for (int i=0; ~~i<len~~; i++)

{ System.out.println(st1.charAt(i));

}

}

Output

j
a
v
a
d
e
v
e
l
o
p
e
r

Assignment — write a program to count a particular character
in a string.

Assignment

```
package com.jspider.Stringclass
```

```
public class Run3
```

```
{
```

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

```
{
```

```
String s1 = "javadeveloper";
```

```
int len = s1.length();
```

```
int count = 0;
```

```
for (int i = 0; i < len; i++)
```

```
{
```

```
if (s1.charAt(i) == 'a')
```

```
{ count++;
```

```
}
```

```
else { System.out.println ("no character found"); }
```

```
System.out.println ("The character 'a' is present  
in string " + count + " times");
```

```
}
```

✓

```
package com.jspider.Stringclass
```

```
public class Run3
```

```
{
```

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

```
{
```

```
String s1 = "javadeveloper";
```

```
int count = 0;
```

```
for (int i = 0; i < s1.length(); i++)
```

```
{
```

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

```
If (charAt(i) == 'e')
```

```
{ count++;
```

```
}
```

```
}
```

```
if (count == 20)
```

```
{ System.out.println ("no character found"); }
```

```
else
```

103

sun.
11/05/2013

package com.jspider.Stringclass;
public class Rents

public static void main(String[] args)

}

String s1 = "java is very easy";

String s2 = "very";

int len = s1.length();

~~for (i=0; i<len; i++)~~

{ if (s1.indexOf(s2) != -1)

{ System.out.println("found"); }

else

{ System.out.println("not found"); }

}

~~if (s1.indexOf("very") != -1)~~

~~System.out.println("yes")~~

else

~~if (a1 != -1)~~

{ found = true }

else

~~{ not found }
}~~

→ contains() → gt will directly check, no need
to write if conditions

→ s1.contains(s2): — gt given directly returns
true and false

→ gt checks for, whether s2 string
is present in s1 string or not

* for reversing

public class Revb6

public static void main(String[] args)

{

String s1 = "javadevlepor";

for (int i = s1.length() - 1; i >= 0; i++)

{

System.out.print(s1.charAt(i));

}

}

b152

public class Revb6

{ public static ---

{

String s1 = "javadev";

String s2 = ~~"~~, "";

int len = s1.length();

for (int i = len - 1; i >= 0; i--)

{

s2 = s2 + s1.charAt(i);

}

S.O.P(s2);

}

Curly brace

replace() method

It ~~is~~ replace string or character in one string by other string or character

S.replace (find, replaceWith)

* public class Pm7

```
{ P.S. v. m
  {
    String s1 = "java";
    String s2 = s1.replace('a', 'o');
  }
  S.O.P(s2);
}
```

obj jovo



* public class Pm8

```
{ P.S. v. m
  {
    String s1 = "java"
    String s2 = s1.replace("a", "");
  }
  S.O.P(s2);
}
```

obj jav

It will replace a with null, means, a is deleted

Something we get the question, delete all 'a' or 'b' from String, do by upper method
replace by "".

~~good method to count particular character~~

~~public class Run7~~
{
 p.s.v.m}

String s1 = "java developer";

String s2 = s1.replace("e", "");

int count = s1.length() - s2.length();

}
S.O.P ("No of ~~e~~ 'e' is " + count);

~~D/P~~ = 3

* Count how many time string is exist

~~(✓)~~ String s1 = java + java + java;
=====

public class Run8

{
 public static void main(String[] args)

~~String s1 = "java java"~~

~~String s1 = "java+java=2java";~~

~~String s2 = s1.replace("java", "x");~~

S.O.P ("s2 = " + s2);

String s3 = s2.replace("x", "");

S.O.P (s2.length() - s3.length());

=====

~~D/P~~ = ~~(2) P~~

105

substring() method:-

public class Run9

{ public static -

String s1 = "javadeveloper" "javadevelopes"

String s2 = s1.substring(4); // developer

System.out.println(s2);

→ It will start from 4,
the first character
extract

String s3 = s1.substring(4, 11);

System.out.println(s3); // develop

if we write (4, 10) -
 It will extract 10-1 end,
 It is defined like that
 So, if we want to extract
 4 to 10, then always put
 end character 10
 i.e., (4, 10) → D

To mind Java developer like javadeveloper

↓

ja

jav

java

javad

javade

public class Run10

{ public static void main(String[] args)

{ String s1 = "javadeveloper";

int length = s1.length();

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

{ System.out.println(s1.substring(0, i));

javadeveloper

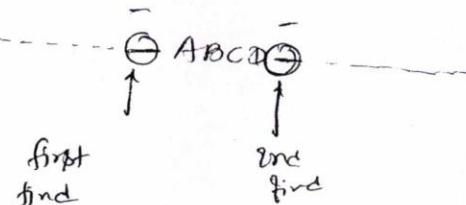
javadevelopers
ava developer
vadeveloper
adav -
= =

for (i=0; i<n; i++)
{
 System.out.println(s.substring(i, i+1));
}

for (i=0; i<len-1; i++)
{
 System.out.println(s.substring(i));
}

j
for
for
javadev

String s1 = "-----



substring()
substr()

to print string or character between $(-)$ —
then use substring method

class Run11

{
 P.S. V. M

String s1 = "javade-gaurav-kumar

```
int i = indexOf("-");  
int j = indexOf("-", i+1);  
System.out.println(s1.substring(i, j));
```

Output:
java

toLowerCase() — entire string in lower case.
toUpperCase() — entire string in upper case.

public class Run12

{
String s1 = "JavaDeveloper";

s.o.p(s1.toLowerCase());

s.o.p(s1.toUpperCase());

}

OP
javadeveloper
JAVADEVELOPER

split() — method later in Array

→ methods available in String class

- toString()
 - equals()
 - length()
 - indexOf(str)
 - charAt(i)
 - replace(old, new)
 - contains(char or str)
 - split()
 - substring(index, index)
 - toLowerCase(str)
 - toUpperCase(str)
- String[] arry1 = s1.split(",");
String[] arry2 = s1.split(" ");

Wrapper class

primitive type ————— corresponding wrapper class

byte	—	Byte
short	—	Short
int	—	Int
long	—	Long
float	—	Float
double	—	Double
boolean	—	Boolean
char	—	Char

- Wrapper classes are used to convert primitive into instances.
- All wrapper classes are available in "java.lang" package
- There is a particular wrapper class for each primitive type.

byte — Byte
int — Int

→ Boxing — (encapsulating a primitive ^{type} into in a instance)

* Converting a primitive into an object using respective wrapper classes is known as Boxing operation.

primitive ————— Boxing
 using wrapper classes → Object

* In all defined class, toString() method is overriden so, in All wrapper class also toString() method is also overridden.

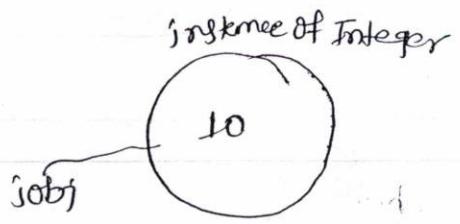
167

↳ Comm. if spider. boxing;

Public class Rem1

{ P. S. V. m (s -)

{ int i = 10



Integer iObj = new Integer(i); // Boxing

S.O.P(iObj);

int j = iObj.intValue(); // Unboxing

S.O.P("i=" + j);

Integer iObj = new Integer(i);

↳ Boxing operation

Integer iObj = i → AutoBoxing

UnBoxing

Converting an object into primitive
is called UnBoxing

int i = iObj.intValue();

↳ Unboxing

Both boxing & unboxing is automatic
in Java

Public class Rem2

{ P. S. V. m

{ Integer iObj = 10; // AutoBoxing ↗ Automatically calling new()

int i = iObj; // Autounboxing

S.O.P(i);

↳ ↗ it is automatically
call iObj.intValue() ~

```

public class Run3
{
    public static void main()
    {

```

```

        Integer iObj = 10; // Autoboxing
    
```

```

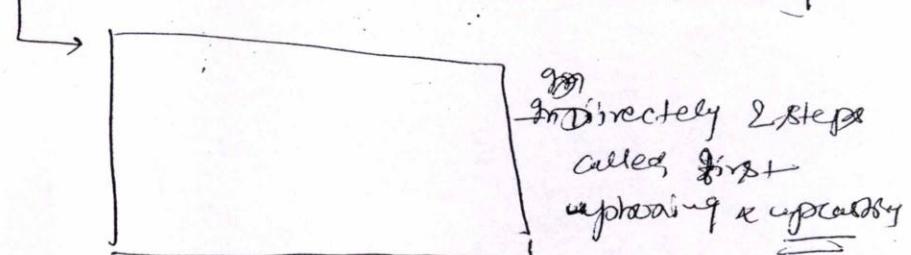
        int i = iObj; // Autounboxing
    
```

```

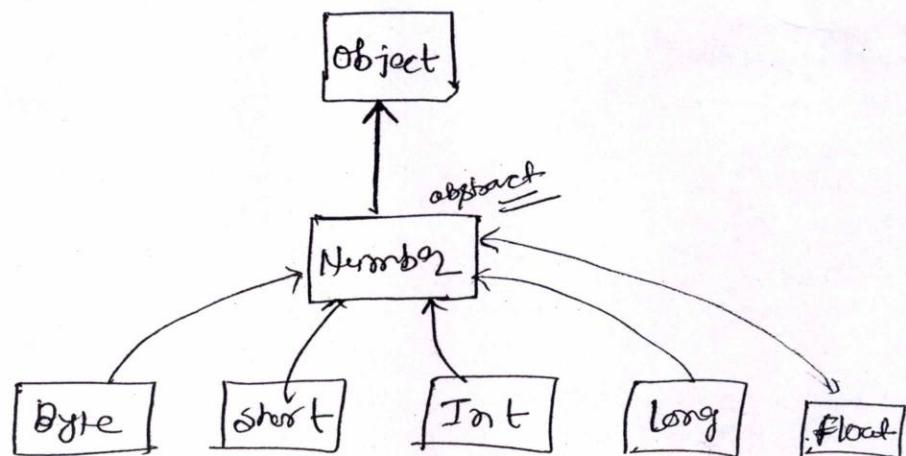
        System.out.println(i);
    }
}

```

} Object o = 10; // Auto upcasting and auto upcasting



Hierarchy



To read data from User we form cmd prompt

→ Scanner class present in "java.util" package
so, we need to import scansly.

→ Scanner.next() → It always read as String only

108

```
package com.jspider.boxing;  
import java.util.Scanner;  
public class Run3{
```

P. S. R. M

}

Scanner scn = new Scanner(System.in)

scn.p("enter your name")

↑
to read ~~the~~
input from
System

→ we

String name = scn.next();

read as

scn.p("Entered name=" + name);

String

scn.p("enter your age");

int age = scn.nextInt();

scn.p("age=" + age);

read as Int

}

}

O/P :

System.out.println(" ");

↑ ↑
Class static
reference
variable of
System class

non-static
overloaded method
of outputString.

out is reference variable of class OutputString.

Class A

```
{  
    static B bl;  
  
    static  
  
{  
    bl = new B();  
}  
}
```

→ we can call, non static method test

```
A.bl.test1(10);  
A.bl.test1("abc")  
A.bl.test1("double");
```

Class B

```
{  
    void test1(int c)  
}{  
    ==  
}
```

```
{  
    void test1(String s)  
}{  
    ==  
}
```

```
{  
    void test1(double d)  
}{  
    ==  
}
```

like some as System.out.println

class System → present in java.lang package

```
{  
    static OutputString out;  
  
    static  
  
{  
    out = new OutputString();  
}  
}
```

class OutputString

```
{  
    void println(int c)  
}{  
    ==  
}
```

```
{  
    void println(String c)  
}{  
    ==  
}
```

```
{  
    void println(double d)  
}{  
    ==  
}
```

→ so, we can call like that @

System.out.println("abc");

System.out.println(10);

109

Date
13/05/2012

Difference b/w String class & String Builder

String Builder class

- It is similar to string class. but string builder is mutable.
- All the method() present in string class, are also present in StringBuilder class. Here extra method is append() method
- In StringBuilder class, instances can be created using ~~new~~ new only, directly string can not be created
~~StringBuilder s1 = new StringBuilder();~~
~~X StringBuilder s1 = "";~~ X - can not write like
package com.jspid02.StringClass
{
 public static void _____
}
StringBuilder s1 = new StringBuilder();
s1.append("i");
s1.append("a");
System.out.println(s1);

StringBuffer() class

- It is same as StringBuilder but the methods are synchronized,
- Hence StringBuffer methods are thread safe ↴

Arrays:-

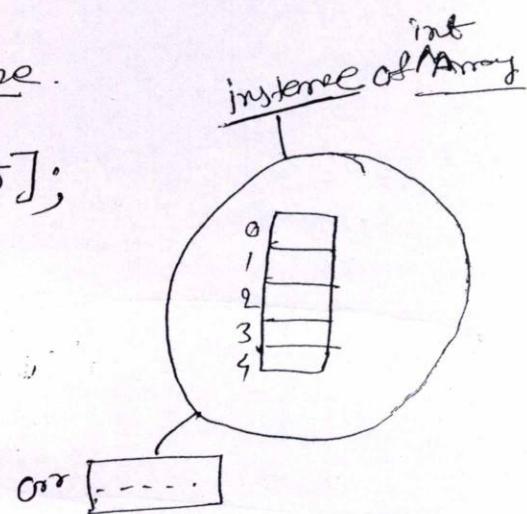
- Arrays is the group of similar data.
- In Java, Arrays is a class.

* Declaration of Array with size.

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

can write in
two line also

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



In General

<code>type[] arrayName;</code>	<u>declaration</u>
<code>arrayName = new type[size];</code>	<u>initialization</u>

* package com.jspiders.Arrays;

public class Demo{

public static void --

{

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

`arr[0]=10; arr[1]=20; arr[2]=30; arr[3]=40;
arr[4]=50;`

`s.o.p("array size=" + arr.length);` // Here length is variable

`s.o.p("arr =" + arr);`

It gives string representation
of array like → `[I@1ffbc08]`

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

{ `s.o.p(arr[i]);`

↳ length is variable, it
is not a method

It gives, how many
elements are present
i.e. size of arrays

110

System.out.println("arr = " + arr)

gives string representation

([I @ 4bcde08])
↑ ↑
Indicate arrays Integer type
Hexadecimal Address

For each or Advancee for looping:

Syntax variable name
for(int x: arrayName)
{
 S.O.P(x);
}

** S.O.P ("printing arr using for each loop")

for(int x: arr)
{
 S.O.P(x)
}

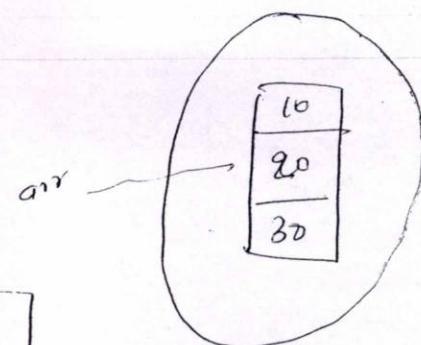
using advancee for, we can print
whole array element. we can
put any condition

O/P

Other way to initialize array

```
→ [int [] = arr;
   arr = { 10, 20, 30 };]
or in one line
```

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



Here Java automatically create instance and initialize the array.

* split() method :-

- ~~get~~ split() the string is based on some String
- split() method always returns array
- It is available in String class

package com.jspider.array;

```
public class Pm2{
```

```
    p. s. v. m{
```

String SL = "ab; cd; ef";

```
[String [] StrArr = SL.split (";");]
```

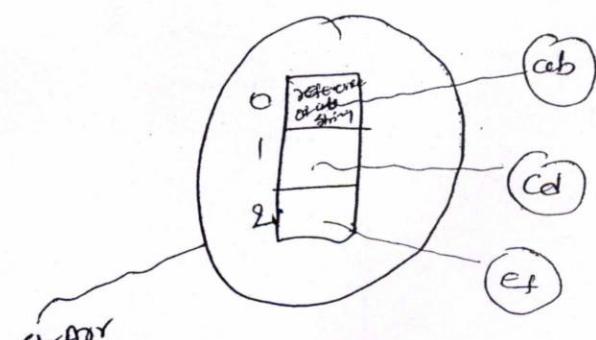
S.O.P (StrArr.length);

```
for( String x : StrArr)
```

```
{   S.O.P (x);
}
```

O/P

ab
cd
ef



111

→ split() method split the string are stored in and return array for we have to store

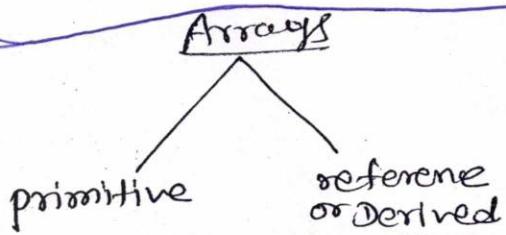
```

package com.jspidol.array;
public class Run3{
    p.s.v.m{
        double [] d = new double[4];
        d[0] = 20.34; // explicit widening.
        d[1] = (double) 30; // auto widening.
        d[2] = 40;
        d[3] = 50;
        for (double a : d)
        {
            s.o.p(a);
        }
    }
}

```

} → In this program we are showing that we can store different type of data in different type of arrays.

→ There are two types of Array



- Based on contents, arrays are classified into primitive arrays & derived arrays
- primitive arrays holds primitive data
- derived arrays references of the instance

Q18

a1

```
package com.jspider.array;
```

```
class A
```

```
{
```

```
}
```

```
public class Run4 {
```

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

```
        A[] a1 = new A[4]; → Derived arrays
```

```
        a1[0] = new A();
```

```
        a1[1] = new A();
```

```
        a1[2] = new A();
```

```
        a1[3] = new A();
```

```
        for (A x : a1)
```

```
        {
```

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

```
}
```

Output

```
com.jspider.array.A @ 46caef08
```

```
com.jspider.array.A @ 330de83
```

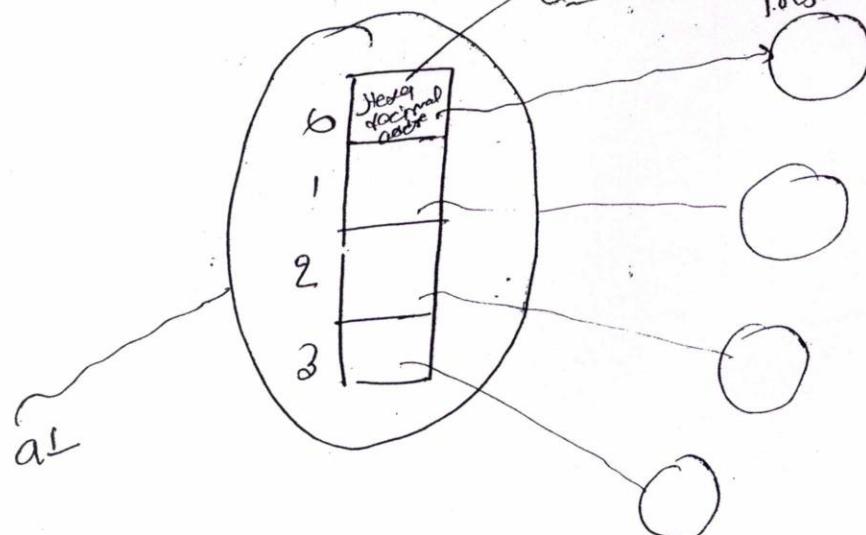
```
----- A @ 890885
```

```
----- A @ 4567897
```

print string representation of
for all array that
contain reference of
instance of A.

reference i.e.
hexadecimal address of instance of A

instance of A



class A

{
A (int i)

{
this. s = i;
}

public String toString()
{
return "i";
}

}

public class Rmt{
p.s.v.m{

pub

A [] a1 = new A [4];

a1 [0] = new A (10);

a1 [1] = new A (20);

a1 [2] = new A (30);

a1 [3] = new A (40);

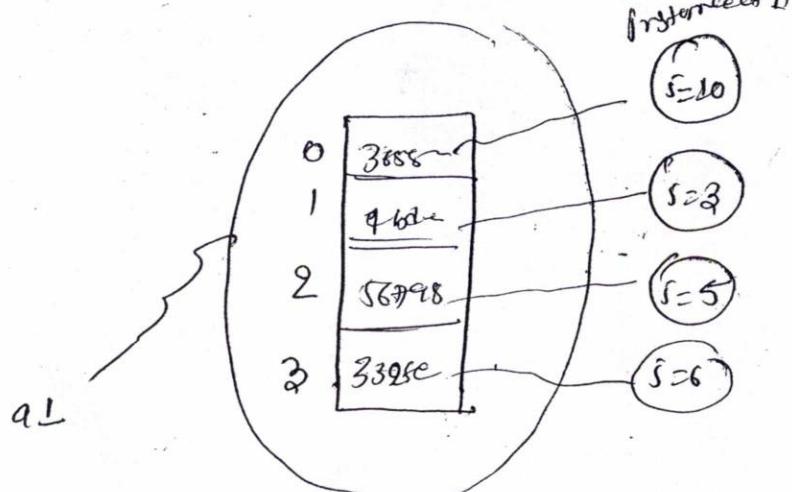
for (A x : a1)

{
x.o.p(x);
}

}

slp

10
20
30
40



slp

}

Class B

{

}

Class C extends B

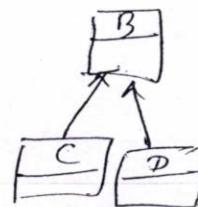
{

}

Class D extends B

{

}



public class Run5 {

 public static void main (String [] args)

{

~~class~~

 C [] c1 = new C [4];

 c1 [0] = new C ();

 c1 [1] = new C ();

 c1 [2] = new C ();

 c1 [3] = new C ();

 D [] d1 = new D [4];

 d1 [0] = new D ();

 d1 [1] = new D ();

 d1 [2] = new D ();

 d1 [3] = new D ();

 B [] b1 = new B [4];

 b1 [0] = ~~c1~~ [0]; // auto upcasting

 b1 [1] = new C (); // auto upcasting

 b1 [2] = new D (); // auto upcasting

 b1 [3] = ~~d1~~ [0]; // auto upcasting.

}

113

Points

- For an array of type of super class, we can assign any sub-class instance reference.
- For ex:, for an array of type animal, we can assign animal instances (dog, cat, ...);
- * An array of type Object can hold any type of data.

package com.jgepides.Arrays;
public class Arrs {

 public static void main() {

 Object[] a = new Object[4];

 a[0] = new Integer(10); // auto upcasting

 a[1] = 20; // auto boxing - auto upcasting

 a[2] = new A(3);

 for (Object x: a)

 {
 System.out.println(x);
 }

Printing
in
Ascending
order.

Output

10
20
3

null → array size is 4 and, we have not stored any data so, get all print null.

Q1

- * If I try to assign primitive data for array of type Object, first the primitive will be boxed to a respective wrapper class and then upcasted to Object.

Sorting Primitive Arrays :

```

package com.jspiders.Arrays;
import java.util.Arrays;
public class Run7
{
    public static void main(String[] args)
    {
        int[] a = { 3, 5, 1, 7, 10, 4 };
        System.out.println("before sorting");
        for(int x : a)
        {
            System.out.println(x);
        }
        Arrays.sort(a);
        System.out.println("after sorting");
        for(int x : a)
        {
            System.out.println(x);
        }
    }
}

```

sort in Ascending order.

before sorting
3
5
1
7
10
4
after sorting
1
3
4
5
7
10

gt is static method that present in Array class

Available in java.util package

Available in java.util package

use to sort the primitive data

114

→ To sort Arrays, we use static sort
method of Arrays class.

→ Arrays class is available in java.util
package!

Assignment

(1) W.A.P to create an integer array
& greatest no. in that without
using sort method

(2) W.A.P to create an integer Array
and sort that array in descending
order

```
package com.jspider.Array;  
import java.util.Arrays  
public class Pm8
```

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

```
{ int [ ] a = { 2, 6, 8, 4, 9, 11 }
```

```
System.out.println ("before sorting");
```

```
for (int x : a)
```

```
{ System.out.print (x); }
```

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

```
{ for (int j = 0; j < i + 1)
```

```
{ max = a[i]; }
```

```
if (a[j] > a[i])
```

```
{ a[i] = max;
```

```
}
```

```
{ }
```

i₂ = ①
int a[6] = { 2, 6, 8, 4, 9, 11 }

2

6

8

4

9

11

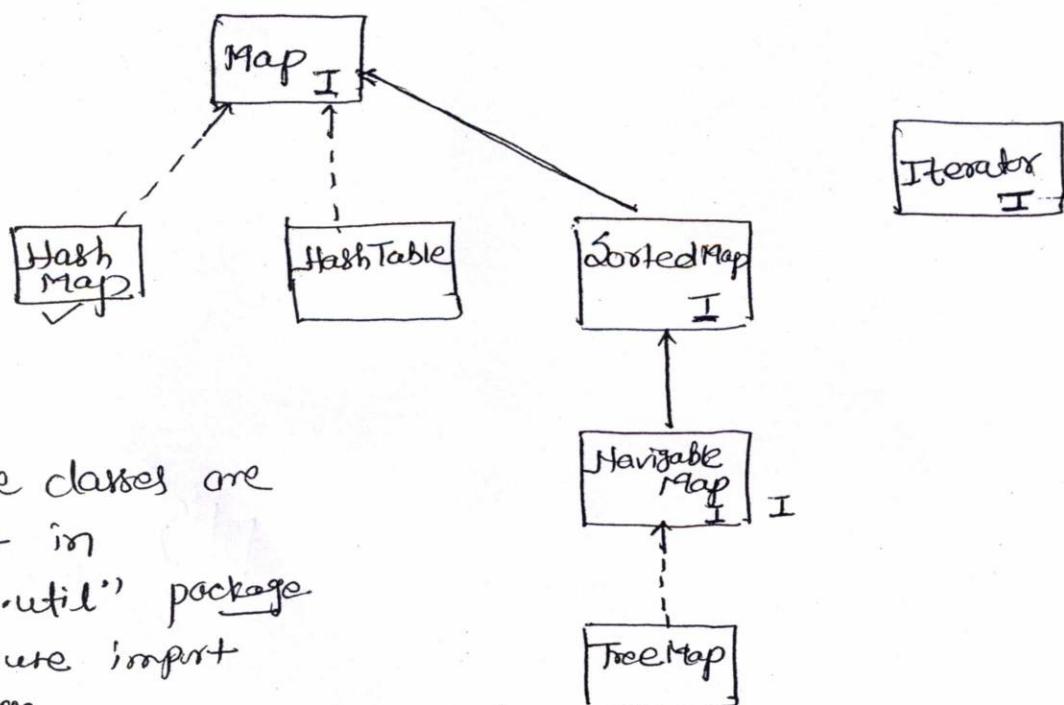
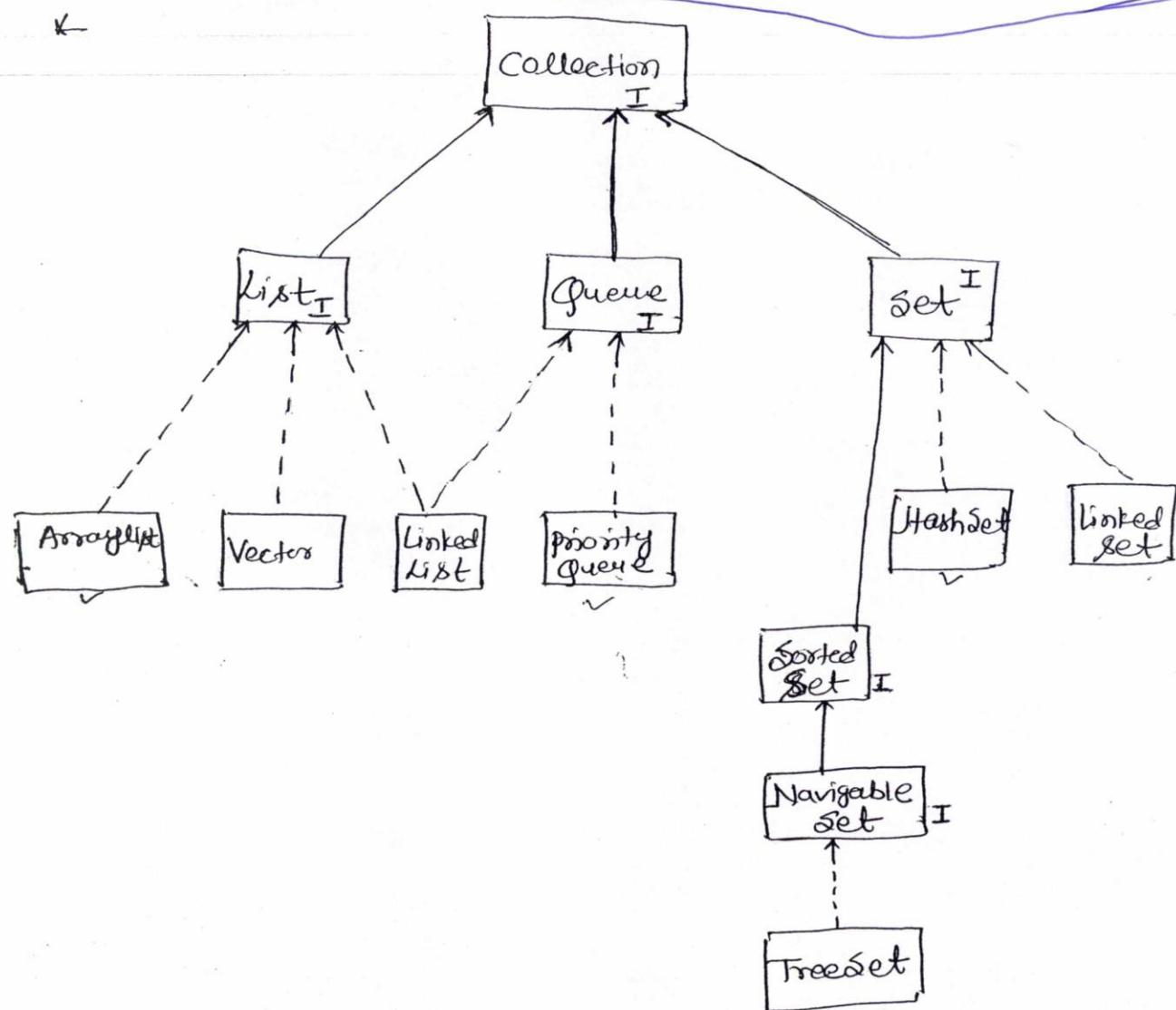
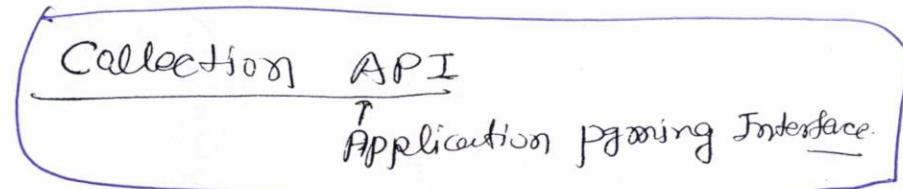
10

(m)

4(B) > a(G)

115

Date
14/05/2022



- * All the classes are present in "java.util" package
- * Go to use import package

- * Collection API is a library of collection classes which are used to store data.
- * All collection class stores data as ~~object~~
a type object
- * All collection classes are available in java.util package.
- * Type Casting i.e upcasting
allows us to create a generic class or Generic methods

List :-

- ArrayList is indexed based
- ArrayList allows duplicate and null values
- Each element in ArrayList is an object type.

methods in ArrayList

do we have
get(element) → index
of elements

add(element) → Add at last index

add(index, element) → Add the element at index
and after that all index shifted

get(index) → get element at index

* check
can we find
get(element)
at index
of element

set(index, element) → store the element at index
with current element, and
size does not change i.e replace
element with argument element.

remove(index) → particular element at index is
remove & size will reduced

remove(element) → remove that element and reduce
size

removeAll → remove all the element

116

size() → give ArrayList size.

```
* import java.util.ArrayList
```

```
public class Run1 {
```

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

```
    { ArrayList a1 = new ArrayList();
```

```
        a1.add(10);
```

```
        a1.add(2);
```

```
        a1.add(4);
```

```
        System.out.println(a1);
```

```
        a1.add(1, 20);
```

```
        System.out.println(a1);
```

```
        a1.set(2, 50);
```

```
        a1.remove(3);
```

```
        System.out.println(a1);
```

```
? 
```

Output

[10, 2, 4]

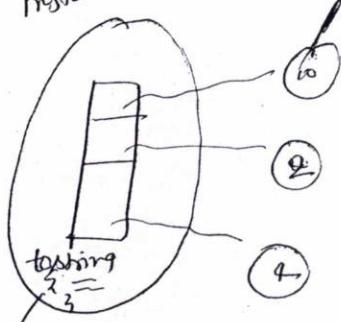
[10, 20, 2, 4]

[10 20 50 4]

[10, 20, 50]

[10]

instance of ArrayList



Here, ArrayList always stores primitive values in arrays as objects, when we pass primitive integer, add method of ArrayList, but automatically boxes it called and change in primitive int into a Integer object using wrapper class.

on ArrayList to String
method is overridden

System.out.println(a1);

when we print reference variable, toString method is called of ArrayList, then it called the toString method of Integer object.

on Integer class, toString() method is overridden so, it returns value at array a1 and no to override the toString method in class

in class

```
System.out.println(a1);
```

```
System.out.println("Size" + a1.size());
```

✓
public
store a1
type
lower
return
at value
size
object
ArrayList

s.o.p ("pointing element using for");

```
for(int i=0; i<a1.size(); i++)
```

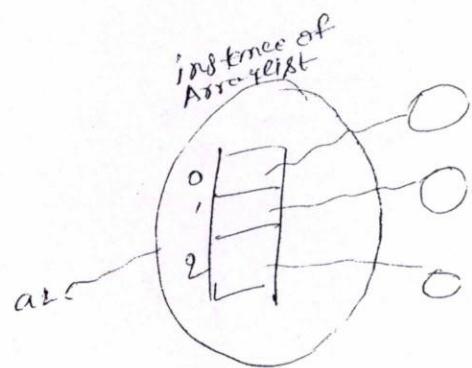
```
{ s.o.p(a1.get(i)); }
```

s.o.p ("pointing elements using for each");

```
for(Object x:a1)
```

All arraylist
Store as object
Type: so
Lowercase
return,
get value
Store in
object type
array. 

```
{ s.o.p(x); }
```



[10, 2, 4, 20]

size = 4;

pointing element using for

10
2
4
20

pointing element using for each

10
2
4
20.

* package com.jspider.collections

class A

```
{ int i;  
A(int i)  
{ this.i=i;  
}
```

public String toString()

```
{ return ""+i;  
}
```

}

117

```
public class Rem2
```

```
{
```

```
    public void print(String[] args)
```

```
{
```

```
        ArrayList al = new ArrayList(5);
```

```
        al.add(new A(2));
```

```
        al.add(new A(3));
```

```
        al.add(new A(10));
```

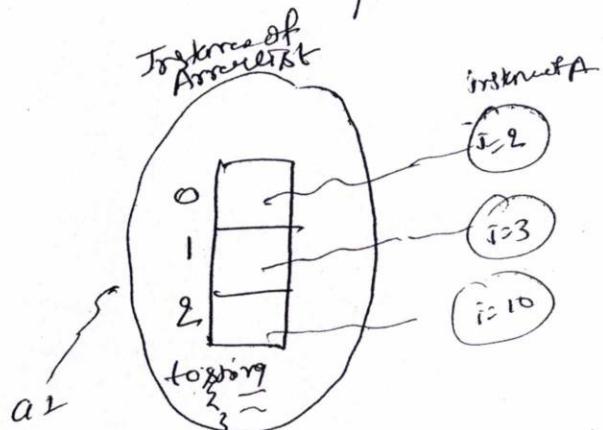
```
    } S.O.P(al);
```

```
}
```

Output: [2, 3, 10]

V.V.P

In Integer class, toString() by default overridden to print content, but in class A by default toString() of object class is called & string representation will print.



Here, in class A, we have to override the toString() method, otherwise sop(a) return the string representation of all three instances.

because, in class A, we must, we have to

override toString() method, when we call, sop(al), to print al, then, toString() method of al is called, then, it print reference variable but

it also called toString() method of class A,

if there is no toString() method in class A,

then, class A is extend by object class, then

toString() method return string representation.

so, ~~class A~~ so, ultimately it return ~~reference~~

string representation. So, to print value or

return value, we must override the toString() method in class A.

Queue :- (Interface)

PriorityQueue

- Queue is not an indexed based.
- Queue implement FIFO concept.
- Queue allows duplicate value,
- Queue does not allow null value.

```

package com.tutorialspoint.collections;
import java.util.PriorityQueue;

public class Run3 {
    public static void main(String[] args) {
        Priority Queue pq = new PriorityQueue();
        pq.add(30);
        pq.add(20);
        pq.add(2);
        pq.add(5);
        pq.add(10);
        // pq.add("manoj") → we can write but at run time
        // it will throw that class cast exception
        System.out.println(pq);
        System.out.println(pq.peek());
        System.out.println(pq.size());
        System.out.println(pq.poll());
        System.out.println(pq.poll());
        System.out.println(pq.poll());
        System.out.println(pq.size());
        System.out.println(pq);
    }
}

O/P [10, 20, 2, 5, 30]
size = 5
  
```

* * *
 The priority queue
 is not indexed
 based, so we
 can not use normal
 for loop, - to access
 the elements we
 have to use only
 advanced for
 loop

118

`add(int)` = only one add method is there, because it is not based on index.

`peak()` → return current head element

`pall()` → remove head element & return that element

In priority queue, it is automatically sorted the element

* `SOP(p)` → It will give random element enter $\{2, 5, 10, 20\}$ but

but

`SOP(pq, peak)` → It will give (2) so, It shows It is sorted.

→ Some type of element only can be stored in queue otherwise it will throw error at run-time.

run-time

when we print reference variable of queue, it gives random output not sorted output

points

when we add elements into a queue, elements are automatically sorted

* `peak()` method returns the head element.

* `pall()` method remove the head element and return the removed element.

* Each time when we use `pall()`, queue size will be reduced by 1.

* To get the 4th element in queue, first we have to remove, first three elements using `pall()`, then read the head element using `peak()`.

But when we use `peak()` & `pall()` method it gives o/p from sorted array in normalist

③ Set :- Interface

- It takes an element of type object.
 - It doesn't allow duplicates & nulls.
 - It should be unique no duplicates.
 - If we have duplicate value & then only one will be stored
- * Package com.jspiders.collections;

```
import java.util.HashSet;
```

```
public class Run4
```

```
{
```

```
PSVM (String [] args)
```

```
{
```

```
HashSet s1 = new HashSet();
```

```
s1.add(10);
```

```
s1.add(20);
```

```
s1.add(10);
```

```
s1.add(30);
```

```
SOP(s1);
```

```
for (Object x : s1)
```

```
{
```

```
SOP(x);
```

```
}
```

counted

so not

duplicate

O/P
Size = 2

[10, 20, 30]

20

10

30

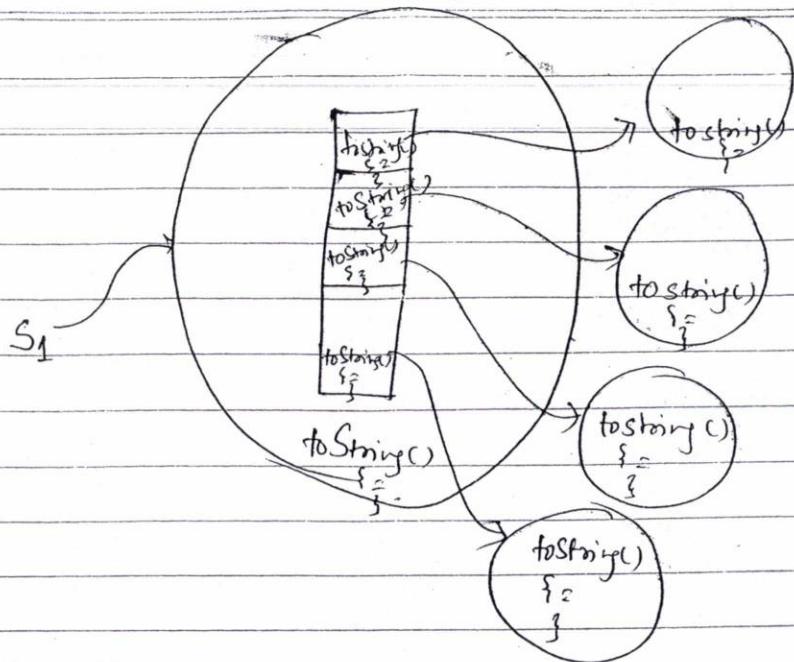
V.V. This

~~* all collection (arrays) does not contain int or any primitive data.~~

every Primitive data are upcasted to Object type & that is refered stored somewhere, & that reference value is stored in array.

See Next Page →

&
 which red box
 portment is
 handled class
 find out



if it is not overridden then we'll
get addrs. of instance only:

(4)

Iterator :-

- ① → Public object next()
- ② → Public boolean hasNext()
- ③ → Public void remove()

- The above methods are used without using the Index.

See
defn in
book

Iterator is an interface, which has a method called Iterator, which is used to iterate the values present inside object through reference.

variable

most defn

(5)

list Iterator

How to access iterator value
in and list Iterator

Eg:

Class A

{

Private class X implements Iterator // No other class
can use this

{

next()

{

}

hashNext()

{

}

remove()

{

}

}

nonstatic methods
of class X which are
implemented here
otherwise it'll become
abstract.

Public Iterator iterator() // nonstatic method

{

Iterator i = (Iterator) new X();

OR

Iterator i = new X(); // auto
upcasting.

return i; // as return type are
same (i.e. Iterator)
so it'll return.

} iterator() \Rightarrow method
I lowercase
Iterator \Rightarrow Interface
U uppercase

or

return new X(); // auto upcasting.

{

} // end of class A.

On another program,

A a1 = new A();

Iterator ito? a1.iterator();

Q. Why iterator() ?

Ans

as class X is Private & its instance can't be used

be outside class A. so we're

creating a method which can take X's
instance & return.

If we call this iterator() method, we'll get a Iterator Object.

Q. return new X(); ?

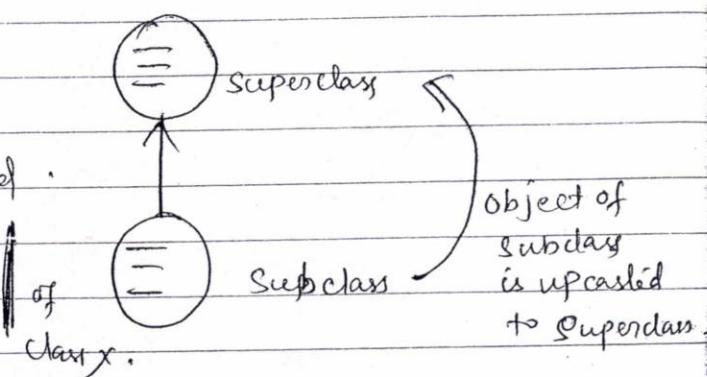
Creating a instance, at runtime it invokes.

if we call using
instance of class X

then method of
class X only called.

that is, next()

hasNext()
remove()



As Superclass is an interface (Iterator)

so it has only declareⁿ no defⁿ so subclass
methods only can be ~~invoked~~ invoked.

A a1 = new A();

Iterator ~~is~~ as. Item

A a1 = new A();

② Iterator it = a1.iterator();

it.next();

it.hasNext();

it.remove();

} of class X methods
are invoked.

Q. Why to implement all those 3 methods?

next() {} becoz, for ArrayList, Queue, Set,

hasNext() {}

remove() {}

PriorityQueue these codes are

different so we use a

Interface where no defⁿ there

in all class, methods already written there in java internally.



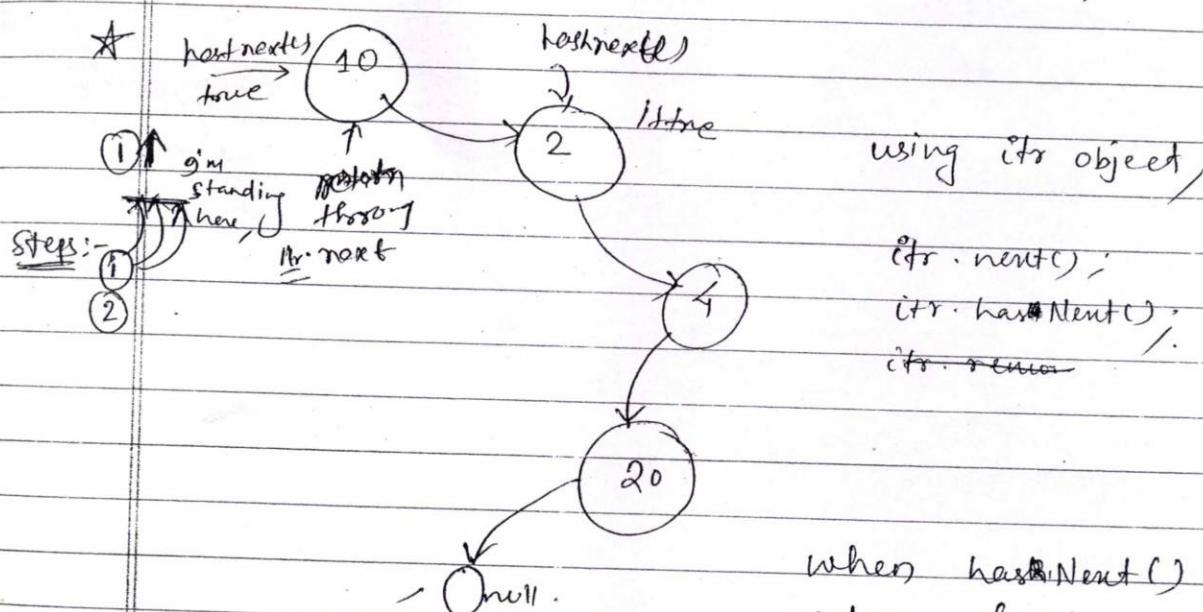
→ Assume class A & class ArrayList.

& along with iterator()
g've add()

Object.

methods are there.

Same for queue, Priority queue, Set, Map....



when hasNext()
returns false we can't
use next() anymore.

A a1 = new A(); // ArrayList a1 = new ArrayList();

Iterator itr = a1.iterator();

while (itr.hasNext())

{
 System.out.println(itr.next());
}

while it'll be executed

fill hasNext()
returns false

fill that it'll read
next element by
using next().

Came for
queue
set
stack
Priority queue.
itr.hasNext()
itr.next()
System.out.println()
}



the class X written there in example,

is not exactly the class X is ArrayList or
any "collection" classes. It has some other name
at there. To know that class name we can see the

```
* Package com.jspiders.collections;
import java.util.ArrayList;
import java.util.Enumeration;
```

```
Public class Run5
```

```
{
```

```
    Psvm (String[] args)
```

```
    { ArrayList a1 = new ArrayList();
```

```
        a1.add(10);
```

```
        a1.add(2);
```

```
        a1.add(4);
```

```
        a1.add(20);
```

```
        Sop(a1);
```

```
Iterator itr = new Iterator();
```

Why wrong?

So we can't create
an instance → interface
of it →

Polyorphism

```
Iterator itr = a1.iterator();
```

```
while (itr.hasNext())
```

```
{
```

```
    Sop(itr.next());
```

```
}
```

```
}
```

Output

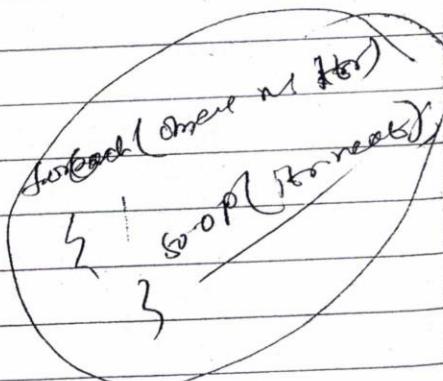
[10, 2, 4, 20]

10

2

4

20



Requirement :- (In Selenium)

web element

Iterator :-

- It is an interface present in (jav.util) package.
- It has 3 abstract methods.

- (1) next()
- (2) hasNext()
- (3) remove()

- (1) next() is used to read the next element.
- (2) hasNext() is used to check the next element is exist or not.
- (3) remove() is used to remove the next element.

- In all collection classes there is a inner class which implements Iterator interface.
(that means all methods are overridden)
- In all collection class there is a ,
public iterator () method which returns
instance of type Iterator.
- Using Iterator instance we can iterate
from first element to last element using
hasNext() & next() .

(5)

Map :-

* Package com.jspiders.collections;

Import java.util.HashMap;

Public class Run6

{

PSVM (String [] args)

{

// HashMap hm = new HashMap(); // or

// hm.put ("sub1", 35); object value

object key

HashMap();

Very useful technique
value can be duplicate

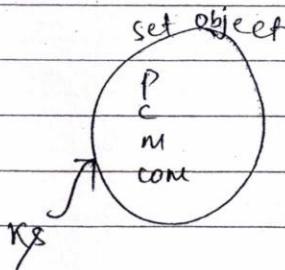
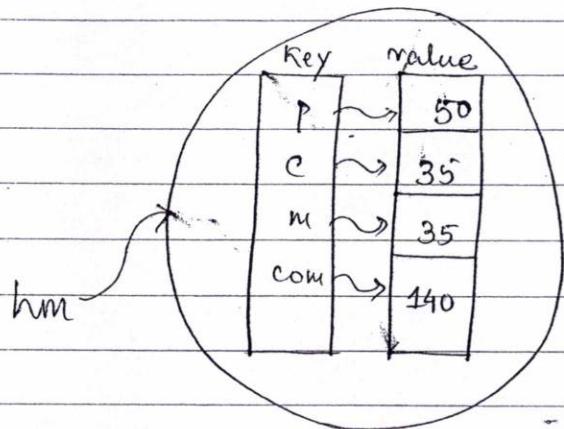
192

```
hm.put("physics", 50);  
hm.put("chemistry", 35);  
hm.put("maths", 35);  
hm.put("computers", 140);
```

SOP(hm); to store overridden
SOP(hm.get("physics")); to print key value
SOP(hm.get("computers")); pair in random
~~order~~

Set ks = hm.keySet();

for(Object n : ks)



O/P

Printing // { computers = 140 , physics = 50 , chemistry = 35 ,
maths = 35 }

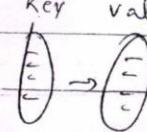
(hm)
object

50 // printing physics marks.

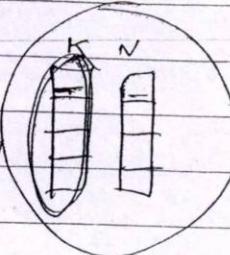
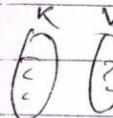
140 // printing Computers marks.

*

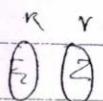
for student 1 → key value



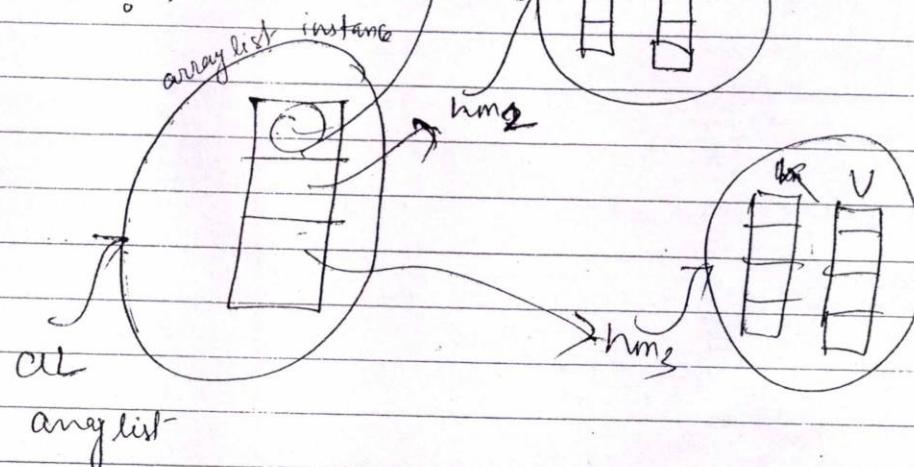
for stud 2 →



for stud 5 →



how to do?



HashMap *hm1* = (HashMap) *al*[0] // downcast .

hm1 . get ("physics") ;

we are downcasting here

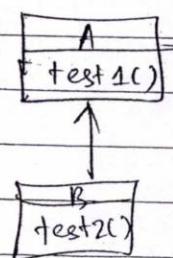
below whenever I'm accessing

(*al*) we are getting address . so downcast

then you can access

all values .

(upcasted to object).



upcasted to A

instance of B

now I can't call

test2() of B

(it'll give error at compile time)

we have to down cast

then call test2()

if overridden

then no need to

downcast becoz at compile time it'll check Yes

test overriding

~~ans~~

```
Package com.jspiders.collections;
import java.util.ArrayList;
import java.util.HashMap;
```

Public class Run5

}

PSVM (String[] args)

{

```
HashMap hm1 = new HashMap();
hm1.put ("Physics", 50);
hm1.put ("Chemistry", 35);
hm1.put ("Maths", 35);
hm1.put ("Computer", 70);
```

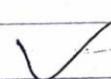
Set RS1 = hm1.keySet();

for (Object n : RS1)

{

SOP (~~n~~; hm1.get(n));

}



```
HashMap hm2 = new HashMap();
```

hm2.put ("Physics", 75);

hm2.put ("Chemistry", 39);

hm2.put ("Maths", 45);

hm2.put ("Computer", 63);

Set RS2 = hm2.keySet();

for (Object n : RS2)

{

SOP (~~n~~; hm2.get(n));

}



```
HashMap hm3 = new HashMap();
```

hm3.put ("Physics", 92);

hm3.put ("Chemistry", 85);

hm3.put ("Maths", 89);

```

    Set ks3 = hm3.keySet();
    for( Object x : ks3 )
    {
        System.out.println(hm3.get(x));
    }
}

```

ArrayList a1 = new ArrayList();

a1.add(hm1);

a1.add(hm2);

a1.add(hm3);

HashMap s1 = (HashMap) a1.get(0);

a1.get(0)

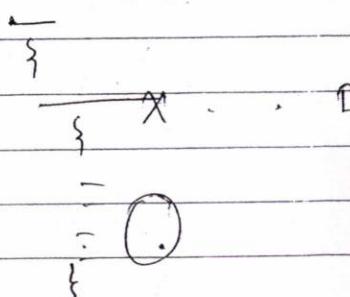
It'll print first hm1 or student details

\rightarrow System.out.println((HashMap)a1.get(0).get("physics"));
 \rightarrow System.out.println((HashMap)a1.get(0).get("chemistry"));
 \rightarrow System.out.println((HashMap)a1.get(0).get("physics"));

 \leftarrow Student 2 details
 \rightarrow System.out.println((HashMap)a1.get(1).get("computer"));

 \leftarrow Student 3 details
 \rightarrow System.out.println((HashMap)a1.get(2).get("maths"));
 \leftarrow

Iterator ?



Iterator()

Object obj = a1.get(0); // hashmap object upcasted to object;

instance

50

35

75

63

89

Object obj = a1.get(0); // hashmap object upcasted to object;

HashMap s1 = (HashMap) obj; // downcasted HashMap

How to get iterator of Map?

Map hm = new HashMap();

Set setViewOfMap = hm.entrySet();

hm.keySet();

Iterator it = setViewOfMap.iterator();

while (it.hasNext())

{

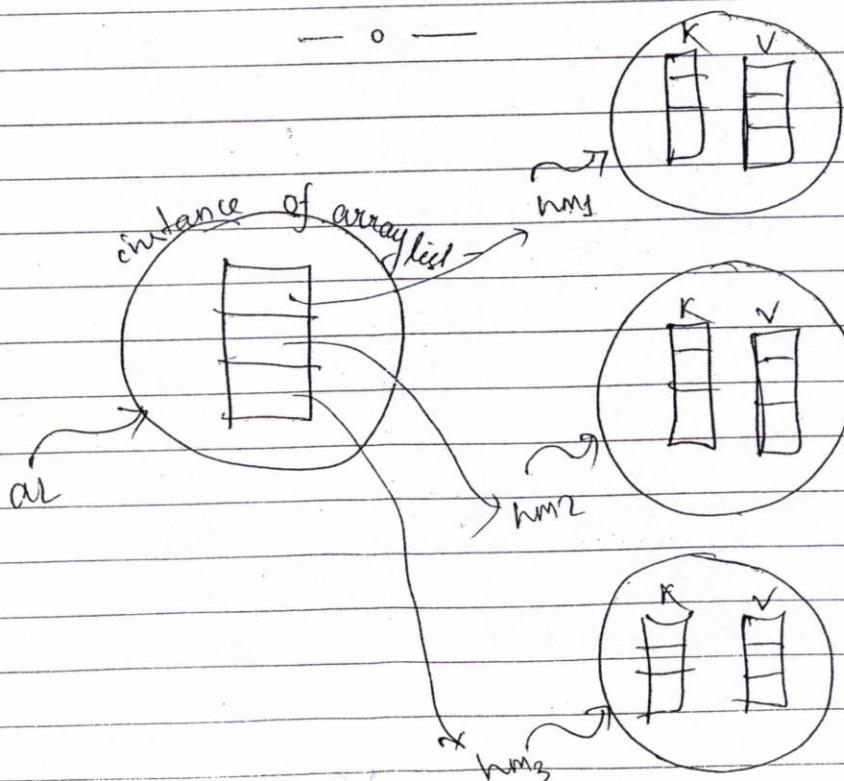
itr.

(Map.Entry) itr.getKey();

.getValue();

* Map does not have iterator.
as Map is derived from Dictionary classes.

→ Map doesn't implement Interface.



Package com.jspiders.collections;
import java.util.ArrayList;
import java.util.HashMap;

Public class Run7

{

PSVM (String [] args)

{

HashMap hm1 = new HashMap();
hm1.put ("P", 65);
hm1.put ("C", 68);

SOP (hm1.get ("P")); }

HashMap hm2 = new HashMap(); hm2.get ("C"); }

or

Set keys1 = hm1.keySet();
for (Object x : keys1)
{
SOP (hm1.get (x));
}

HashMap hm2 = new HashMap();
hm2.put ("P", 63);
hm2.put ("C", 62);

Set keys2 = hm2.keySet();
for (Object x : keys2)
{
SOP (hm2.get (x));
}

HashMap hm3 = new HashMap();
hm3.put ("P", 67);
hm3.put ("C", 69);
Set keys3 = hm3.keySet();
for (Object x : keys3)
{ SOP (hm3.get (x)); }

```
ArrayList al = new ArrayList();
al.add(hm1);
al.add(hm2);
al.add(hm3);
}
```

KeySet():-

KeySet() is a nonstatic method present in a
HashMap object which creates a set
for all keys & returns it.

```
* Package com.jspiders.collections;
import java.util.HashMap;
import java.util.Set;
public class Run
{
    public static void main(String[] args)
    {
        HashMap hm1 = new HashMap();
        hm1.Put("P", 85);
        hm1.Put("C", 50);
        hm1.Put("K", 59);
        hm1.Put("E", 67);
        hm1.Put("B", 83);
    }
}
```

```
Sop( hm1 . get("P"));  
Sop( hm1 . get("C"));  
Sop( hm1 . get("K"));  
Sop( hm1 . get("E"));  
Sop( hm1 . get("B"));
```

// using normal code.

```
Sop( "-----");
```

```
Set KS = hm1.KeySet();
```

```
for ( Object n : KS )
```

{

```
    Sop( hm1 . get(n));
```

}

}

Op

35

50

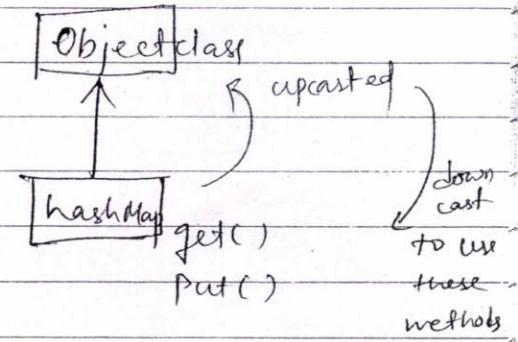
59

67

83

.....

35
50
59
67
83



→ `KeySet()` reads all the elements in a set & passing & passing the key value using `get()`.

`KeySet()` is a type of Set.

Keys have unique value.

→ what code written that it'll do.

★ package com.jspiders.collections;
import java.util.ArrayList;
class Dog
{
 String name;
 public String toString() // overriding the toString()
 {
 return name;
 }

void bark() // new method,
{
 System.out.println("Bow Bow");
 System.out.println(name + " is barking");
}
Dog(String name) // constructor.
{ this.name = name;
}

public class Rang

public void main(String[] args)
{
 Dog d1 = new Dog("pinky");
 System.out.println(d1.bark()); ✓
 System.out.println(d1.toString()); ✓

Object o1 = new Dog("pinky");

System.out.println(o1.bark()); X
System.out.println(o1.toString()); ✓

Dog d = (Dog) o1;
System.out.println(d.bark()); ✓

```
ArrayList dogList = new ArrayList();
```

```
dogList.add(new Dog("Pinky"));
```

```
dogList.add(new Dog("Bulldog"));
```

```
Sop(dogList.get(0).toString()); ✓
```

```
Sop(dogList.get(1).toString()); ✓
```

or

```
{ Object o1 = dogList.get(0);  
Sop(o1.toString()); ✓
```

```
{ Object o2 = dogList.get(1);  
Sop(o2.toString()); ✓
```

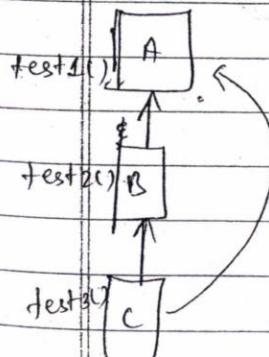
```
Sop(o1.bark()); X
```

```
Sop(o2.bark()); X
```

```
Sop(((Dog)o1).bark()); ✓
```

// down cast & call bark()

```
Sop(((Dog)o2).bark()); ✓
```



c is upcasted to A.

```
A a = new C();
```

now you call test1() ✓

" " test3() X

Even if test3() is there

in C class but it can't call

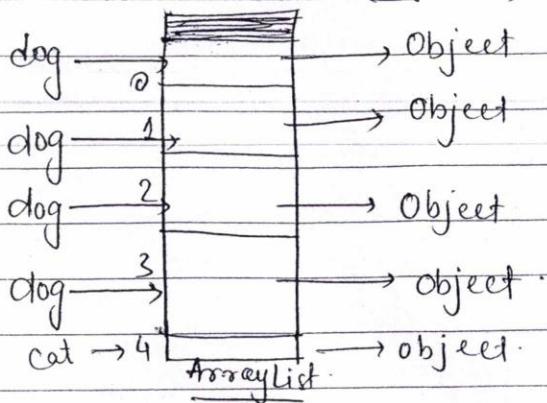
compile time error & becoz { test1() |
test2() } all are
test3() independent method

if overridden then it can

call.

not overridden

using get()



After adding dog to an arrayList it'll be upcasted to Object type & stored.

So we can't use bark() but we can do toString(), equals().

So by using get() we can't get dog we'll get object.

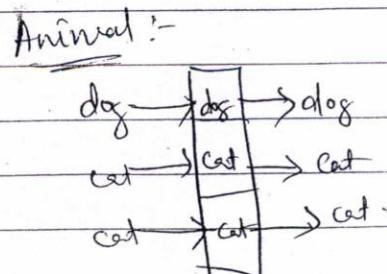
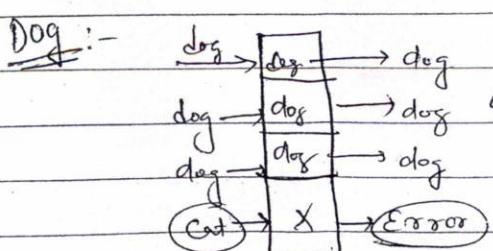
So downcast to get dog again & then call bark(). ✓

- So to overcome this Generics is present.
there is no downcasting headache.
it'll take as dog, stored as a dog, give as a dog only.

Generics:-

Syntax : ArrayList <className> objectName = new ArrayList<className>();

e.g. ArrayList < Dog > dogList = new ArrayList<Dog>();



```

★ Package com.jspiders.collections;
import java.util.ArrayList;

Public class Run10
{
    PSVM (String [] args)
    {
        ArrayList<Dog> dogList = new ArrayList <Dog>();

        dogList.add( new Dog("jimmy"));
        dogList.add( new Dog("tony"));

        dogList.get(0).bark();
        dogList.get(1).bark();
    }
}

```

O/P

► jimmy is barking.
tony is barking.

} As present in same package . . so it can access previous program bark() & class & all.

In Generics :-

- get() takes only Dog type which is specified inside <*>. Other than that type it can't take anything else.

ArrayList :-

- But it can take any thing (Dog, cat, . .) & gives object.

&

ArrayList <webElement> al = new ArrayList();

↓
get webElements
& store as a
webElementonly.

↓
it returns
an arraylist
of webElements.
& returns & store

★ Class A

Public Static B getB()

{
= // gives instance
} of B

} Class B

Public C getC()

{
= // gives
} instance of C.

} Class C

Public void Print()
{
SOP("java");
}

assume A, B, C

all have private constructor
A(), B(), C() ... so

we can't create instance
of A, B, C.

how to create ?

B b1 = A.getB();
(C c1 = b1.getC();
c1.print());

or

A.getB().getC().Print

returns some
object let O1

in that object O2
there is a method
getCC()

in that method
we can again get
an object O2

through O2 we can call
print();

★

Class A

{
Private Dog d;

★ } Class A we can
use to store one
dog object.

B Public void Set(Dog d1)

{
d = d1;

// setter
method

Public void get(*) // getter
method

{
return d;

~~class Dog~~
refresher

A doglist = new A();
 doglist.set(new Dog);
 Dog dog = doglist.get();

- So by using the above we can't store cat, snake . . .

So to store, what "modifica" is to be done?

Class A

{
 Private Animal d;
 Public void set(Animal d1);
 {
 d = d1;

{
 Public Animal get();
 {
 return d;

A al = new A()

doglist.set(new Dog);
 Dog dog = doglist.get();

- Now we can't store car, bike . . .

So to store car - what modifica?

Class A

{
 Private Object d;
 Public void set(Object d1);

{
 d = d1;

Public Object get();

{
 return d;

like this ArrayList is created.

So every time we can't do ~~every~~ ^{this} time.

what
So we'll do is we'll tell user that
you'll only specify that what you want?

class A<T>

{

private T d;

public void set (T d)

{

d = d;

}

public T get()

{

return d;

}

{

A<Dog> a₁ = new A<Dog>();

a₁.set (new Dog (" "));

a₁.get().bark(); // no need to downcast.

A<Cat> a₂ = new A<Cat>();

a₂.set (new Cat (" "));

a₂.get().drinkMilk();

~~A<Dog> a₁ = new A<Dog>();~~

~~a₁.set (new Dog());~~ doglist.set (new Dog());

~~a₁.get();~~ do ~~---~~(

~~a₁.get().bark();~~ // no need to downcast.

A<Dog> a₁ = new A<Dog>();

now Dog is given to T --- returns Dog

now for cat, cat is given to T --- returns cat

This is Generic. It can be otherwise called

as,

Parametrised class / Generic class

(invoked from jdk 1.5.0)

bcz not data, here data type is parametrized

If not passing anything it'll take

Object type

129

Date
16/05/2023

```
package com.jspiders.Collections;  
import java.util.ArrayList;  
import java.util.HashMap  
class public class Rmt  
{  
    public static void main(String[] args)  
    {  
        ArrayList al = new ArrayList;  
        HashMap hm1 = new HashMap();  
        hm1.put("P", 50);  
        hm1.put("C", 48);  
        hm1.  
        HashMap hm2 = new HashMap();  
        hm2.put("P", 65);  
        hm2.put("C", 68);  
        HashMap hm3 = new HashMap();  
        hm3.put("P", 70);  
        hm3.put("C", 88);  
        a.add(hs);  
        a.add(hr);  
        a.add(h3);  
    }  
}
```

```

Object ob = a.get(0);
HashMap & sl = (HashMap) ob;
S.O.P(sl.get("P"));

```

, or

```

S.O.P((HashMap) a.get(0).get("P"));

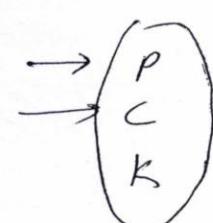
```

~~**~~

```

Set keys = hm1.keySet()
for(Object x : keys)
{
    S.O.P(hm1.get(x))
}

```



get is method in set

it get the all keyset and
there is HashMap to print all
the value to related to keyset

keyset() is a method available in HashMap.

```

hm1.put("P", 67);
hm1.put("C", 80);
hm1.put("K", 70);
hm1.put("E", 37);
hm1.put("B", 59);

```

```

S.O.P(hm1.get("P"));
S.O.P(hm1.get("C"));
S.O.P(hm1.get("K"));
S.O.P(hm1.get("E"));
S.O.P(hm1.get("B"));
S.O.P(" " + " " + " " + " ");

```

Verifiable, we
can write any
string.

Instead of write method

```

Set keys = hm1.keySet();
for(Object x : keys)
{
    IIS.O.P(x);
    S.O.P(hm1.get(x));
}

```

?

6 problem

class Dog

{ String name

Dog(String name)

{ String name = name;

public String toString()

{ return name

void bark

{ sop("Bow Bow");

}

class Run

{ public run (String[] args)

{

Dog d1 = new Dog("pinkey");

d1.bark(); ✓ } no problem
d1.toString(); ✓ here

Object o1 = new Dog ("pinkey");

o1.toString(); ✓ → Here during overriding
program checks method

o1.bark(); ✗ ✗ class toString() method
is parent so no ==

↓ during run time
decide which toString()
method will be executed

Here is problem, that
is bark is independent
method in class Dog. even
though it exists from
object class. so we can't
call bark() method. so
to call we have to
use the object

Dog d = (Dog) o1;

191

* where to downcast class object
public class Run

{ p. s.v.m(--

ArrayList doglist = new ArrayList();

doglist.add (new Dog("pinkiey"));

doglist.add (new Dog("Bulldog"));

object o1 = doglist.get(0);

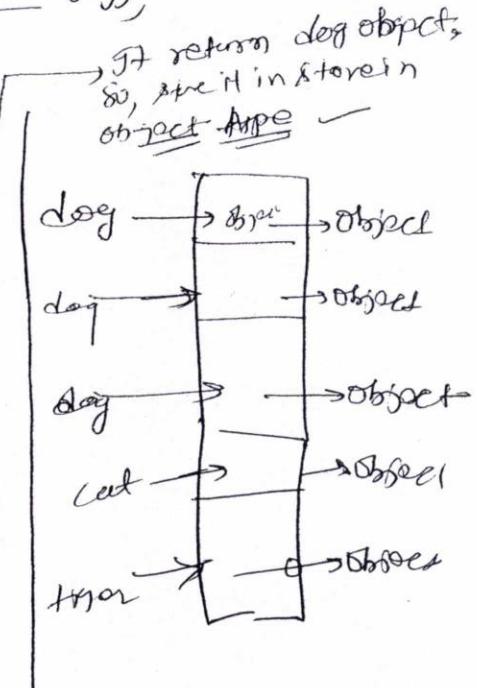
s.o.p (o1.tostring());

((Dog)o1).bark();

object o2 = doglist.get(1);

s.o.p (o2.tostring());

((Dog)o2).bark();



package com.7spider.collections;

import java.util.ArrayList;

class Dog

{ String name;

Dog (String name)

{ this.name = name;

}

public String toString()

{ return name;

}

void bark()

{ s.o.p (name + " is barking");

}

Generic class

```

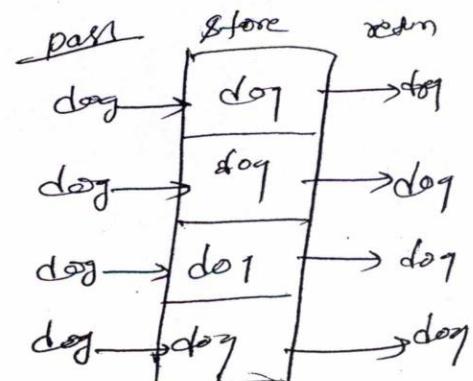
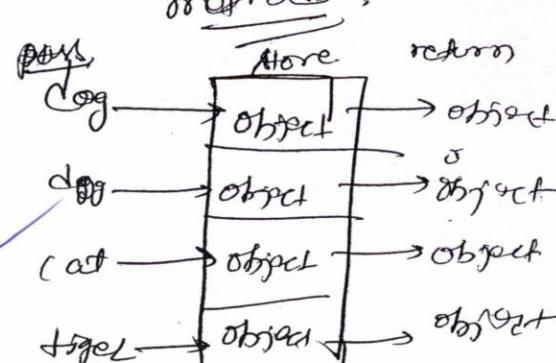
package com.jspiders.collection;
import java.util.ArrayList;
public class Run10
{
    public void m1(String[] args)
    {
        ArrayList<Dog> dogList = new ArrayList<Dog>();
        dogList.add(new Dog("Jony"));
        dogList.add(new Dog("Tommy"));
    }
}

```

`dogList.get(0).bark();` → Here we can
`dogList.get(1).bark();` call bark method
 directly, it has
 created a generic
 object of Dog so
`get()` method

return only the ^{as} Dog, not DogObject.
 So, directly, directly can call bark
 and we can store only Dog.

But if don't write generic
 it can take any type of object and
 return as Object, so specially we have
 down cast exact object and call ~~Object~~
method.



class A

```
{  
    private A U  
    public static B getB()  
}  
{  
    ==  
}  
{  
    --  
}  
}
```

class B

```
{  
    private B U  
    {  
        :  
    }  
    public C getc()  
}{  
    ==  
}  
{  
    --  
}
```

class C

```
{  
    private U  
    {  
        :  
    }  
    public void point()  
    {  
        sup(jam)  
    }  
}
```

How to call, the print method

Because constructor is private
so, we can not create instance.
of class C.

but in class A, protected is
static, with creating an
object, we can method.

B b1 = A.getB(); } } A.getB().getc().print();
C c1 = b1.getc(); } }
c1.print(); }

class A

{

private

(Dog) d;

(Animal)
Object

public void set (Dog d1)

{

d = d1;

(Dog) d1
(Animal)
Object

}

public

(Dog) get()

{

return d;

}

Using Generics,
we can create
parameterized
classes.

class A<T>

{

private T d;

public void set (T d1)

{

d = d1;

}

A <dog> ad = new

A <dog>();

ad.set(new dog(" "));

ad.get().bark();

A <cat> ad = new A <cat>();

ad.set(new cat(" "));
ad.get().drinkmilk();

public T get ()

{

return d;

}

if we ~~don't~~ write without
<>, then, get will take
as an object

139

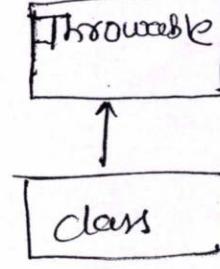
For some class, we are store different type of Animal

- Using Generics, we can create parameterized classes.
- Using Generics we are parameterizing type.
- Generic features were added from 1.5 and 5.0
- all built-in collection classes, are implemented using Generics.
- In the above example, class A is a generic class, T is generic type. The Actual type is decided at the time of execution based on the user input type.

Out
10/10/2012

Exception Handling

- * Exception means an abnormal condition.
- * When program is executing, if java finds any abnormal condition, java throws an exception object; which contains details of the abnormal condition.
- * Java throws only an object of type Throwable (it is class).
- * When exception is thrown, it is mandatory to catch that exception, otherwise, program execution stops.
- * Exceptions are handled in java, using try, catch block.
- * try
{
 ____;
 ____;
}



try is a block, which contains statements or method calls, which might generate exceptions.
- * catch() is a special kind of method if any exception is generated in try, catch argument takes that exception, executes the catch block.

catch (Exception e)
{
 ____;
}
____;

134

- * try, catch(), should written togetherly i.e. no other statement written between try & catch.

```
package com.jspider.exceptionhandling;
```

```
public class Run1
```

```
{
```

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

```
{
```

```
    System.out.println("Program Starts");
```

```
    int i=10;
```

```
    int j=0;
```

```
    int k=0;
```

```
    try
```

```
{
```

```
    k=i/j;
```

```
} catch(ArithmaticException e){
```

```
{
```

```
    System.out.println("Exception is caught");
```

```
    System.out.println(e.getMessage());
```

```
    System.out.println("i=" + i);
```

```
    System.out.println("j=" + j);
```

```
    System.out.println("k=" + k);
```

```
}
```

```
System.out.println("Program ends");
```

Here automatic
exception object will be printed
that present exception
object.

```
}
```

O/P

Program starts.

// Exception is caught:

10
0
0

Program ends.

* If java gets any exception in try block

it immediately leave the try block, and

go to the catch block, and handled the exception

through catch block and continue through the

program till ends, but after exception find if

will not execute the statement after exception

public class Pgm2

{
 psvm (String [] args)

{
 s.o.p("Pgm starts");

 int i = 10;

 int j = 0;

 int k = 0;

 try {

 s.o.p("try starts");

 k = i/j;

 s.o.p("try ends");

 } catch (ArithmaticException e) {

 s.o.p("inside catch");

 s.o.p(e.getMessage());

}

As exception
found, flow
now
does not
go back
execute
constant
rest of the
main
parts

 s.o.p("i = " + i);

 s.o.p("j = " + j);

 s.o.p("k = " + k);

}

O/P

Pgm starts

try starts

inside catch

* by zero

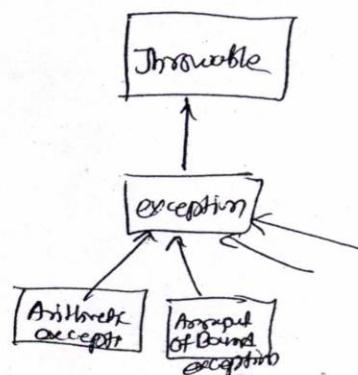
i=10

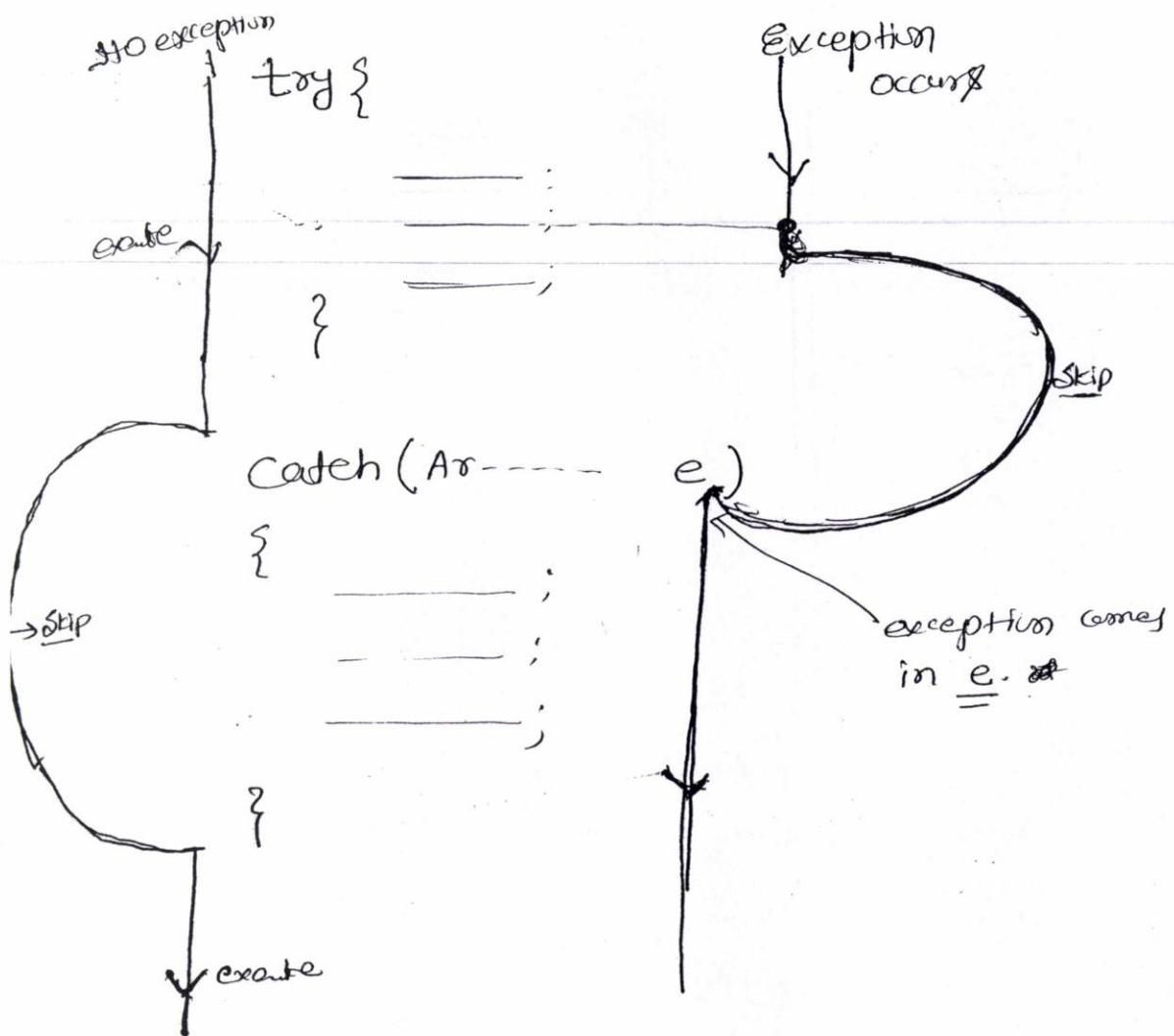
j=0

k=0

As exception occur it
jump to directly to
catch block & execute.

185





* If there is no exception in try, All the statements in try will be executed, and catch block will be skipped.

* If there is an exception in try block, immediately Java stops executing the try block.. and it executes the catch block ..

```
public class Run2
{
    public static void main()
    {
        int arr = new int[2];
        try
        {
            arr[0] = 10;
            arr[1] = 12;
        } catch( --- )
        {
            catch (ArrayIndexOutOfBoundsException e)
            {
                System.out.println("Inside 2 catch");
            }
        }
    }
}
```

*previous program without
adding else part here.*

one more catch block

~~**~~ A try{} can have multiple catch() block,
But it executes the catch block based on
exception thrown.

* A try{} can have multiple catches-

public class Run2

{

 P.S.V.m (String[] args)

{

 S.O.P("Program starts");

 int i = 10;

 int j = 0;

 int k = 0;

~~int[] arr = new int[2];~~

 try {

 S.O.P("try starts");

 k = i / j;

 arr[2] = 12;

 S.O.P("try ends");

 } catch (ArithmaticException e) {

 S.O.P("Inside 1st catch");

 S.O.P(e.getMessage());

 } catch (ArrayIndexOutOfBoundsException e) {

 S.O.P("Inside ~~except~~ 2nd catch");

 } S.O.P(e.getMessage());

 S.O.P("i = " + i);

 S.O.P("j = " + j);

 S.O.P("k = " + k);

 } S.O.P("end");

 O/P

 Program starts:

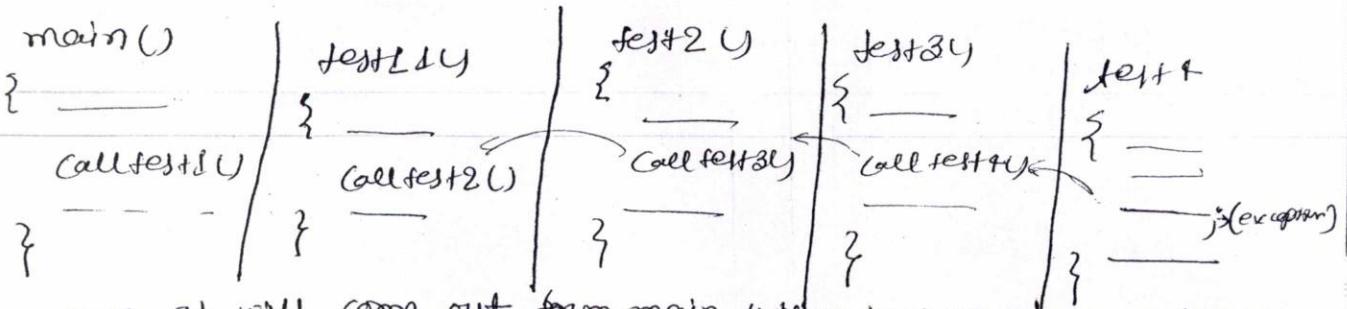
 try starts

 inside 2nd catch

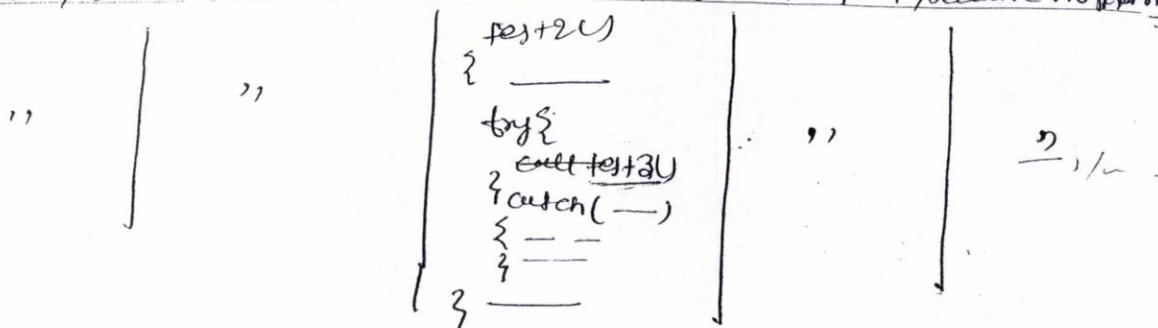
 2
 62 80

196

* Exception propagation :-



Here, gt will come out from main, & goes to sum, because no blocks



package com.jspider.exceptionhandling;

```

public class Rem2
{
    p.s.v.m (String [] args)
    {
        S.O.P ("main starts");
        test1();
        S.O.P ("main ends");
    }
}
  
```

```

static void test1()
{
    S.O.P ("test1 starts");
    test2();
    S.O.P ("test2 ends");
}
  
```

```

static void test2()
{
    S.O.P ("test2 starts");
    try
    {
        test3();
    }
    catch (Exception e)
    {
        S.O.P ("exception is caught");
    }
}
  
```

```

    S.O.P ("test2 ends");
}
  
```

```

static void test3()
{
    S.O.P ("test3 starts");
    throw e;
}
  
```

```

static void test4()
{
    S.O.P ("test4 starts");
    int i = 10/0;
    S.O.P ("test4 ends");
}
  
```

O/P

main starts

test1 starts

test2 starts

test3 starts

test4 starts

Exception is caught

test2 ends

test1 ends

main ends

Here exception is caught in test2 from test3, remaining code will execute

Date
21/05/2013

Exception propagation :-

- * When exception occurs in any method, it checks for ~~a~~ handlers in that method. If no handler was found, exception is propagated to its parent method by removing the current method from the stack.
- * If the parent method, also, there is no handler, then exception is propagated to its parent method by removing that method from the stack.
This process repeats till an exception finds ~~a~~ handler.
- * If there is no handler in entire program, then exception is propagated to JRE (i.e. out of main method). Then JRE brings stack trace, & program execution will be stopped.

by me

In above, program, if, there is no catch block in program code above, it gives exception that from where it is generated from the end.

Q.P

Exception is caught at main method

→ guru
called
Stack
trace

- * we can print the stack-trace also.

C. printStackTrace() → method

187

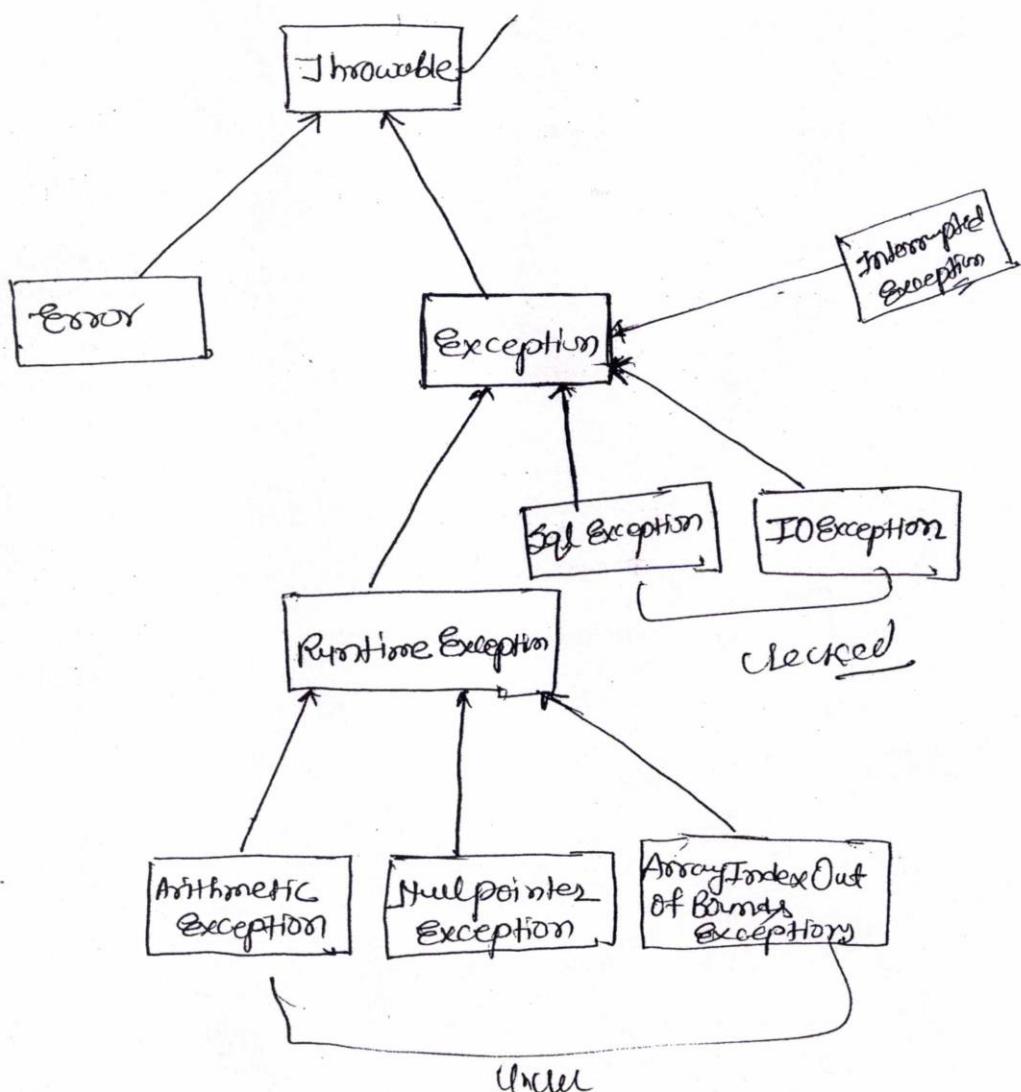
Throws

- * Throws is a keyword which is used to throw ^{or generate} exception.
- * Using throws, we can throw only the instances of type throwable.

ex:-

```
class A extends Exception  
{
```

```
    }  
    throw new A(); // or throw i;
```



Types of exception

- There are two types of exception
 - i) checked Exception.
 - ii) unchecked Exception.
- Exceptions that are detected at the time of compilation are known as checked exception.
If checked Exceptions are not handled, then Java compiler does not allow to compile. (mandatory explicitly we have to handle in program).
- Exceptions that are not detected at the time of compilation are known as unchecked exception (Run-time exceptions).
- exception classes which extends Run-time exception come under unchecked exception.
- exception classes which extends exception come under checked exception.

package com.jspiders.exceptionhandling;

public class RunTimeException

{ p.s.v.m (String[] args)

{ t = new Thread();
t.start();
S.O.P ("i=" + i);

try

{ Thread.sleep (1000)

} catch (InterruptedException e) {

e.printStackTrace();

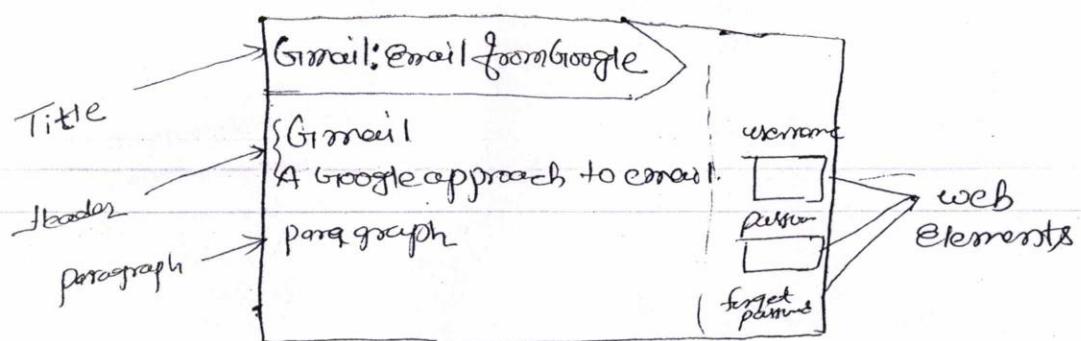
138

is in checked exception

Some other thread might interrupted sleep thread during run time, so it may be generate exception & program aborted

If we don't put Thread.sleep(1000) in try block, it will give compile time error, that's it might throw exception so plz handle it & it's called checked exception

2013
10/05/2013



* HTML Sample program

```
<html>
```

```
 <title> My custom webpage </title>
```

```
 <body>
```

```
   <h1> Welcome to new webpage </h1>
```

```
 </body>
```

```
</html>
```

- HTML language is used to describe the web-pages.
- HTML is main mark-up language to creating a web-page and information can be described in web-browser.
- HTML is written in the form of HTML elements, consisting of tag enclosed in angle "<>" brackets.
- HTML tags most commonly comes with pairs.
- HTML elements form the building blocks of all websites.
- HTML allows to create images and objects
- to create edit box tag:— then type = "text"

```
<input type = "text", id = "User", name = "User" />   
mandatory      optional      optional.
```

Username : `<input type = "text", id = "User", name = "User" />` Username:

Password : `<input type = "text", id = "Pass", name = "Pass" />` Password:

to create Button:-

<input type='button' name='btn' id='btnd' />  
button created

- In the above form, sleep() is static method that present in Thread class.
- sleep() method might throw interrupted exception during execution.
- Since, InterruptedException is checked exception, we have to handle mandatory at the time of compilation.

package com.jspiders.exceptionhandling;

class Account

```
{  
    private int accNumber;  
    private int balance;  
  
    Account(int accNo, int bal)  
    {  
        accNumber = accNo;  
        balance = bal;  
        o.o.p ("Account is created");  
    }  
}
```

```
public int getBalance()  
{  
    return balance;  
}
```

189 public void deposit (int amt)
{
 balance + = amt;



```
public void withdraw(int amt) throws InsufficientfundException
```

```
{  
    if (amt <= balance)
```

```
{  
    balance -= amt;  
    System.out.println("amt " + amt + " is withdrawn");
```

```
else
```

```
{  
    int shortage = amt - balance;
```

```
    throw new InsufficientfundException("you have "+  
        shortage + " shortage");
```

```
Class InsufficientfundException extends Exception
```

```
{  
    String msg;
```

```
InsufficientfundException(String s)
```

```
{  
    msg = s;
```

```
public String toString()
```

```
{  
    return msg;
```

```
public class Pmt
```

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

```
{  
    System.out.println("PMT starts");
```

```
    Account ac1 = new Account(11, 2000);
```

a1. deposit(3000);

s.o.p("Balancee;" + a1.getBalance());

try {

a1. withdraw(4000);

} catch (InsufficientFundException e) {

} s.o.p(e);

s.o.p("Balancee;" + a1.getBalance());

try {

a1. withdraw(3000);

} catch (InsufficientFundException e) {

s.o.p(e);

}

s.o.p("Balancee;" + a1.getBalance());

s.o.p("ends");

}

O/P

program starts

Account is created

Balancee: 3000

deposited 3000 is

Balancee: 6000

4000 is withdrawn

Balancee: 2000.

you have 2000 is shortage

Balancee: 1000.

140

on de..

Interview question

① How do we handle checked exceptions.

↓
Ans
two ways

① ~~error~~ using try catch

② or by using throws

* How do we handle checked Exceptions?

Ans by two ways:

(1) using try catch.

(2) using throws → whether throw or throws

* diff b/w throw, throws, Throwable.

→ throw is a keyword used to generate an exception.

→ throws is a keyword which is declared what type exception a method might throw.

→ Throwable is a class.

All exception classes are inherited from Throwable class.

```
package com.jspider.exceptionhandling;
```

```
public class Run5{
```

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

```
        System.out.println("main starts");
```

```
        try{
```

```
            System.out.println("try starts");
```

```
            int s=10/0;
```

```
        } catch(ArithmaticException e){
```

```
            System.out.println("catch block");
```

```
        }
```

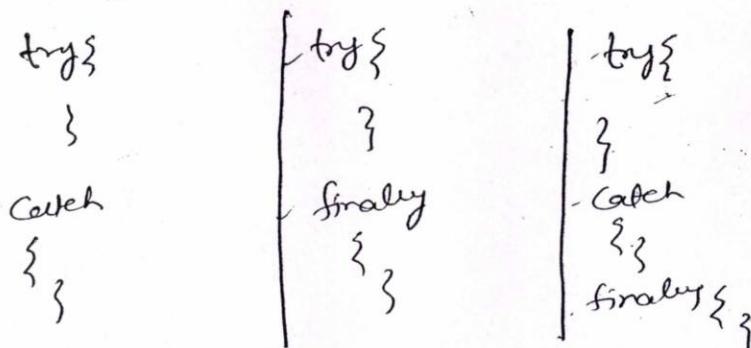
```
        finally{
```

```
            System.out.println("inside finally");
```

$\text{S.O.P} ("j = " + j);$
 3 } $\text{S.O.P} (" \text{main ends} ");$
OLP — ~~Finally~~

Finally → it is class or Block.

- Finally block executes immediately whether exception occur or not.
- we can write try Block three ways



we can write try either with catch or finally Block. But try can not be alone

- Between try, catch and finally, no other statement can be occurred
- we can not write alone try
- try should always contain either catch or finally or both
- when we are writing, try, catch, finally, then order should be try → catch → finally.
- No other java statements are allowed b/w try, catch & finally

~~Inside, the finally, we write a mandatory statement, such as closing, the database connection, closing the N/w connection, closing file etc.~~

Q1

* whether finally can be written alone

extra covered

class tiger

{ int size;

static int count = 0;

tiger(int size)

{ this.size = size;

height = 0;

}

void print()

{ System.out.println("Tiger " + size);

System.out.println(" or instead " + count);

)

Pr

PrintWriter pw = new PrintWriter(10);

for (int i = 0; i < 8; i++)

pw.println(" " + i);

pw.println("Total Height (read as " + tigers)");

pw.println("Total Height (read as " + tigers));

opening @ <http://vista.com>

guru
testing

* Md. sahir — 9738186603 - Jspider

Email: id ↗

201 → 61 →
314 → 620

Date
22/05/2013

Multitasking

- perform multiple task parallel known as multi-tasking.
- To achieve multi-tasking in program or application, we have to develop multi-threads.
- Thread is nothing, it is an independent flow of execution.
- when we execute class file, main() thread starts.

* We can create thread in two ways:-

- ① Using Thread class
- ② Using Runnable Interface.

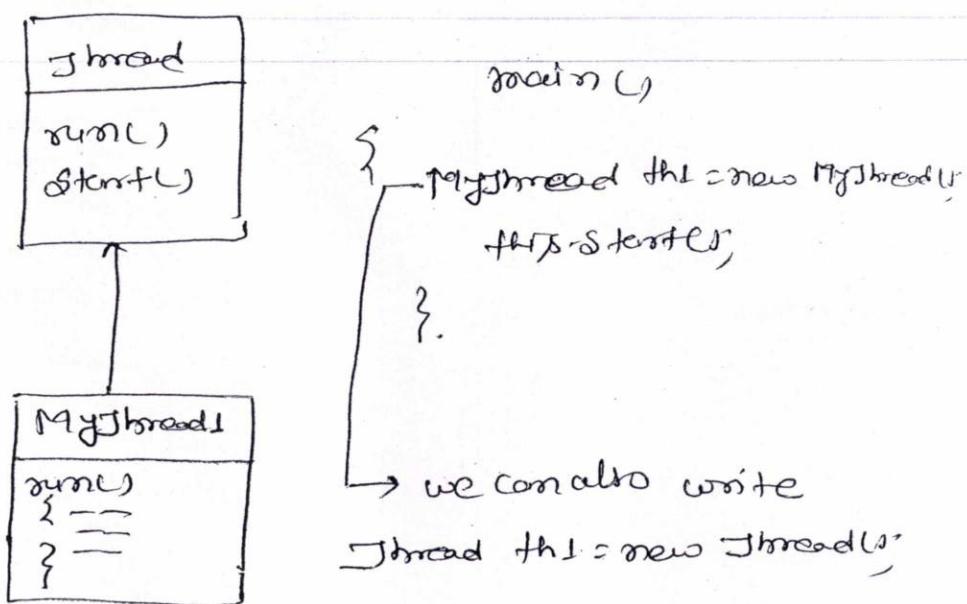
package com.jspider.multitasking

```
public class Run1
{
    public static void main(String[] args)
    {
        for (int i=1; i<=20; i++)
        {
            System.out.println("i=" + i);
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        for (int j=21; j<=30; j++)
        {
            System.out.println("j=" + j);
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

142

when we execute file program will give sequential output not parallel.
first of all main thread starts, first for loop starts & print the value on console then after ending the end for loop will start again.

Using Thread class



- Create a class which extends Thread class.
- Thread class has two methods.
 - (i) Run()
 - (ii) Start()
- In sub-class override Run method (write the code which we want to execute as separate thread).
- In Main method, create an instance of sub-class & call Start() method
 - ↳ [for every thread separate heap will be created].

package com.jspiders.Threads;
class MyThread1 extends Thread

```
{ public void run()
{
    System.out.println("MyThread1 Started");
    for(int i = 21; i<=40; i++)
    {
        System.out.println("I = "+i);
    }
}
```

Java
uni
alt
auto
upla
over
over
retro
metho
all

```

try
{
    Thread.sleep(1000),
}
catch (InterruptedException e)
{
    e.printStackTrace();
}
}
}

```

public class Run

```

{
    public void run()
    {
        System.out.println("Main thread starts");
    }
}

```

MyThread th = new MyThread();

However we can write thread also get will automatically updating & whenever we call run or starts method, say if method can be executed

th.start();

for (int i=1; i<=20; i++)

```

{
    System.out.println("i=" + i);
}
}

```

try
{
 Thread.sleep(1000);
}

}
catch (InterruptedException e)

```

{
    e.printStackTrace();
}
}
}

```

System.out.println("Main thread ends");

O/P

Main thread starts



L49

- in O/P
Here both for loop
parallelly or
simultaneously
executed

* for each Thread, there is separate memory created. Every Thread has separate heap & stack and Thread is scheduled by thread scheduler. Each Thread is different flow of execution.

public class myThread1 extends Thread

{

s.o.p("myThread1 Starts");

public void run()

{

s.o.p("myThread1 Starts");

for(i=1; i<=20; i++)

{

s.o.p(i);

try

{ Thread.sleep(1000);

} catch(InterruptedException){

e.printStackTrace();

}

s.o.p("Thread1 ends");

public class myThread2 extends Thread

{

public void run()

{

s.o.p("Thread2 starts");

for(j=21; j<40; j++)

{

s.o.p(j);

try

{ Thread.sleep(1000);

} catch(InterruptedException){

e.printStackTrace();

```
{  
    s.o.p("Thread 2 ends");  
}
```

public class Run2

```
{  
    p.s.v.m(--);
```

```
{  
    s.o.p("Main thread starts");
```

```
myThread1 th1 = new myThread1();  
th1.start();
```

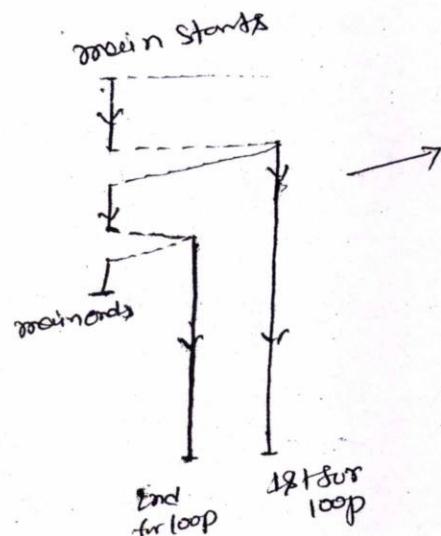
```
myThread2 th2 = new myThread2();  
th2.start();
```

```
s.o.p("Main ends");
```

```
}
```

Q.

O/P

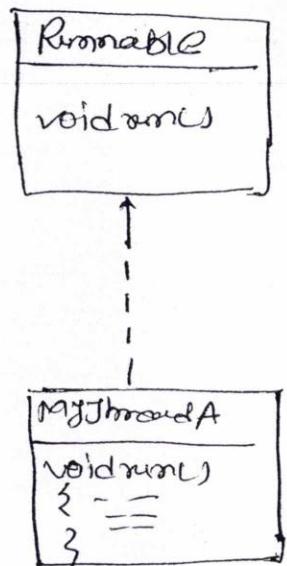


Here main ends
as it starts
both the thread.
after ending main,
other thread still
executing because all
thread have separate
memory and separately
executing.

144

15

Using Runnable Interface:



```

} others Thread
} Runnable &
Thread()
} }

Thread( Runnable r )
{ this.r=r;
}
if(r==null)
{ run();
}
else{ run(); }
  
```

The code block is enclosed in curly braces and describes the implementation of the `run` method in the `Thread` class. It checks if the parameter `r` is null, and if not, it calls the `run` method of the `Runnable` object `r`.

```

Thread th1 = new Thread( new
                        ThreadA() );
th1.start();
  
```

This code creates a new thread `th1` by passing an instance of the `ThreadA` class to the constructor of the `Thread` class. It then starts the thread by calling the `start` method.

Why we use Interface `Runnable`, because of a class already some other class, then we can not extend one more ~~class~~ Thread class. because multiple inheritance not allowed so, it is interface, now so any class can implement interface either A class extends a class or not.

- Create A class which implements `Runnable` interface.
 - `Runnable` Interface has abstract run method.
 - In the sub-class override & complete the run method (write the code, which we want to execute as separate thread).
 - * Create an instance of Thread, by passing ~~a type~~ instance of type Runnable
 - * Call start method of Thread instance
- Thread runnable
- (Start) (run)

class A implements Runnable

```
{ public void run()
{ S.O.P("JHL Starts");
for(int i=1; i<=20; i++)
{ S.O.P(" i=" + i);
try { Thread.sleep(1000);
} catch(InterruptedException e) {
{ e.printStackTrace();
S.O.P("JHL ends");
}
```

class B implements Runnable

```
{ public void run()
{
    for(i=20; i<=40; i++)
    { S.O.P("i");
}
```

public class Run2

```
{ P.S.V.M(--)

{ S.O.P("main Thread Starts");
Thread th1 = new Thread(new AL());
Thread th2 = new Thread(new BL());
th1.start();
th2.start();
S.O.P("main ends");
}
```

package com.jspider.Threads

class pointer

```
{ synchronized void point( String msg)
```

```
{ int len = msg.length();
for(int i=0; i<len; i++)
{ S.O.P("msg.charAt(" + i + ")");
}
```

```

    } } Thread.sleep(500);
} catch(InterruptedException e) {
    } } e.printStackTrace();
}

```

Dec
24/12

class printThread implements Runnable

```

{
    pointer P;
    String S;
    printThread(pointer P, string S)
    {
        this.P = P; this.S = S;
    }
    public void run()
    {
        P.print(S);
    }
}

```

public class Run3

```

{
    p.s.v.println("Main Starts");
    System.out.println("Main Starts");
    pointer P = new pointer();
    Thread t1 = new Thread(new printThread(P, "Java"));
    t1.start();
    Thread t2 = new Thread(new printThread(P, "development"));
    t2.start();
    Thread t3 = new Thread(new printThread(P, "Jspiders"));
    t3.start();
    System.out.println("Main ends");
}

```

Q10 : main starts

main ends

a
g
e
r
e
y
T
S
P
d
e

Synchronization :-

When two or more threads need access to shared resources, they need some way to ensure that the resources will be used by only one thread at a time. The process by which this is achieved is called synchronization.

Date
24/05/2013

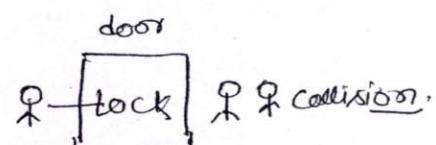
* Thread Synchronization :-

- when two or more thread accessing the shared resources (shared instance), changes made by one thread can effect the execution of the other thread.
- we can solve the above problem by synchronizing threads.
- To synchronize the threads, declare the shared resources methods as synchronized.

Syntax

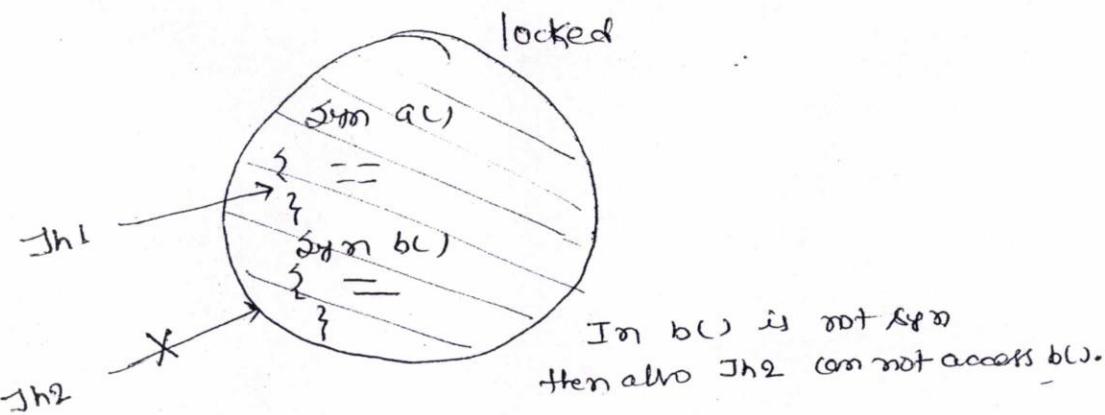
```
class Printer
{
    synchronized void print (String msg)
```

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



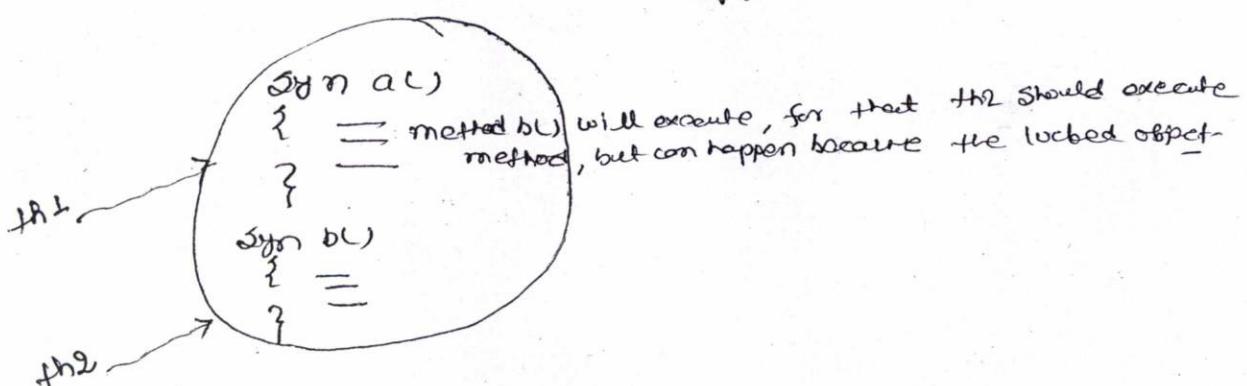
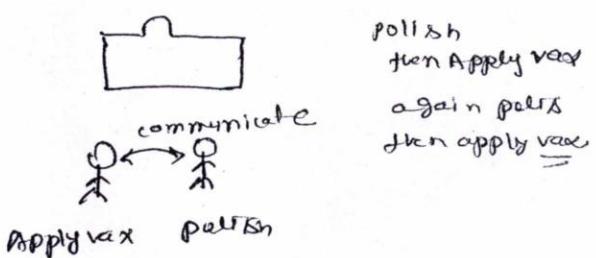
- when a thread enters into a synchronize method, it creates an object lock so that other threads can not use (call) any methods of the locked object. (even they can not even access methods if the using methods)
- Object lock will be released when the the thread ~~lets~~ complete its job & coming out from a synchronized method.
- when object lock is released the other thread enter into method & it can start its execution.

- If a class contains two synchronize method a() & b(), if a thread creates a lock, while entering into method a(), other thread can not call method b() also until an object lock is released.



Technique

- * what is thread synchronization.



wait() → object class they are present

notify()

notifyAll → notify all the thread in wait pool

→ wait() & notify() should not be used for non

class factory

{
 private int stock;

synchronized void produce (int n)

{
 stock += n;
 S.O.P ("stock is updated");
 notify();
}

synchronized void consumer (int n)

{
 if (stock < n)
 {
 S.O.P ("out of stock");
 try {
 wait();
 } catch (InterruptedException e)
 {
 e.printStackTrace();
 }
 }
}

stock -= n;

S.O.P ("car are consumed");

}

class prodThread implement Runnable

{
 factory f;
 int n;
 prodThread (factory f, int n) {
 this.f = f;
 this.n = n;
 }

```
public void run()
{
    f.produce(n);
}
```

class consumer Thread implements Runnable

```
{
    factory f;
    int n;
    consumerThread (factory f, int n)
}
```

```
{    thr.f=f;
    thr.n=n;
}
```

```
public void run()
```

```
{    f.consumer(n);
}
```

```
}
```

public class Run4

```
{    p.s.v.m (String [] args)
```

```
{        s.o.p ("main starts");
    }
```

```
    factory f = new factory();

```

```
    Thread th1 = new Thread (new consumerThread
        (f, 2));

```

```
    th1.start();

```

```
    Thread th2 = new Thread (new producerThread
        (f, 2));

```

```
    th2.start();

```

Q10

main starts
out of stock
main ends
stock is updated
car is consumed.

Thread Deadlock:-

- when multiple thread accessing common resources, thread 1 completes its job if we gets some update from thread 2, but thread 2 enter & execute until thread 1 completes its job. This is known as Thread deadlock.
- We resolve Thread deadlock problem by establishing inter-thread communication.
- We can achieve inter thread communication using wait() & notify() method.
Both the methods are defined in Object class.
- When a thread executes wait() method, it temporarily releases object block and waits in wait pool.
- The second thread can enter into an object and it can start its execution.
- When second thread completes its execution, it executes notify method & comes out from the method.

148

- Notify method notifies the thread, which is waiting in waiting pool.
- when a thread gets notification, it resumes the execution from where it has stopped
- wait is diffn b/w sleep & wait()
 - sleep does not sync
 - wait() releases the object class

Sleep()

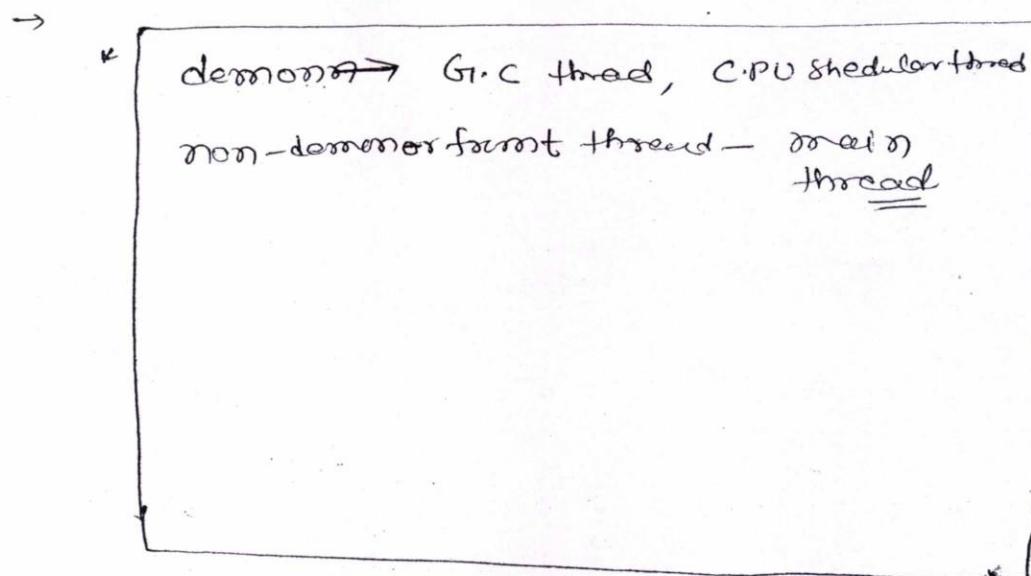
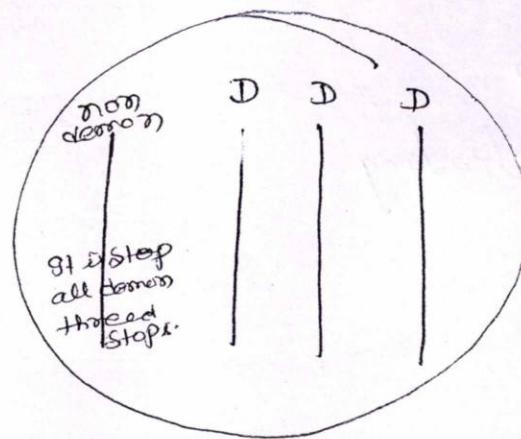
- Sleep is static method defined in Thread class.
- suspends the thread execution without releasing object lock.
- we can call Sleep method in both syn & non syn method.

wait()

- wait is non-static method defined in object class.
- wait suspends the thread execution by releasing the object block.
- wait can only be in synchronized method

- wait(), notify(), notifyAll() can only be call in synchronized method
- non-syn method run time exception, if we use these method.
- suppose, if non-daemon thread is close, then all daemon thread will also close - for running daemon thread

→ only one non-demon should run for running of all demon thread will stop.



19 → Methods

23 - method overloading

Da
2510

*

}

==

⇒

so

eo

o

2023
25/05/2023

File Handling

* package com.jspiders.filehandling;
import java.io.File;

public class Run1 {

 public static void main(String[] args)

}

 File f = new File("D:\\sample");

^{It is class available in io package}

 so import

 S.O.P(f.exists()); // false so, have to pass some path

 f.mkdirs(); // create directory or folder

 S.O.P(f.exists()); // true

 }

 f.delete(); // delete the folder

 S.O.P(f.exists()); // false

D.P.

false

true

false

used to create a single directory

* mkdir() → both return boolean value, we can
mkdirs() → call it also
→ used to create multiple directory

⇒ file f = new File("D:\\Folder1\\Folder2\\Folder3");
f.mkdirs();

f.listFiles() → It returns array of String.

file object : array of file

↳ i.e. return instances of each file in array of file

f.getName() → getName method give you the current file name

LSD

```

File[] f = new File("D:\Java\Basic\src");
File[] file = f.listFiles();
for (File x : files)
{
    System.out.println(x.getName());
}
    ↑
    to print current filename
    in array of file.

```

Assignment

→ Write a Java program to delete only the file which name starts from A, in a particular folder.

→ I/O Exception → i.e. checked exception, <sup>it is also class, present in
java.io.IOException package</sup> _{import}

```

public class Rent
{
    public void main (String [] args)
    {
        File f = new File ("D:\test1.txt");
        System.out.println(f.exists()); // false
        try
        {
            f.createFile();
            f.createNewFile();
        }
        catch (IOException e)
        {
            e.printStackTrace();
        }
        System.out.println(f.exists());
    }
}

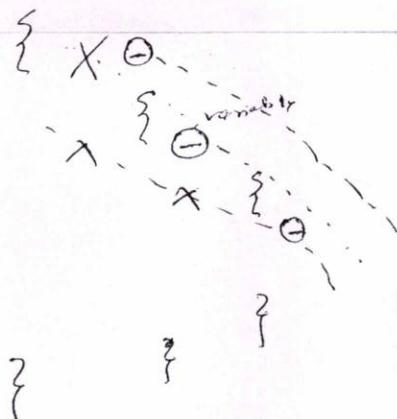
```

I/O Exception → i.e. checked exception, <sup>it is also class, present in
java.io.IOException package</sup> _{import}

f.createNewFile()
generate checked exception: java.io.IOException
defines a class IOException, where it throw a exception using throws so, we have to handle here. It is checked exception
to use createNewFile() we have import package also in package

filewriter — to write in a file.

file reader — to read from file.



In try block also, variable
is local to try block so for
inner block if available, but for outer
block it is not available

- Filewriter also throws exception, since it is checked exception, so, we have handled mandatory by me.
- when we use fw.write → It write the string and return in Output Stream. Java can write each character or characterwise in file.

public class Rento

{ PSVM(--)

} Filewriter fw; →

try {

[fw = new Filewriter(new File("D:\\test1.txt"));
fw.write("gaur");
fw.close();] → All three methods return IOException so, it's checked exception so, we have to handle it mandatory.

} catch (IOException e) {

e.printStackTrace();

Read operation

```
public class Read {
```

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

```
        FileReader fr = null;  
        char[] arr = new char[4];
```

```
        try {
```

```
            fr = new FileReader(new File("D:\\test1.txt"));  
            fr.read(arr);
```

```
        } catch (Exception e) {
```

```
            e.printStackTrace();
```

```
        for (char x : arr)
```

```
        {  
            System.out.print(x);  
        }
```

* f.length() → return long type → to store in int, downcast it
it returns no. of characters present in file

```
import java.io.FileNotFoundException;  
import java.io.FileReader;  
import java.io.File;  
import java.io.IOException;
```

```
public class Read
```

```
{  
    public void
```

```
        FileReader fr = null; local variable for
```

```
        File f = new File("D:\\test1.txt");
```

```

try {
    fr = new FileReader(f);
} catch (FileNotFoundException e) {
    System.out.println("exception caught");
}

```

```

int n = (int) f.length();
char[] arr = new char[n];
try {
    fr.read(arr);
} catch (IOException e) {
    e.printStackTrace();
}

for (char x : arr)
{
    System.out.println(x);
}
}

```

All FileWriter &
FileReader, are
character based operations.
It processed on each
character

→ file reader, we have to
pass, empty character
array.

So, problem so,
BufferedReader, BufferedWriter

→ Internally BufferedWriter & BufferedReader ~~use~~ internally
need FileWriter & FileReader.

public class RunT

```

public class RunT
{
    public static void main (String [] args) throws IOException {

```

File f = null;

FileWriter fw = null;

BufferedWriter bw = null;

f = new File("DII/test2.txt");

fw = new FileWriter(f);

bw = new BufferedWriter(fw);

bw.write("java");

bw.close();

FileWriter

152

if we don't add
try catch block, then
we can ~~still~~ throws an
error, but it is not
standard practice.

bw require fw and fw
requires file always

BufferedReader

```
public class RMS
```

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

```
        File f = null;
```

```
        FileReader fr = null;
```

```
        BufferedReader br = null;
```

```
        f = new File("D:\\test2.txt");
```

```
        fr = new FileReader(f);
```

```
        br = new BufferedReader(fr);
```

```
        System.out.println(br.readLine());
```

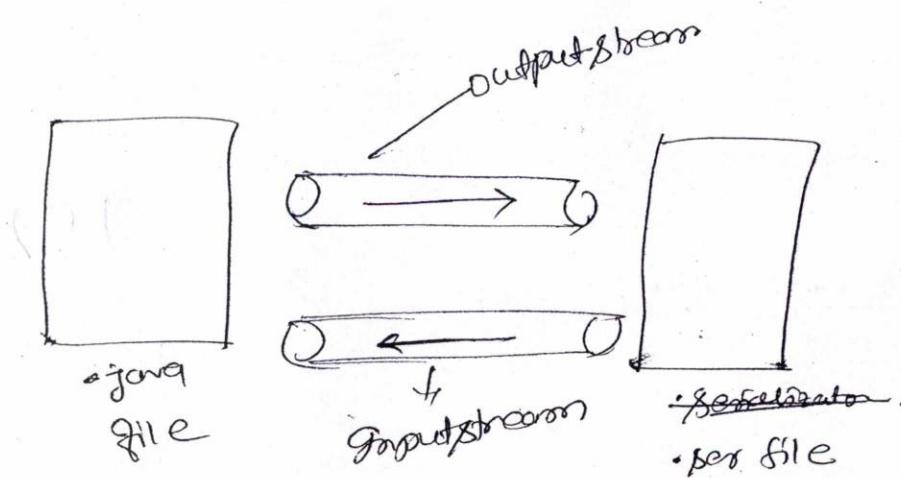
```
        System.out.println(br.readLine());
```

```
        br.close();
```

```
        fr.close();
```

=====

Serialization & de-serialization



add, update
don't do
final field
serializable

public
{
 public
}

If we not
write file
then get
will throw
error that
instance
cannot
be created

Serialization

- writing an instance into a file, is known as Serialization. — we need OutputStream here.
- creating an instance from the file is known as deserialization. — we need InputStream here

```
public class Emp implements Serializable
{
    String name;
    transient int sal;
    Emp(String s, int i)
    {
        name = s;
        sal = i;
    }
}
```

~~add, we don't want transfield serialized~~

↑
Serializable is a empty interface. There is no method. It is marker interface.

~~Object~~

writeObject() method only take a class of Serializable otherwise throw exception

```
public class Run
```

p.s. run () throws IOException {

emp e1 = new Emp("abc", 3000);

File f = new File("D:\\test3.ser");

* FileOutputStream os = new (FileOutputStream(f));

* ObjectOutputStream objos = new ObjectOutputStream(os);

If we not write file then it will throw error that instance cannot be created

objos.writeObject(e1);

objos.close();

objos = new ObjectOutputSteam();

153

```
public class Run10
{
    public void(--) throws Exception {
        File f = new File("D:\test8.ser");
        FileInputStream OS = new FileInputStream(f);
        ObjectInputStream ObjOS = new ObjectInputStream(OS);
    }
}
```

* Emp e1 = (Emp)ObjOS.readObject();

S-O-P(e1.name); }
S-O-P(e1.sal); }

}

points

* we can serialize only the object of type Serializable.

- Serializable is an interface which is available in java.io package.
- Serializable interface does not contain any methods.
- An interface with no methods is called marker interface.

transient

- `gt` is a keyword, which is used for non-
serializable fields.
- If we don't want any field serializable, then
make it transient. And When we deserialized
that variable, `gt` return, null or zero.

154

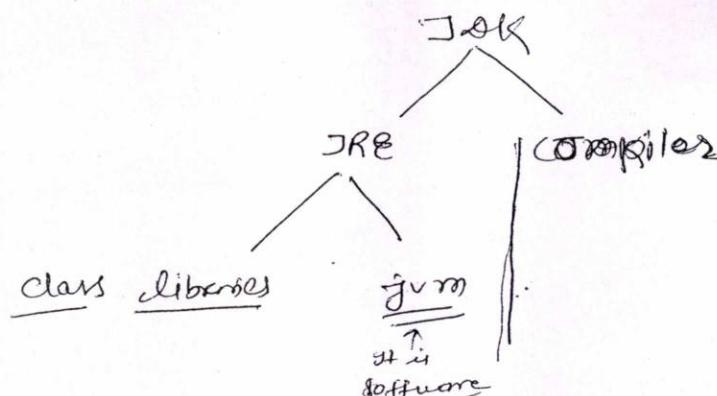
~~abst~~
27/05/07



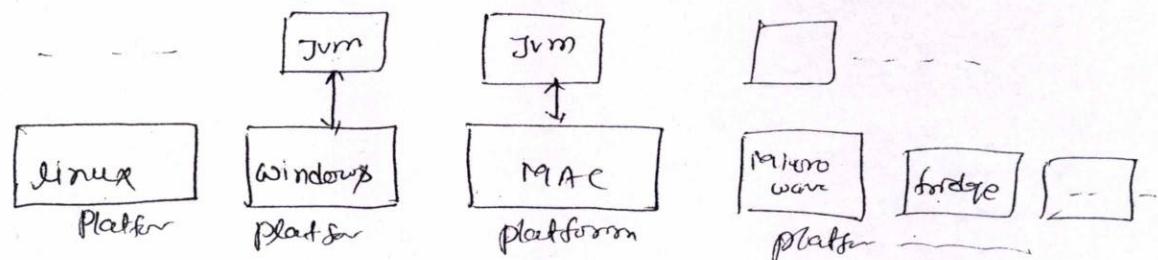
Date
27/05/2013

J2EE

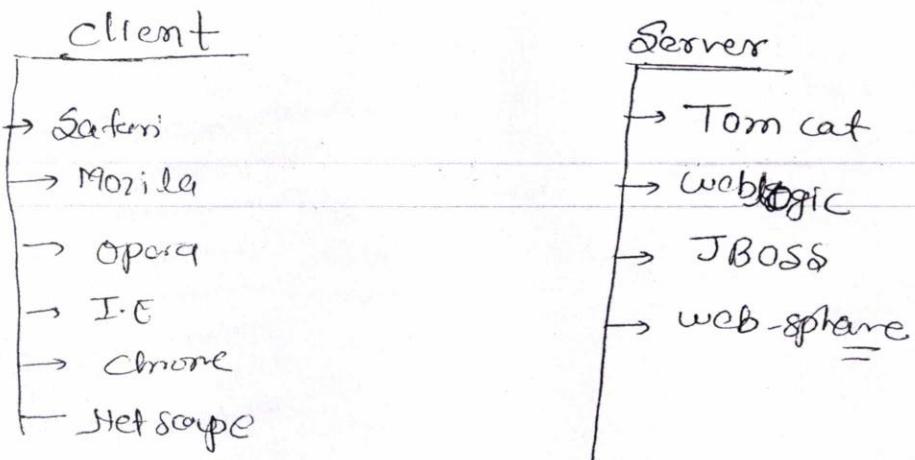
Java - version using → 1.7
best in industry → use 1.6 version



- website is static but web-application is dynamic.



- jvm understand .class file or bytecode that change in machine code.
- Each platform have its separate JDK and jvm. jvm is platform dependent. we can not take jvm from one platform and install at other platform.
- But Java is completely platform independent. we can take .class file from one platform and can execute at any platform because .class contain bytecode, and jvm can understand bytecode.
- jvm using (JIT) just in time compiler to change byte code into its native platform.



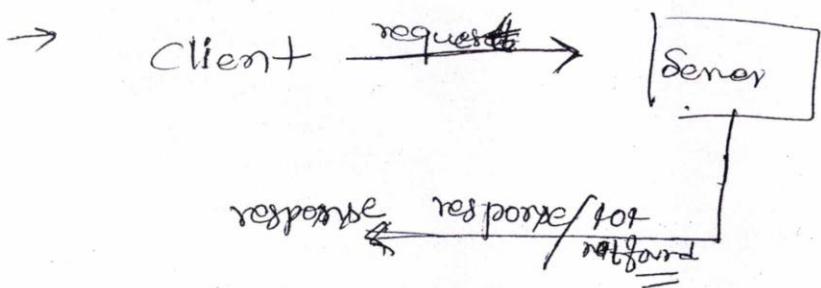
Client is mostly

Browsers

to show IP → IP config.

→ Tomcat Server I.O download zip version

→ IDE → Eclipse - Indigo - above 3.6 use
 → Jdk - 1.6 download 3.7, - -



→ Client & server understand html language

→ using http protocol communication is done b/w client & server.

→ HTTP — Hypertext transfer protocol.

→ HTTP built upon TCP/IP.
 ↳ make file transfer over internet

to see which version is working

java version.

- Apache HTTP Server → It is used for website, ie static
- Apache Tomcat → download . war for web application ie dynamic.

JKW
28/05/2013

To set the path

JAVA_HOME

New System variable — CATALINA_HOME

PATH — where the server is computer.

PATH — " . . . \bin;

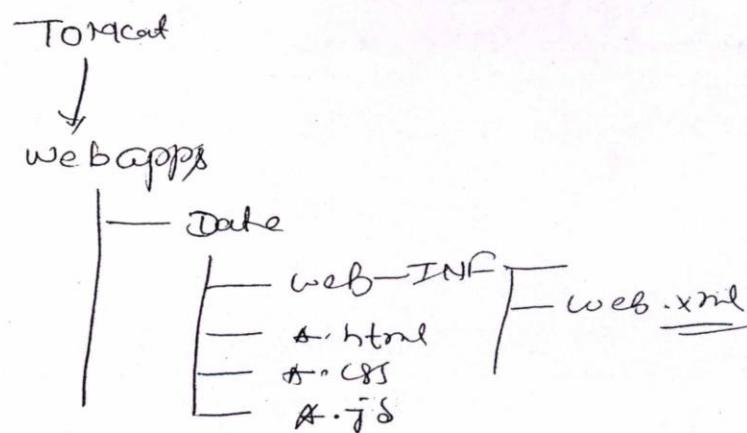
HTTP — responsible to transfer data on internet.
it takes some data from client &
give to server. it have request
& response sequences.

** Servlet-api is used to create dynamic web application.

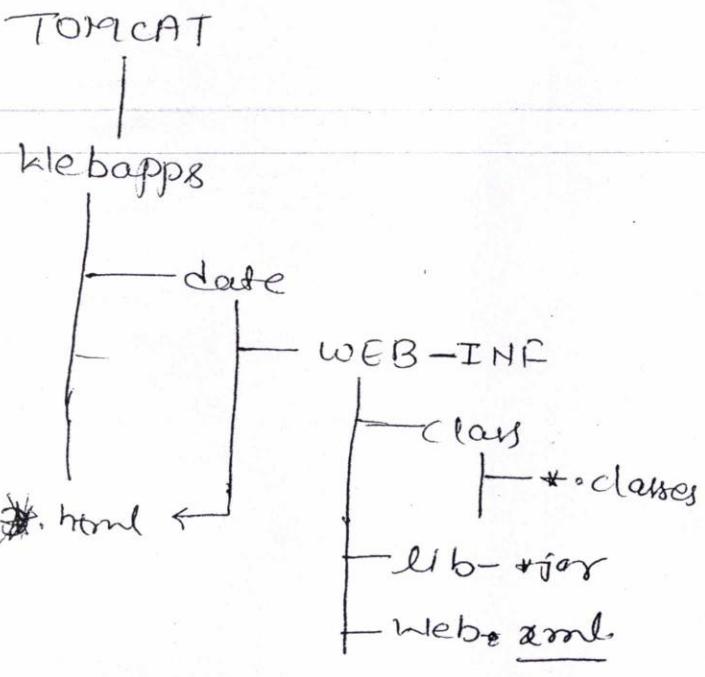
* Download Eclipse — JDK version (not STK version)
Should be used to create dynamic web application.

* web.xml — This is used to tell the server the configuration of our application. By default it contains index.html if we want to change it

[http://localhost:8080/date] — in I-E



directory structure



Note:-

- - i) create Dynamic project in Eclipse.
 - ii) write an html inside content.

```
<body>
<h1> This is Date Appln from jang <h1>
<h3> <a href = "Date.do" > click for date </a>
</body> </h3>
```

- iii) copy ~~server-apis.jar~~ from tomcat lib folder & paste it inside WEB-INF lib folder.

- (5) write a java servlet program

```
package com.jspy.date;  
import java.io.IOException;  
import java.io.Serializable;  
import java.util.Date;
```

```

import javax.servlet.ServletException;
        ,, , , , http.HttpServlet,
        ,, , , , HttpServletRequest,
        ,, , , , HttpServletResponse;

public class DateServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest request,
                         HttpServletResponse response) throws
    ServletException, IOException {
        response.setContentType("text/html");
        Date date = new Date();
        PrintWriter out = response.getWriter();
        out.println("<html><body><h3>" + date + "</h3></body>
                    </html>");
    }
}

```

This is used to write on page of browser

- ⑤ web.xml is used to give description of an application to the server.

web.xml

```

<welcome-file-list>
<welcome-file> welcome.html </welcome-file>
</welcome-file-list>

```

```
<servlet>
```

```
< servlet-name> mydate </servlet-name>
```

```
< servlet-class> com.jspy.date.DateServlet </servlet-class>
```

```
</servlet>
```

< servlet-mapping >

< servlet-name > mydate < /servlet-name >

< url-pattern > /date.do < /url-pattern >
< /servlet-mapping >

< /web-app >

Steps to create dynamic web project

- (i) Start the eclipse & goto file → new
- (ii) Dynamic web project — give project name → select dynamic module ✓

- (iii) Now go to file, → create a html file (welcome.html) & save it.

- (iv) Now click on project date, & click on webcontent & then WEB-INF, → now configure web.xml file.

- (v) copy the servlet-api.jar from server under lib folder & copy it at WEB-INF/eclipse

- (vi) create a java file (DateServlet.java)
~~button~~

- (vii) create a folder date in the server under apps folder.

- (viii) in date folder, copy the welcome.html & WEB-INF.

- (ix) in WEB-INF folder, copy the .class file of DateServlet.java & web.xml file

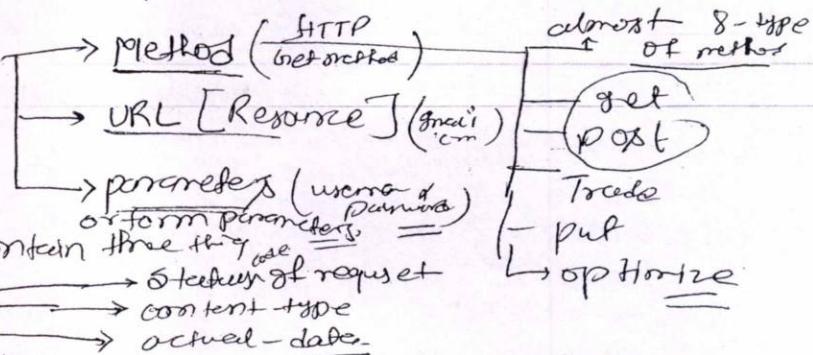
- (x) Start the server.

- (xi) Start T.S & type localhost:8080/date

Date
29/05/2013

→ when a client request for something then

using HTTP



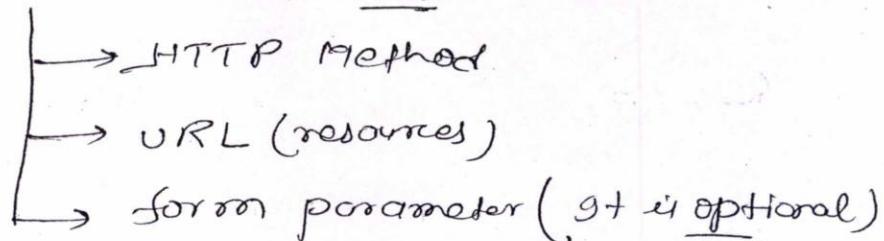
server send
HTTP response contain three thing case =

→ There are 7-different HTTP methods.

- i. Get
- ii. post
- iii. put
- iv. Trace
- v. Head
- vi. option
- vii. Delete

{ In Google
Search
web-browser
alternative
see
ReX - Search HTTP server
viewer

* HTTP request is formatted by the browser and with it, ~~it~~ throws three (3) main info. to the server.

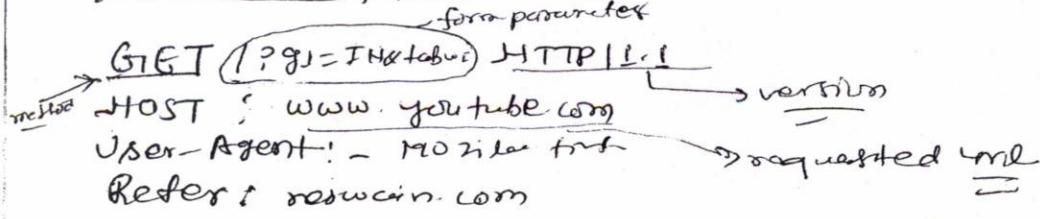


→ always ~~use~~ use a get method

→ Suppose enter a url.

www.youtube.com/?gj=IN & tab w1

Sending Request Contains



HTTP Response :-

→ Once the response reaches to the server, it sends response ~~headers~~ back to the Browser.

→ It contains mainly 3 things.

- (1) Status code
- (2) Content type
- (3) Actual content

→ Get method sends the data through URL to the server so, it is not secured. Instead of we, post method it is secured as it sends data on hidden form and changes it in some other form.

Q. What is difference between GET and POST?

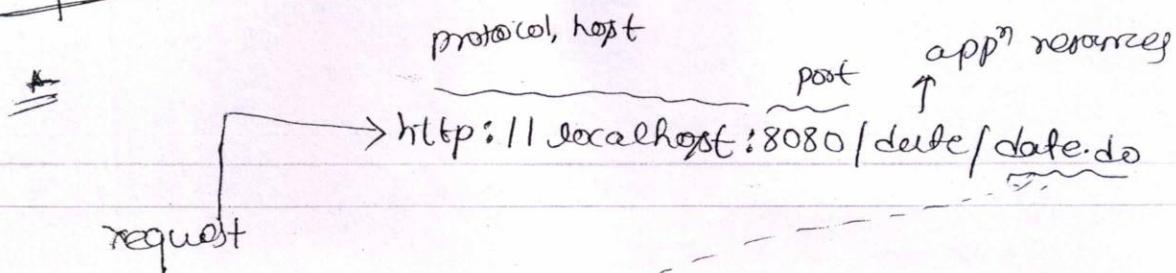
Get

- Data goes with URL
- Not secured
- Max 512 characters
depends on server memory
- Book marks

Post

- Data goes with body only
- Secured
- Data sent can be unlimited
- Can not bookmark
- Data does not goes with URL

30/05/2022



description
of application

root element is `<web-app>`

* `web.xml` — deployment description and configuration setting of my application

* `index.html` — default welcome page in tomcat

→ If that resource called `date.do` and it's not found then it will give 404 error

→ If there is `date.do` instead of do, it can be anything like car, bus

html

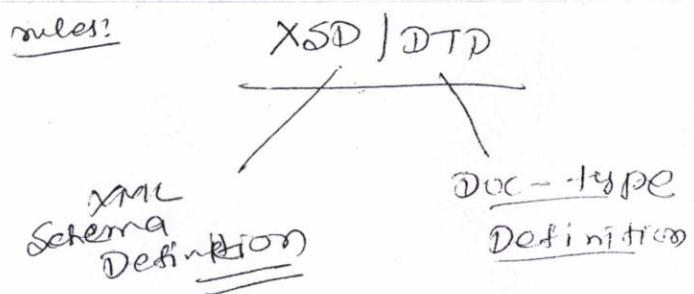
- hypertext markup language → root element is `<html>`
- predefined tags
- provide look & feel to the user
- Browser understand the language
- provide texting.
- It is less strict (not strict about closing tags)

XML

- It is more strict (must have to close `</tags>`) & follow rule
- Extensible markup language = XML for custom tags
- It has own tags.
- Storing & transporting the data
- It allows to user to define own tags or custom defined
- Using XML, two different application written in different language can communicate to each other

`<web-app>`
is not element

→ Although XML ~~is~~ can define custom defined tags
but to define a new tag up to some
standard rules, because gener can understand



XML

- Extensible Markup language
- XML allows user defined tags
- XML is used to store and transport small amount of data.
- XML is strict.
- XML gives description of data being stored.

~~*~~ Servlet

- Servlet is a Java program that runs on a server.



→ click on web content → html page

MessageServlet.java

```
public class MessageServlet extends HttpServlet {
```

① override

```
protected void doGet( -- -- )
```

```
resp.setContentType("text/html");
```

```
PrintWriter out = resp.getWriter();
```

```
String messageByUser = req.getParameter("message");
```

```
out.println("<html><body><h1>" + messageByUser
```

"</h1></body></html>");

② override

```
protected void doPost( --- )
```

→ message.html

```
<body>
```

```
<h1> This is message page </h1>
```

```
<form action="display.do" method="post">
```

If we don't write, then by default
it is get, it send data with URL.
but wrong POST, it
is occurred &
does not send data
with URL.

```
Enter message: <input type="text" name="message"/>
```

```
<input type="submit" name="enter" value="enter"/>
```

```
</form>
```

```
</body>
```

web.xml

<servlet>

<servlet-name> message </servlet-name>

<servlet-class> com.jsp.date.MessageServlet

</servlet>

<servlet-class>

<servlet-mapping>

<servlet-name> message </servlet-name>

<url-pattern>/display.do </url-pattern>

</servlet-mapping>

</web-apps>

====

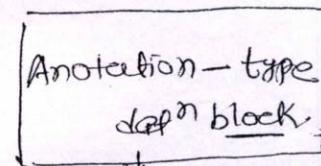
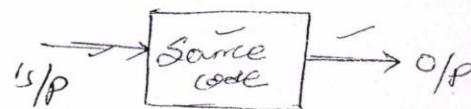
Date
28/05/13

Junit

→ Here if fewer class is called as Business class.

→ variable → called unit member

→ method, constructor → called Business.



- like
@test
@ignore
@after
@before
@before class
@after class

* for Junit

Download junit 4.11 from Google.

eclipse → right click source → Build path → Configure Build path

→ Libraries → Add external jars → Browse file path & open.

→ eclipse → source → new → junit test case.

↳ If there is no need to download

* Create a package → from open a junit test case file

Annotate parent in Junit

- @test
- @ignore
- @after
- @before
- @before class
- @after class

→ Junit is an API, used for white-Box testing.

→ Junit also known as white-Box testing framework.

Annotiations

→ An Annotation, it is a type defn block, which is used to implement the predefined meaning of operations.

```
package org.junit;
import java.util.*;
import org.junit.Test;
public class Sample{
```

if we any annotation
we must import that
Annotation

① Test

```
public void test1()
```

```
{  
    System.out.println("running test1 method");  
}
```

② Test

```
public void test2()
```

```
{  
    System.out.println("running test2 method");  
}
```

O/P
running test2 method
running test1 method

Junit provides matrices also about file

Pass	Error	Failure
2/2		FO Failure

it give output randomly
it does not give sequential
O/P

import static org.junit.Assert.*;

import org.junit.After;

" " • Before;

" " " • Ignore;

" " " • Test;

" " " • BeforeClass;

" " " • AfterClass;

public class Sample {

① Before

public void beforeTest()

{

S.O.P("running before Method");

}

② After

public void AfterMethod()

{

S.O.P("running after Method");

}

③ Test

public void Test()

{

S.O.P("running Test Method");

④ Test

public void Test2()

{

S.O.P("running Test2 Method");

⑤ Test

public void Test3()

{

S.O.P("running Test3 method");

Standard

class name is
Sample

then method name like

SampleTest

~~Sample~~

Method

TestMethod

or ~~TestMethod~~

① Test

```
public void Test() {  
    System.out.println("running test");  
}
```

② Afterclass

```
public static void TestAfterClass() {  
    System.out.println("running afterclass method");  
}
```

if we don't put static with Afterclass or beforeclass then in Junit report it give error.

③ Beforeclass

```
public static void TestBeforeClass() {  
    System.out.println("running beforeclass method");  
}
```

Output

- running TestBeforeClass method → only one running
{ running before method
{ running test4 method
{ running after method → running each time with method
{ running before method
{ running test3 method
{ running after method
{ running before method
{ running test2 method
{ running after method
{ running before method
{ running test1 method
{ running after method
running TestAfterClass method → only once running

- A junit class is java class, which is made up of annotations.
- we use `@Test` before method to make sure all the test methods runs randomly.
- when a junit class is run, gt gives the reports, i.e known as (junit report) which consist of how many numbers of test cases pass & how many test case failure & reason for failure.
- `@Ignore` annotation will be use if at all any ~~test~~ test method is to be skipped.
- `@Before` annotation will be use, gt runbefore each & every test method.
- `@After` annotation runs ~~for~~ each & every Test method.
- `@AfterClass` runs only once for the entire junit test class.
- `@BeforeClass` runs for entire junit class, before junit starts running.
- we use `@BeforeClass` to define, to any pre-condition required for the junit class to run.
- we use `@AfterClass` to define, to any post condition required for the junit class to run.

→ To define methods in junit class
We have to follow below rules.

- * The test method ~~will be static~~ should be public & ~~non~~ static and have void return type.
- * It should not have any arguments.

Syntax

```
public void MethodName()  
{  
}
```

- * `@afterclass` & `@beforeclass` should be declare the static keyword

How to execute a Business class or java class

first create a java class

```
package com.jspiders.frames;
```

```
public class MethodOp {  
    public void m(String[] args)  
{
```

```
        public static int addition(int a, int b)
```

```
    {  
        int c = a + b;
```

```
        return c;
```

```
}
```

```
        public static int mul (int a, int b)
```

```
    {  
        int c = a * b;
```

```
public static int sub(int a, int b)
{
    int c = a - b;
    return c;
}
```

Now for testing method, create a JUnit class

```
package org.guru;
import static org.junit.Assert.*;
public class MathoppTest()
{
    static int a; static int b;
    static Mathopp math;
    @BeforeClass
    public static void BeforeClass() {
        a = 20;
        b = 30;
    }
}
```

@Test

```
public void testAddition()
{
    int actual = math.addition(a, b);
    int expres = 50;
    Assert.assertEquals(actual, expres);
}
```

class
ancestor
in JUnit
for files

method
that static
method

O/P test case Pass

If we want test case fail then pass the wrong
expected value 55

④ Test

```
public void Testmul(  
    {  
        int actual = math.mul(a,b)  
        int expected = 600;  
        Assert.assertEquals(actual, expected);  
    }
```

JIR → in Junit report
test case Pass

Here we can write Test case for each business method & can see whether it pass or fail & why.

- * Assert is a class provided Junit for file.
- * assertEquals is a static method present inside Assert class, which returns boolean value.
- * assertEquals method, which update Junit report based on result of Assertion

