1

DATE

Core Java : (3 to 4 weeks

0806 6

1. introducrion:
2. Java language:
3. oops in Java
4. packages
5. Exceprion Handling
6. Threads
7. Java.lang package
8. Java.util package
9. IOstreams

Ser classpath

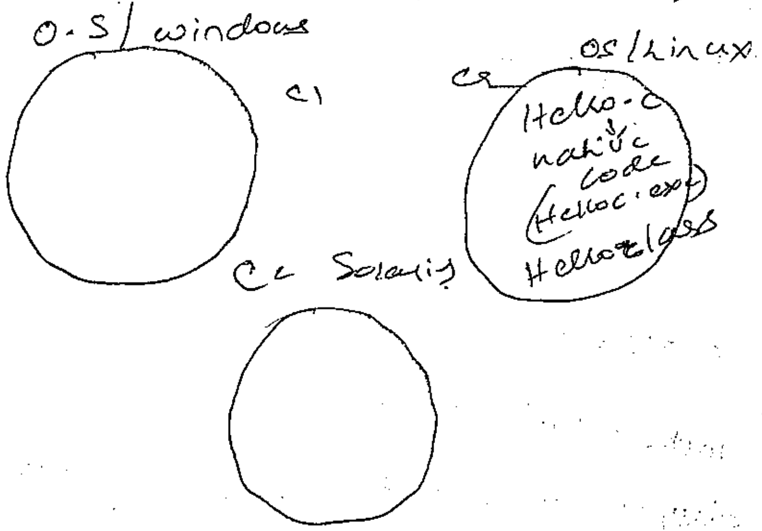
```
D:\> set Path=%Path%; C:\J2SdK1.5\bin;
D:\> set classpath=%classpath%; C:\J2SdK1.5\lib;
D:\> set classpath=%classpath%;.; ↵
```

war do u know about Java?

Java;

- → O.O.P.L
- → Used in Internet appl
- → platform Independent
- → write once, run anywhere.

↳ war is platform independent ↘ (means O.S)



Hello.c

Sourcecode
↓ compilation.
↓ ~~exe~~?
Hello.exe
↓ contains
m/c code

? JVM platform dependent or not
↓ dependent ✓

Hello.Java (Sourcecode)
↓ compilati.
Hello.class - Bytecode (Intermediate code)
↓ run
m/c | Native code
(diff. native code for diff. O.S)

Sourcecode $\xrightarrow{JVM}$ Bytecode

Java is platform independent
Java is JVM dependent

Introduction:

1. Java is a object oriented programming lang.
2. Java is a platform (because ~~Java~~ of JVM)

3. Java is o.s independent.
4. Java is Jvm dependent.

Q.) why Java is o.s independent?

In non Java prog' lang' like c, cpp etc, when u compile the source code, It generates native code directly which can be understood by same o.s. we can't used this native code in any other o.s directly.

In Java when u compile source code, it generates .class file which contains a intermediate code called byte code

SivaPrasad
Icons

This byte code is interpreted by Jvm only, not by the o.s. When u run this byte code, first byte code will be converted to ~~byte~~ native code & then it will be executed by Jvm.

Here we are depending on Jvm for all the things not on o.s, because of this we can say Java ~~o~~ is o.s independent & Jvm dependent.

Java language:

- * character set
- * keywords
- * Indentifier
- * data types
- * variables
- * constants
- * literals
- * operators
- * control stmts
- * arrays.

character set:

- * digits (0-9)
- * alphabets (a-z, A-Z)
- * Special symbols (_ underscore, $ dollar, Remaining all lang special symbols)

Keywords:

* Keywords are also called as a reserved words, which have predefined meaning

if
else
switch
for
while
do
goto
continue
default

static
final
abstract
private
public
protected
Transient
Volatile
Synchronized

class
interface
extends
implements
this
super
int
float

byte
char
short
long
double
boolean
break
try
catch
finally
void

package
import
const ✗
new
native
instance
Throw
Throws
assert
case
return

Total - 49

[goto const] not in use.

Identifier: is a name which we can use for classes, interfaces, variables, methods etc.

Rules for identifiers:

1. We can use all the digits & alphab
2. we can use _ & $ in special symbo
3. first character must be a letter o
   _ (underscore) or $
4. Keywords are not used as identifi

Eg: abc123 ✓, 123abc ✗, _ab19 ✓, $123abc ✓, abc 12 ✗

Data Types:

data type	size (bytes)	initial (or) default value
byte	1	0
short	2	0
int	4	0
long	8	0
float	4	0.0f
double	8	0.0
char	2	ϕ
boolean	–	flase

Variables: is an identifie which is used to store some values.

(or)

is a name used to rep' a mem' loc where the value is stored.

declaraing variables: ② defining a variabl

datatype var1, var2, var3, ... ;

Eg: int x, y ; $\overset{x}{\boxed{0}}$ $\overset{y}{\boxed{0}}$

double a, b; $\overset{a}{\boxed{0.0}}$

c. $\overset{c}{}$

Constants: In C & CPP, we are declaring the const^n as follows.

const int x=10;

But in Java, the keyword const is not allowed, we use the following way $\boxed{\text{final int a =100;}}$

const^n are also called as final variables

Literals: we have 4 types of literals in Java

1. Integer literals
2. floating literals
3. Character literals
4. String literals.

1. Integer literals: 3 Types
   a. decimal integer literal
   b. octal integer literal
   c. hexadecimal Integer literals.

D.I — 10 — 0-9 — 123

O.I — 8 — 0-7 — 0123

H.D.I — 16 — 0-9 + A-F — 0X123
(zero)

x = 12AB ✗ } Errors
x = 0192 ✗

2. floating pt literal:

We can repl floating pt literals in 2 ways.

1. decimal notation.
2. ~~[illegible]~~ exponentational notation.

double x = 10.5; ✓
float x = 11.56; ✗
float x = 11.56f; ✓
} decimal notation

To repl 5.6×10^{-19}

double x = 5.6 E-19 } exponential notation
x = 5.6 e-19

3. character literals:

A single character enclosed bet^n single quotation marks ~~is~~ called character literals.

Eg: 'a', '$' ✓
'1', ' ' ✓

'ab' ✗

4. String literal:

set of characters enclosed bet^n double quotation marks (" ") is literal

Eg: "vas" ✓
"a" ✓
"" ✓
" " ✓

"abc123+-()[]{}"$#" ✗
"abc123+-()[]{}\"$#" ✓
treats single character

Escape Sequences: (2 characters treated as single character)

\" \'
\b \\
\t
\n
\a

Operators:

1. arthemetic operators (0 0 6 : 6
2. Relational operators
3. assignment operators (=, +=, -=, *=, /=, %=)
4. Logical operators (&&, ||, !)
5. Unary Operators (++, --)
6. Ternary operators (?:)
7. Bitwise operators (&, |, ^, >>, <<)

// operators

```
class ODemo
{ public static void main(String as[])
  {
  }
}
```

b=a++ ;	b=++a

b=a ;
a=a+1 ;
	a=a+1 ;
	b=a ;

Ternary operator:

syntax:

(condition) ? trueblock : false block;

(or)

varname = (condition) ? True block : false block;

Eg: (a>b) ? s.o.p(a) : s.o.p(b)

max = (a>b) ? a : b;

a c ~~<((a>b)?a:b)~~ : c

arthematic exp : a+b, a/b

relatn exp : a>b, a<=c

logical exp's is expl which is used to combine 2 or more relatn expl. Result of logical expl will be decided using follow Tables.

A	B	A&&B	A||C
T	T	T	T
T	F	F	T
F	T	F	T
F	F	F	F

Bitwise operators:

& (bitwise and):

Eg: 14 & 5

128 64 32 16 8 4 2 1

0 0 0 0 1 1 1 0 — 14

0 0 0 0 0 1 0 1 — 5

&

0 0 0 0 0 1 0 0 — 04 Ans

0 0 0 0 1 1 1 0 14

0 0 0 0 0 1 0 1 | (or) 5

0 0 0 0 1 1 1 1

15

Bitwise And (&) & Bitwise OR (|)

Bitwise XOR (~~⊕~~,)(^)

↳ Both same - false

0 0 0 0 1 1 1 0 14

0 0 0 0 0 1 0 1 (^) 5(^)

1 0 1 1 11

>> - right shift (half the value)

<< - left shift (double ~~the~~ value)

P = 14 q = 5

P >> 2 = 0 0 0 0 1 1 1 0 >> 2

0 0 0 0 0 0 1 1 = 3

P << 2 = 0 0 0 0 1 1 1 0 << 2

2

0 0 1 1 1 0 0 0

32 16 8 = 56

right shift(>>) — half the value for each shift (integer value)

left shift(<<) — double the value for each shift.

control stmt:

1. if-else
2. switch
3. for
4. while
5. continue
6. break
7. goto.
8. do-while.

① if-else → exp' (logical relational)

```
if (condition)
     T (or) F
{
     // True block / if block


}
else
  {  // false blk / else block
```

② switch (Expression)

```
{
case val1: st1; break;
case val2: st2; break;
      :
case defaults: stmt;

}
```

③

```
      1              2 → T / F       4
for ( initializatn; condn; inc/dec)
  {
      // stmt        3

  }
```

while:

```
   initialization;
   while (condition)
   {
      ______
      ______
      ______
      inc|dec;

   }
```

amanacable no.

assignment

① write a program find a minimum of 4 numbers?

a ~~=10, b=11, c=~~

② w.a.p To print odd no's from 1 to 100

③ w.a.p To find whether the given no. is prime or not.

④ w.a.p To print reverse of the given no.

⑤ w.a.p To find the given no. is amstrong (or) not?

123, 1³+2³+3³ = 123 then

⑥ w.a.p To find whether the given no. is palyndrome.

⑦ w.a.p to print the factorial of a given no.?

8) w.a.p To Print the fibonacci series of upto given no.?

9) w.a.p To find the given no. is perfect no. (or) not?
(Sum of divisibles of given no. is equal to the given no.; Then it's perfect) ;

(10) w.a.p To evaluate the following exp's

$$1+\frac{x}{1!}+\frac{x^2}{2!}+\frac{x^3}{3!}+\frac{x^4}{4!}+\cdots$$

(i/p)[x, n] no. of terms
no. of

first Java prog':

```
class Hello
{ public static void main (String as[])
  { System.out.println ("welcome to
    Java... from Sdsoft");
  }
}
```

1. Sava the prog' as "Hello.Java"
2. Compile as follows
   E:\b7new Javac Hello.Java
3. Ran Java Hello (prog'name)

OOPS in Java

- perfect no.

```
for (i=1; i<=n/2; i++)
{
    if (n%i == 0)
    s = s+i
    n = n/2;
}
```

6 = 1+2+3 = 6
perfect
28 - perfect.

- while(n!=0) Reverse

```
while(n!=0)
{ r = n%10;
  s = s+r ; (m = m*10+r)
  n = n/10;
}
```

↳ reverse of a given

→ sum of digits.

we have four oops conceps

1. Absraction
2. Encapsulation
3. Inneritance
4. polymorphism

Object: Any thing in this world is an Object
— Grady Booch.

• no two objects are same

Object: properties (or) attributes

operations (or) behaviour

Eg: marker
- height
- Color
- width
- cost
- company

} property

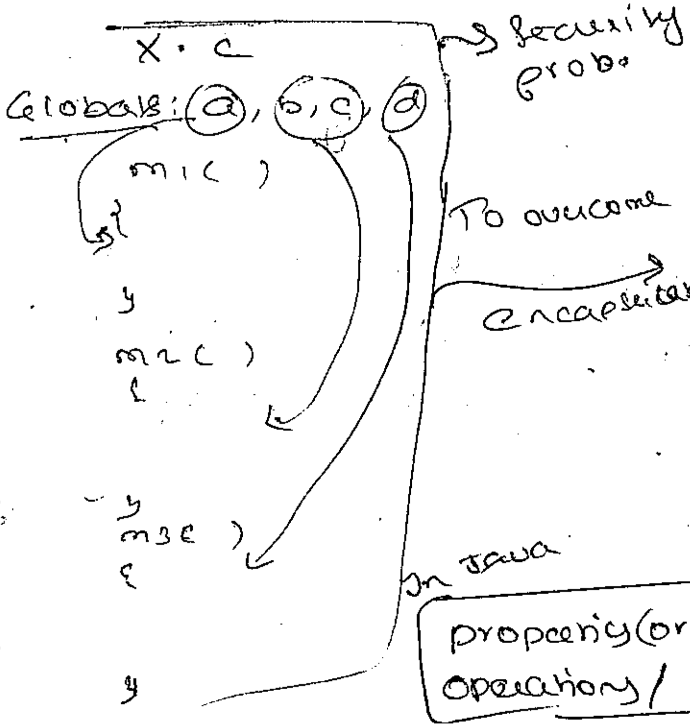
- writing
- Throwing

} operations

Abstraction: providing necessary property & operatⁿ of an obj. by hiding internal details is called an absraction.

2. Encapsulation: [Writing property & operatⁿs that are going to operate on property in single entity is called encapⁿ].

* In Java property are called as variables & operations are called as methods & entity is called as class. i.e writing variables & methods which are going to use variables into a class is called encapsulation.



```
m1( )
{

}
m2( )
{

}
m3( )
{

}
```

```
class c1          class c2()
{                 {
  int a;            int b;
  m1()              m2( )
  {                 {
  }                 }
}                 }

class c3
{ int d;
  m3()
  {
  }
}
```

property(or) attributes / variables
operations / behaviour / mtds

* we ~~e~~can achieve encapsulation by ~~[illegible]~~ privare variables & public methods.

Inheritance: Writing a new ~~[illegible]~~ class by using funtionality of existing class is called as inheritance. Existing class is known as super class or Base class (or) parent class. New class is called sub class (or) derived class (or) child class.

* adv: ~~code~~ reusability.

polymorphism:

Def: One operation, behaving differently in different situat^n is called polymorph i.e one operation will have many implementations.

In Java, we have 2-types of polymorphisms.

1. Compile Time polymorphism
2. Run Time polymorphism

Objects & classes:

class : class is an entity which contain variables & mtds. These 2 are called as members of the class.

Syntax: for class definition

```
Keyword
   class  class-name → Identifier
      datatype var1, var2, ...; } → identifiers
      datatype var1, var2, ...;
      . . . . . .
      . . . . . .                  → Identifier
data type ← returntype  mtd-name (arg)
      {
        //body → valid Java stmts.
      }
}
```

Eg:
```
class student
{ int sno = 99;
  String name = "srinivas";
  String phone = "9999";
  void disp()
  { S.O.P(sno);
    S.O.P(name);
    S.O.P(phone)
  }
}
```

Creating the object:

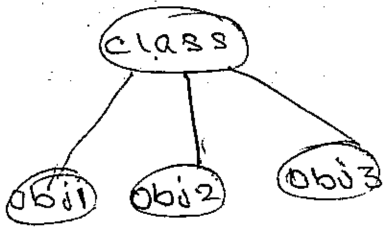
Syntax:

```
class-name objectname = new classname()
```

Eg:
```
student s = new student();
```

Class is a logical entity.

Object is a physical entity.



```
class SDemo
{
public static void main(String as[])
 {
   student obj1=new student();
    obj1.disp();
   student obj2 = new student();
    obj2.disp();
 }
```

["Saving the file name as classname which is having a main method]

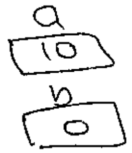
SDemo.Java

```
class student
{ int sno;
  String name;
  String phone;
  void disp()
  { System.out.println(sno);
    System.out.println(sname);
    System.out.println(phone);
  }
}
```

contd...

```
class SDemo
{ public static void main(String ar[])
 { student obj = new Student();
    obj.disp();
   student obj1 = new student();
    obj1.disp();
 }
}
```

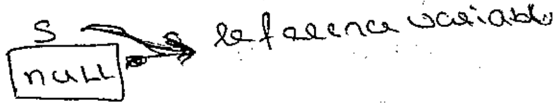
```
int a = 10;
int b;
```



↳ primitive variables

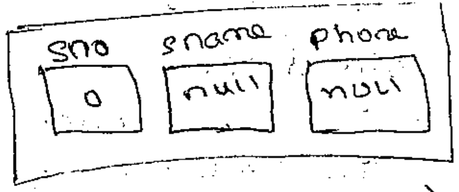
Student S = new student()

1 — Student S ; 3 — = ; 2 — new student()



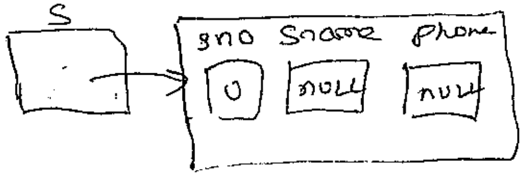
① student S

② new student()



Bcoz sname is a reference variable.

③



- Default value of reference variable is <u>NULL</u>

① Reference variable S will be created and default value null will be assigned

② Allocating mem' for all the variables as a declared inside student class as a block, based on variable Type (primitive, reference)

③ Block address will be assigned to a reference variable S.

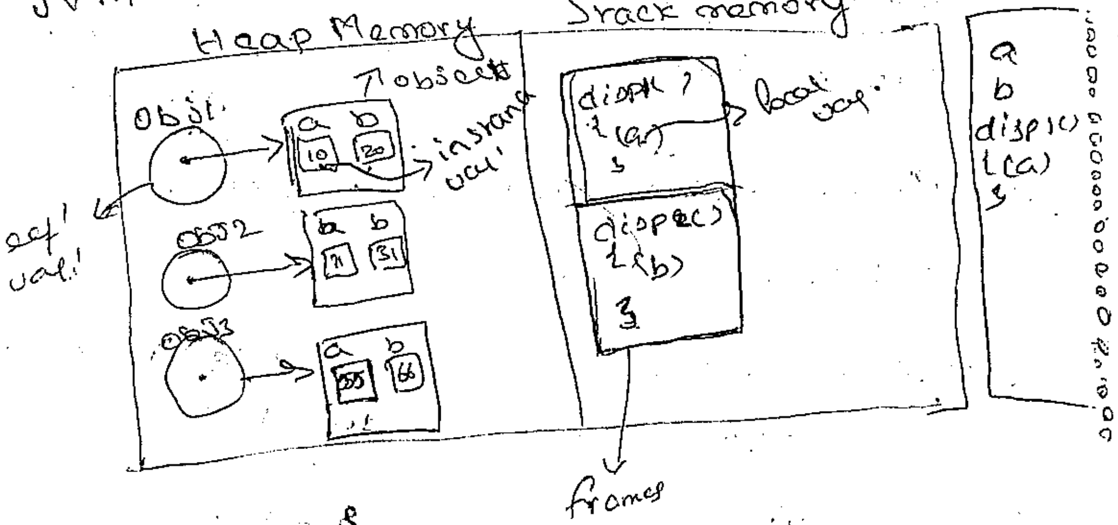
- Default value of primitive var is depends on type of primitive datatype.

primitive var.	reference var
1. variable declared with primitive datatype is called primitive var.	1. var. declared with class type is called refer var.
2. default values for primi var. depends on primitive datatype war we used.	2. Default value of ref var is null for all classes

3. <u>memory size</u> ~~fo~~ of Primitive var. depends on. primitive type var we use.

~~3~~. mem'. Size for <u>ref'</u> var'. is <u>8</u>-bytes (fixed)

JVM



<u>Heap</u>: mem' allocat' for ref' var.
mem' allocat' for object

<u>Stack</u>: method definitions will be stacked in stack frame.

for each new mtd in a class, ~~there~~ will be one ~~fr~~ frame will be allocated in stack.

```
class Hello
{ int a,b;     (They will take default variables)
  void m1()
  { static int a; /
    int c;   (we have to initialize local varia i.e inside mtd)
    S.o.p(a);
    S.o.p(b);
    S.o.p(c); S.o.p(d); -> not allowed
  }
  void m2()
  { int d   (initization must)
    S.o.p(a)
    S.o.p(b)
    S.o.p(c) -> not allowed.
    S.o.p(d)
  }
}
clas MyDemo
{ public static void main(String[])
  { Hello h = new Hello();
    ~~h.m~~ h.m1();
            h.m2();
  }
}
```

Local variables:

- Variables declared inside the mtd are called as local variables.
- scope of the local variables is with in the mtd, where it is declared.
- we have to initialize local variables explicitly; otherwise the following compile time error will come.
  "variable c might not initialized".
- Local variables can be primitive variables & reference variables.
- Mem' will be allocated for loc-var', when ur invoking the mtd.
- JVM allocates the memory for local variables in the stack frame where the definitn is stored.

Instance Variables:

- Variables declared inside the class ^without static^ keyword are called as instance variables.
- ~~scope of the instance variab~~
- Instance var. no need to initialize, i.e when ur not initializing instance var. JVM initializes them with default values.
- scope of the instance var' is with in the class where it is declared, we can use I.var', in all other mem' of that class
- I. var' can be premetive var' & ref 'var.
- Mem' ~~will be allocated~~
- JVM allocates mem' for I. var when ur creating the object.
- JVM allocates mem' for I. var. in the Heap.

Static: is a modifier in Java, which we can use for vars, mtds & for classes (Only for ~~in the~~ inner classes nor for top level classes).

• Members with static keyword are called as static members.

Static variables:

○ variables declared ~~in~~ with the static keyword.

for Eg: static int a;

→ Only one copy of mem', will be allocated for static var', for all the objects. i.e all objects will share same mem' location.

○ Mem' will be allocated for static var. when JVM is loading the class into the mem'.

○ local variables can't be static

○ Static variables belongs to class, so we can call static variables with the class name directly. i.e

class Hell
static int a;
classname.variable name
Hello.a;

Static mtds: Mtds defined with static keyword are called as static mtds.

static mtds belongs to class.

We can invoke static mtds with class name (or) object of the class.

Inside a static mtd we can use static variables & static mtds.

We can't use non static variables and non static mtds directly i.e without objects, but we can use with an object.

Inside non-static mtds we can use static members and non static members.

Eg: class Abc

```
{ int a=10;
  static int b=20;
  void disp1()
  {  m1();
     m2();
     s.o.p(a);
     s.o.p(b);
  }
 static void disp2()
 {  Abc x=new Abc();
    x.m1();
    s.o.p(x.a);
    m2()
    s.o.p(b);
 }
void m1()
 { s.o.p("i am m1()");
 }
static void m2()
  { s.o.p("i am m2()");
  }
}
```

```
class Demo1
 {
  public static void main(String ar[])
  {  Abc.disp2();
     Abc obj = new Abc()
     obj.disp1();
     obj.disp2();
  }
 }
```

→ Non static variable "a" can't be referenced from a static context.

Non static mtd disp1() can't be referenced from a static context.

Static modifier is not allowed for top level classes *

X static class ABC

```
    {

    }
```

:- modifier static not allowed

final variable:

variable declared with final Keyword modifier are called final variable.

final variables are also called as constants.

final modifier is allowed for instance variables, static variables & local variables.

```
class Abc
{
  final int a=10;
   final int c;
 // const int x=99.// illegal start of type
  void displ()
  { final int b=26;
      s.o.p(a);
      s.o.p(b);
      s.o.p(c);
     // a=99;
     // b=99;
     // c=99;
      s.o.p(a);
      s.o.p(b);
      s.o.p(c);
  }
}
class Demo1
{ public static void main(String args[])
  { Abc obj = new Abc();
    obj.displ();
  }
}
```

error: can't assign a value to final variable c;

Method Overloading: Writing more than one mtd with same name by changing parameters is called as mtd overloading.

mtd. overloading belongs to a single class ~~mtd overloading belongs to a single class~~

when ~~overloading belongs to a single class~~

when we are overloading mtds we have to change the parameters by following one of the following rules.

1) No. of parameters
2) order of parameters
3) Type of parameters.

No. of parameters
order of parameters
type of parameters

```
class Abc
{ void m1()
  { s.o.p("m1 with 0");

  }
  void ABC( )
   { s.o.p("hai");
  }
 void m1(int i)
 { s.o.p("m1 with 1i");

 }
 void m1 (double b)
 {
   s.o.p("m1 with 1d");
 }                         9.0      8
 double m1 (double b, int a)
 { s.o.p("m1 with 2di");
     return 9.0;
 }
 int m1(int a, double b)
  { s.o.p("m1 with 2id");
      return 6;
  }
}
```

```
class Demo1
{ public static void main (String as[])
  { Abc Abc = new Abc();
    Abc.m1();
    Abc.m1(9);
    Abc.m1(9.0);
    double y = Abc.m1(9.0, 8);
     int x = Abc.m1(6, 9.0);
      Abc.Abc();
  }
}
```

I am calling a mtd m1 and passing
String reference variable as parameter
and I am modifying String inside the mtd
m1 and I am not returning anything.

Q) can I get the modified value ~~is~~ in. called mtd?

Ans; **NO!**

```
class Abc
{ void m1(String x)
   s.o.p(x);
   x=x+"hai";
   s.o.p(x);
  }
}
class Demo
{ public static void main(String a[])
  { Abc obj = new Abc();
    String y="Hello";
    s.o.p(y);
    obj.m1(y);
    s.o.p(y);
  }
}
```

O/p:

```
Hello
Hello
Hello hai
Hello.
```

Q) I am modifying & returning from the mtd, can I get the modified value in the called mtd.

Yes.

Q) when u have same name for local var. & instance variable. then local variables hides the instance variable. with "This" keyword.

'This' is a reference var. that contains obj. of parent class.

'This' is not allowed to use in static mtd.

```
class Abc
{ static int a=99;
  static int x=89;
  void m1()
  { int a=10;
    int x=98;
    s.o.p(a);
    s.o.p(this.a);
    s.o.p(x);
    s.o.p(this.x);
  }
```

```
class Demo1
{
  public static void main(String args)
  {
      Abc obj = new Abc();
      obj.m1();
  }
}
```

---

native
abstract
synchronized } not allowed, they are for mtds & classes.

static
final
Transient
volatile
private
public
protected
default } allowed for variables (instance)

final
default } only allowed for local variables.

13/06/2006

Constructors:

- Constructor is a special mtd, whose name is same as class name
- Constructors don't contain any return type, even void also
- Constructors will be invoked by the JVM, when we are creating the object.
- Constructors are used to initialize the object with some values.

$$\underbrace{\text{Hello obj}}_{1}\ \underset{4}{\boxed{=}}\ \underbrace{\text{new } \overbrace{\text{Hello()}}^{3}}_{2};$$

① creating ref' variable
② allocate memory for Instance variable
③ constructors will be invoked.
④ assigning the block address to reference variable.

```
class student
{
 int sno;
 String sname;
 long phone;
 ~~Student~~
 String ename;
 void student()
 { s.o.p("hsfhrhdfv");
 }
 student(){
  s.o.p("default constructor");
 }
student(int sno, String sname, Strig email,
                  long phone)
 { this.sno = sno;
   this.sname = sname;
   this.email = email;
   this.~~phone~~ = phone;
 }
void display()
  {   s.o.p(sno);
            (sname);
            (email);
            phone);
```

```
class sdemo
{ public static void main (String args[]),
 { Student obj = new student(99, sri",
       "sri@sd.com", 9999);
   obj.display();
   Student obj1 = new Student(88, "vas",
       "vas@sd.com", 8888);
   obj1.display();
   Student obj2 = new student();
 }
}
```

- when we write a class, without any constructor, JVM inserts default constructor.
- when we write a class with arg' constru, JVM doesn't insert default constructor. we have to write the default constructor explicitly.

access specifiers (or) access modifiers

(or) visibility modifiers :

we have 4 access specifiers.

1. private
2. default
3. protected
4. public

o These specifiers specify the scope

o private members can be accessed, with in the class where they are declared. They are not allowed outside the class.

• Top level classes can be default & public.

o Top level classes can't be private & protected.

o We can use these 4 specifiers for all the members of the class. (including static)

o These 4 specifiers are not allowed for local variables.

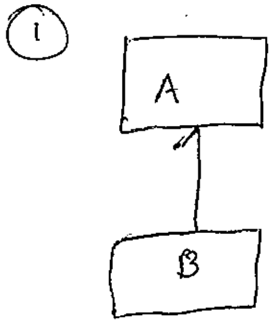
o We can't define local mtds, i.e mtd inside another mtd.

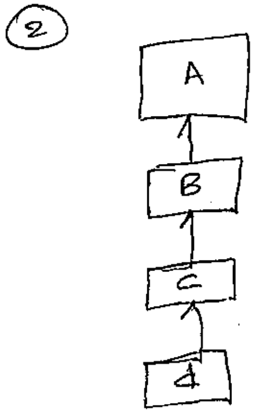
Inheritance:

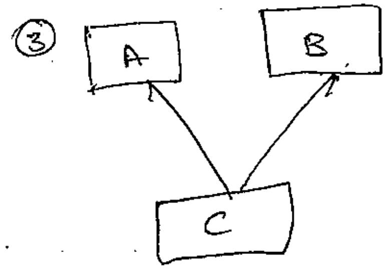
Writing a new class from an existing class is known as inheritance.

following are the different type of inheritance

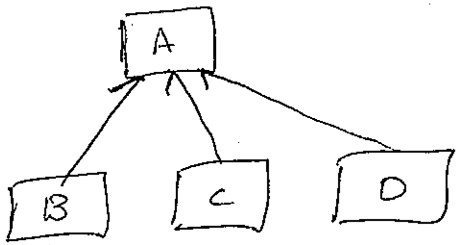
1. Simple inheritance
2. ~~Multiple~~ multilevel Inheritance.
3. multiple Inheritance. ③
4. Hararchical Inheritance. ⟹ Java doesn't allowed.
5. Hybrid Inheritance ⑤



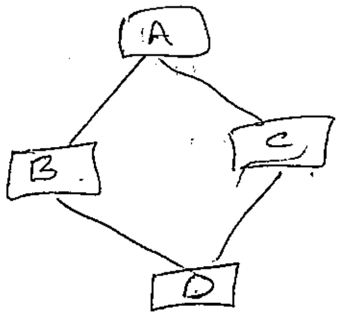




(4)



(5) Comb^n of (3) (4) (5)



Eg:
```
class A
{ int a=10;
  int b=20;
  void m1()
  { s.o.p(a);
    s.o.p(b);
  }
}
class B extends A
{ int c=30
  void m2()
  { s.o.p(c);
```

```
class ABDemo
{
 public static void main(String args[]):
 { ~~int~~
      A  objJ = new B();
      objJ.m1();
      objJ.m2();

   }
}
```

14/06/06

```
Class A
{ int x;
  A()
  { s.o.p("A.defa.con");

  }
  A(int x) super() s.o.p(x);
  { s.o.p("A.arg.con");

  }
}
Class B extends A
{ int y;
   B()
  { ~~super()~~; super();
    s.o.p("B.def.con");
```
(“arg” struck through in the last line, “def” written above)

```
B(int y)
 { super(y);
   s.o.p("1 arg constr");
  }
}
class C extends B
{  int z
  C()
 { super();
   s.o.p("def con");
  }
  C(int z)
 { super(z);
   s.o.p("1 arg constr");
  }
}
class Demo
{public static void main(String ar[]);
 {
   C obj1 = new C();
   C obj4 = new C(99);
  }
}
```

○ super();— used to invoke immediate superclass constructor.

- with inheritance constructors will be invoked from ~~top~~ bottom to top
- constructors will be executed from Top to Bottom
- Super is the first line in the constructor.
- when ur not writing any super() in the constructor, default super() will be inserted by the JVM.
- when u writing any super(), JVM doesn't insert any super()
- Only one super() is allowed in the constructor

Super is used to invoke.

```
class A
{ int x, y;
  A()
  { S.O.P("A..def.con");
  }
  A(int x, int y)
  { this.x = x;
    this.y = y;
  }
}
class B extends A
{ int a, b;
  B()
  { S.O.P(" B..def..con");
  }
  B(int a, int b, int p, int q)
  { super(p, q);
    this.a = a;
    this.b = b;
  }
  void disp()
  { S.O.P(x);      S.O.P(b);
    S.O.P(y);      }
```

```
class Demo
{ public static void main(String arg[])
  { B obj1 = new B();
    obj1.disp();
    B obj2 = new B(1,2,3,4);
    obj2.disp();
  }
}
```

o/p:

```
0
0
0
0
3
4
```

```
class A                                  A()
{ int x = 10; void show()                0
              { S.O.P(x); }
}
class B extends A
{                          (B() { }
  int x = 20
  void disp()
  { x = 30
    S.O.P(x);
    S.O.P(this.x);
    S.O.P(super.x);
    super.show();
  }
class Demo
{ public static void main(String arg[])
  { B obj = new B();
    obj.disp();
```

o/p:

```
30
20
10
10.
```

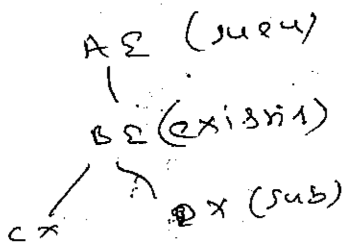
method Overriding:

Implementing superclass mrd In the sub class, with the superclass mrd signature is called mrd overriding.

Rules:

~~1 mrd~~

```
class A
{
                    static
                    final etc..
specifier  modifier  void show() throws someexcep
  public    {
  private     //
  protected }
}
class B extends A
{
            void show()
            {
              // new
            }
}
```



Rules:

1. mrd signature must be same, return type may be anything.
2. when a superclass has a specifier, we must use same specifier (or) any other specifier which has highest privilages.

private	— private, def, pro, pub
default	— def, " "
protected	— pro "
public	— public
superclass	subclass

3. when superclass mrd throws any mrd level exceptions, in sub class;
   - ↳ we can omit that exception
   - → we can use same exception. (BE)
   - → we can use subclass exception to existing super ~~class~~ exception. (CX, DX)
   - → we can't use superclass exception to existing Exception. (AE)

4. we can override static mtds, ~~i.e a~~
~~s~~ when superclass mtd is static
sub class mtd also ~~must be~~ static.

5. we can override private mtds also

6. final mtds can't be overridden.

~~@~~* final classes can't be extended.

(*) Mtd signature must be same & return
Type may anthing

Abstract class:

```
abstract class A
{ abstract void show();
  void m1()
  { System.out.println("hello");
  }
}
class B extends A
{ void show()
  { S.o.p("new thing");
  }
}
```

```
class Demo
{
  psvm(Str args[])
  {
     B obj = new B();
     obj.show();
     obj.m1();
  }
}
```

op: newthing,
hello.

15/06/1~~98~~2006

1. Abstract class is a class, with the
keyword, we should may/maynot contain
abstract mtds & concrete mtds.

2. when ur unable to implement the mtd
i.e when ur unable to provide the
body of the mtd, make the mtd as
abstract.

3. when u write one/more abstract mtds
in the class, we should make the
class as abstract. & reverse is not
a~~bstract~~ True.

4 Abstract classes can't be instantiated but we can create the reference var.

5. when any class is extending abstract class, we have to override all the abstract mtds in ~~the~~ subclass. Otherwise. make ~~the~~ the sub class as abstract:

```
Eg: abstract class A
  { int a;
    static int b;
    static final int c=30;
   A()
   { system.out.println("default..A");
   }
   A(int a, int b)
    {
     this.a = a
     this.b = b
    }
   abstract void show();
   final void m1()
   {
       s.o.p(a);
      system.out.println(b);
   }
   final static void m2()
    {  s.o.p(c);
     }
   }
class B extends A
{
  B()
  {  s.o.p("default...B");
  }
  B(int a, int b)
{   super(a,b);
 }
void show()
 {  s.o.p("nothing");
 }
}
class Demo
 { public static void main (String as[])
   {
```

```
B o1 = new B();
   o1.show();
   o1.m1();
   o1.m2();
B o2= new B(88,8r));
   o2.show();
   o2.m1();
   o2.m2();
}
```

1. we can write concrete static mtds in the abstract class.
2. we can't write abstract static mtds in the abstract class.
3. we can write concrete final mtds in abstract class.
4. we can't write abstract final mtds.
5. we can't write abstract constructors.

Interface: is fully abstracted class which contains only constants & abstract mtds.

Syntax:

```
interface interface-name
{ datatype var-name = value;
   - - - -   - -
   - - - -   - -

  returnType m-name(parameters);

   - - -
   - - -
}
```

Eg:

```
interface animal
{
      String color = "red";
final string   weigh = "2";
      void eating();
public abstract void sleeping();
}
```

- all the var are final & static by default.
- all the mtds are public & abstract by default.
- we can't write var, constructors & concrete mtds in interface
- Interfaces can't be instantiated but we can create the ref' var.
- Any subclass has to implement all the abstract mtds.
- when any subclass implements interface, it should override all the abstract mtds, Otherwise make the subclass as abstract.

class extends class } possible
class implements interface }
interface extends interface

interface ~~class~~ not possible

```
Eg: interface intre
    { void m1();
      void m2();
    }       intreImpl
class implements intre
  {
   public final void m1()
    {   s.o.p("impl m1");

    }
   public final void m2()
    { s.o.p("impl m2");

    }
  }
class IDemo
    public static void main( String as[])
  { intreimpl obj = new intreimpl();
       obj.m1();
       obj.m2();

  }
}
```

① ~~new~~ non static mtds can't be overridden to be static

② static mtds can't be overridden to to be non static.

③ we can achieve multiple inheirance with interface

```
Eg: Interface Inte1
    { void m1();
    }
    interface Inte2
    { void m2();
    }
    class ~~imple~~ Inteimpl implements
                  Inte1, Inte2
    {
      // override m1() & m2() here
    }
```

Go to 19/06/2006

differences & then start here

String Tokenizer: 16/06/2006

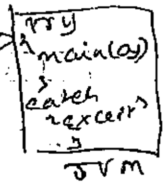
```
class X
{ public static void main(String args[])
 { String str="i am Mr vas Dande";
   String s="";
   S.o.p("forward...");
   for(int i=0;i<str.length();i++)
   {
        if(str.charAt(i)!=' ')
        { s=s+str.charAt(i);
        }
       else
        { S.o.p(" "+s);
          s=" ";
        }
   }
   S.o.println(s);
   ~~s=""~~ s=" ";
   ~~System.out.println(s);~~
   S.o.p("backward");
for(int i =str.length()-1 ; i>=0 ; i--)
  {  if (str.charAt(i) != ' ')
     {   s =s+str.charAt(i) ;

     }
     else
       {  s.o.p(s);
          s=" ";

       }
  }
  s.o.p(s);
  s=" ";
}
```

~~Backward~~ forward	backward
I	ednad
am	nav
mr.	rm
ran	ma
dande	I

o/p

```
class StringTokenizer extends
 Object implements Enumeration {
 {  StringTokenizer(String, String);
    StringTokenizer( String);
    boolean hasMoreTokens();
    String nextToken();
    String nextToken(String);
    boolean hasMoreElements();
    Object nextElement();
    int countTokens();
```

```
class X
{ public static void main(String args[])
  { int []a = new int[5];
    a[7]=99; →problem
    s.o.p("i am not there");
  }
}
```



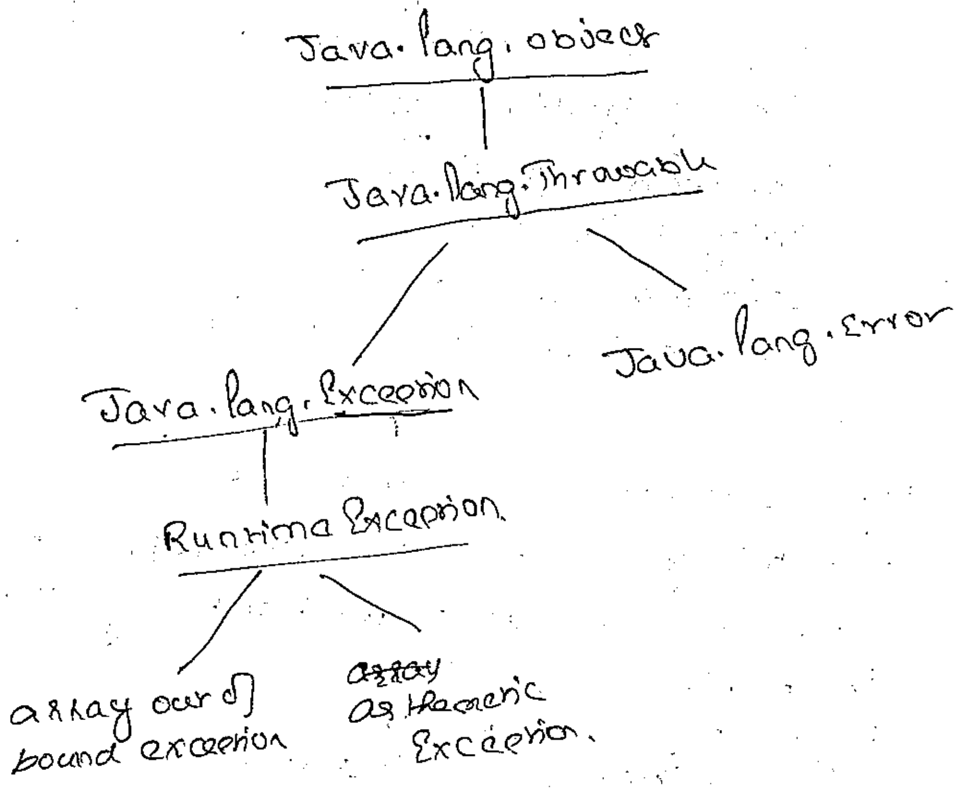
JVM:

1. Monitoring all the stmts.
2. If an error, then identifies corresponding exception class.
3. create the object for exception class
4. Throws the object
5. catch the object & Terminate the prog.
6. JVM displays info in that object.

Eg:
```
class X
{
  p s v m(String args[])
  { try {
      int []a = new int[5];
      x = 10/0;
      a[7]=99;
```

~~Jvm try blk used all the errors all the ki~~

```
catch(Exception e)
{
   s.o.p("i caught it")
   s.o.p(e); (or) s.o.p(e.getMessage());
   (or) e.printStackTrace(); s.op(e.getCause());
}
s.o.p("i am not there");
}
```



o/p:
1) i caught it
2) java.lang.ArrayIndexOutOfBoundsException: 7
3) 7
4) (2) --- at x.main(x.java:7)

```
class Throwable extends object
    implements Java.io.serializable &
{
    Throwable();
    Throwable(String);
    Throwable(String, Throwable);
    Throwable(Throwable);
    String getMessage();
    Throwable getCause();
    String toString();
    void printStackTrace();
    void printStackTrace(printStream);

}
```

-> Try without catch not possible

-> ~~catch~~ Try & catch are one after the other immediately without any stmts in betn.

```
try {          }             try          }
}              }  ~~Error~~  {            }
catch          }  ✓          }            }  wrong
{              }             stmt1        }
}              }             catch        }
                             {            }
                             }
```

• ~~we can~~ do ~~the exception~~ handling

• When u got any prob. in ur Java code than JVM will handle the problems. If u want u can also handle it

• we have 2 types of problems.

1. Exception which can be handle
2. Error which can't be handle
   Eg: class defn not found error.
   no such mtd error
   -> without defining class if we create object.
3. All exceptions in Java are classes.
4. All exceptions are sub classes of Java.lang.Exception.
5. In all Exception Classes we have only constructors, we don't have any other mtds, All exception subclasses are using superclass throwable mtds - like getmsg(), printStackTrace(), toString().

6. we can handle the exception, with the following 5 keywords.

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

try: blk is used to place the stmts, which we want to monitor specially; bcoz u r expecting some prob. in those stmts.

catch: blk is used to catch the exceptions raised in try blk.

- When u write try, catch is mandatory. Other stmts are not allowed betⁿ try & catch. i.e catch is followed immediately by try.
- For one try we can write more than one catch blk.
- when u writing more than one catch blk, exceptions ~~in~~ ~~class~~ order in catches must be subclass to superclass
- Other stmts are not allowed betⁿ catch blks.

finally: blk is used to execute some mandatory stmts, bcoz finally blk will be executed once whether ~~there~~ is an exception raised in try blk or not. Other stmts are not allowed betⁿ catch & finally, try & finally.

(1)
```
try.
{
}
catch()
{
}
catch()
{
}
```

(2)
```
try.
{
}
catch
{
}
finally
{
}
```

(3)
```
try
{
}
finally
{
}
```

- only one finally is allowed.
- when u have stmts like system.exit(0) in try blk than finally will be not be executed. This is ~~the~~ only one case where finally not executed. Remaining all the senarioes, finally blk will be executed

```
Eg: class X
{ p s v m (String args[])
 { try {
     int []a=new int[5];
    // a[7]=99;
     int x =10/0;
     s.o.p("ok ok");
    }
  catch[ ArthematicException ae)
    {
      s.o.p(ae.getMessage());
    }
  catch (ArrayIndexOutBoundsException e)
   {
     s.o.p(e.getMessage());
   }
  catch (Exception e)
   {
    System.out.println(e);
   }
  finally{
   {
     s.o.p("I am here");
   }
 }
```

1906 : 6

Throws:

Key word is used to specify the mtd level exceptions.

When u writing any stmts inside the mtd, those stmts may throw some exception. If u want to handle the exceptions, provide try, ~~case~~ catch block for those stmts. as follows.

```
public void m1()
{
 try{
   int a=10/0;
  }
 catch(Exception e)
  { s.o.p(e);
  }
}
```

If u don't want to handle the excep^n inside ~~the~~ mtd, instead of try, catch blocks provide mtd level exceptions as follows

```
public void m2() throws ArrayIndexOutOfBounds
                         Exception, ArthmeticException
{
   int a=10/0;
   int x[]=new int[5];
     x[10]=99;
}
```

In ~~the~~ above mtd m2, we are not ~~handling~~ specifying ~~the~~ exceptions, but we are indicating the exception for ~~the~~ caller. i.e caller of ~~this~~ mtd m2 has to handle the exceptions as follows:

```
   try{
      m2
   }
  ~~catcht~~
Catch (exception e)
   { s.o.p(e);
   }
```

Eg:
```
class Hai
{
 p s v m (String as[]) throws Exception
 {
  try
  {
   m1();
  }
  catch(Exception e)
  {
   s.o.p("yes");
   e.printStackTrace();
  }
 }
 Static void m1() throws Exception
 { m2();
 }
 Static void m2() throws Exception
 {
   m3();
 }
 Static void m3() throws Exception
 { m4();
 }
 Static void m4() throws Exception
 { int x=10/0;
 }
}
```

Error if any:
~~o/p~~: Unreported Excep^n Java.lang.Exception; must be caught (or) declared to be thrown.

Throw: is used to throw the exceptions in ur own.

JVM

① Monitor the error —— JVM

② Any Problam, Identify the excep^n class

③ Create the object } JVM | Throw

④ throw the object

Throw

⑤ catch (or) declare to catch — JVM | Throw/ trycatch/finally

[illegible]/throws

If we use try, catch, finally & throws keyword, then we can handle the ⑤th steps, remaining all take care by JVM.

If we want to handle ②, ③ ④ & ⑤ then use Throw.

JVM handle only Built in Exceptions.

This will not handle (JVM) application level exceptions (or) user defined Excep^ns.

* monitoring will always be done by JVM.

Syntax: throw object;

Eg: 1. throw new customernotfoundException;

2. throw ce;

Eg:
```
import Java.io.*;
import Java.sql.*;
Class Hai
 { public s v m( String as[]) throws IOException
  { m1();
  }
 static void m1() throws IOException
 { try{
      m2();
    }
   catch( Exception e)
    { s.op("yes");
      throw new IOException();
    }
  }
 static void m2() throws ArithmeticException,
                     SQLException.
 {
    m3();
  }
static void m2() throws ArithmeticException,
                        SQLException
{ m1();
  throw new SQLException();
}
static void m1() throws ArithmeticException
{
int a=90; b=80;
  int x=a/0;
  }
}
```

User defined Exceptions:

1. write ur own exception class by extending java.lang.exception (or) java.lang.runtime exception.

2. write one or more constructors based on ur requirement.

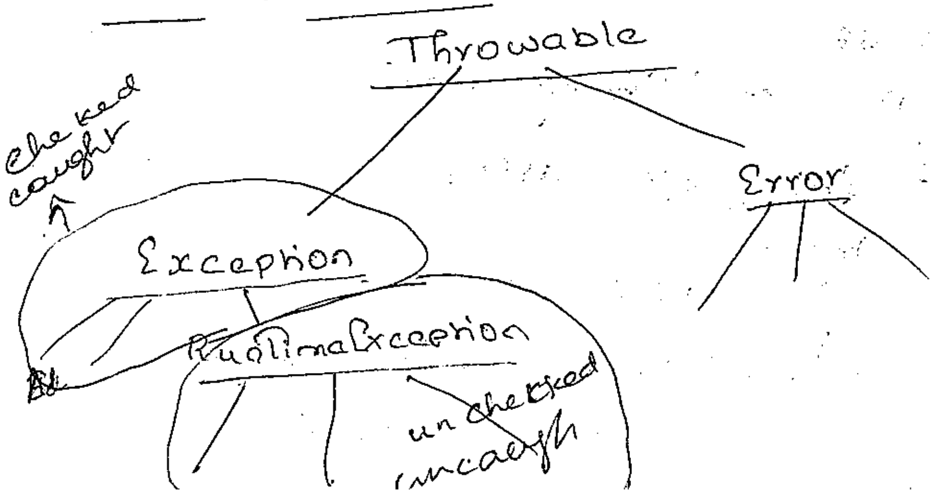
3. override ~~to~~ toString mtd.

4. If req; override equals & hashcode(#) code mtds.

Eg:

```
class InvalidCCException extends
                          Exception
{
 String ccn=" ";
 public InvalidCCException(){ }
 public InvalidCCException(String ccn)
 { this.ccn=ccn;
 }
 public String toString()
 { return "Credit card Number"+ccn+"Invalid
            ...Try Again";
 }
}
class UE
{ psvm(String as[])
  { String ccn = as[0];
    try { m1(ccn);
    }
    catch(InvalidCCException e)
    { S.o.p(e); / e.toString()
 }. }
public static void m1(String ccn)
        throws InvalidCCException
{
  if(ccn.length()==16)
    { S.O.P("ok ok");

    }
  else
     { throw new InvalidCCException();
                    (or)
     } throw new InvalidCCException(ccn);
  }
          o/p : javac US.java
                java US 123456719
                         101112.
}
```

Types of Exceptions:



Based on verification, we have 2 types of exceptions

1. Checked Exceptions

2. Unchecked Exceptions

1. Checked Exceptions:

These are excepts which are verified by the compiler at compilation time.

1. all the subclasses of exception class, except runtime excepts & its subclasses, are called as checked excepts.

2. Unchecked Exceptions:

are exceptions, which are verified at runtime.

1. RunTime Excepts class & its subclasses are called as ~~check~~ unchecked Excepts.

we can also divide the excepts into 2 following categories.

1. caught
2. uncaught

caught Exceptions:

we must handle this type of excep's.
Otherwise it will give an error called
" must be caught (or) declared to be
thrown".

Must be caught means we should
provide try catch mechanism

Declared to be thrown means, we should
provide mtd level Exception.

[Javap.Java.io.IOException]

Uncaught Excep'n:

check or caught → are same }
unchecked or uncaught → are same }

~~only~~ excep for Userdefined excep'n.

no need handle these excep's,
Compiler won't verify this & it won't
give error called must be caught
or declared to be thrown.

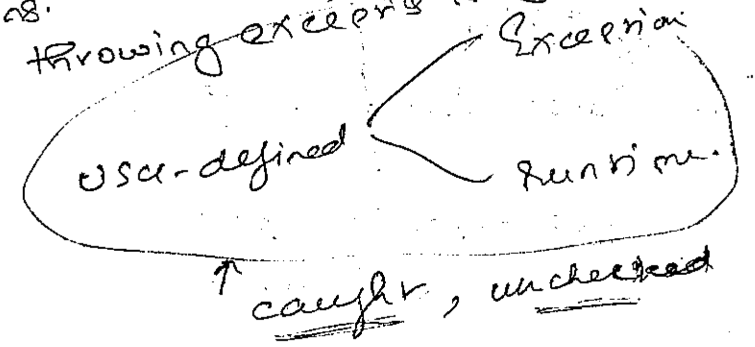
Built in Exceptions:

All checked excep's are caught excep's

All unchecked excep's are uncaught excep's.

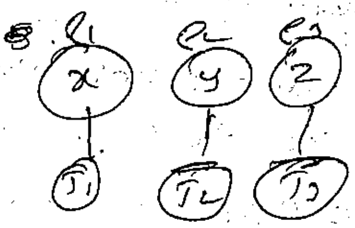
User Defined Exceptions:

are unchecked Excep's (Runtime Excep's)
are handled at runTime
UDE are caught Exceptions:
~~User~~ UDE are caught
All User throwing exceptions are caught
exceptions.
JVM throwing excep's may be caught (or) uncaught



JVM throwing → built in → any thing
User throwing → caught excep'n

Multithreading: Threads

Thread based Multitasking	Process Based Multitasking
(part of prog')	(prog')
1 prog — 5 Tasks 5 Threads	5 progs — 5 Tasks 
(1) memory: less P1 P1 T2 T3 T4	
(2) Context Switching Easy & fast.	difficult & Slow
(3) Communication - Easy	difficult

```
class Thread extends Object
        implements Runnable
   { Thread();
     thread (Runnable);
     Thread (ThreadGroup, Runnable);
     Thread (String);
     Thread (ThreadGroup, String);
Thread (Runnable, String);
Thread (ThreadGroup, Runnable, String);
Thread (ThreadGroup, Runnable, String, long);
  int MIN-PRIORITY ;  1
  int NORM_PRIORITY; -5
  int MAX-PRIORITY; -10
  static native Thread currentThread();
  static native ~~Thread~~ void yield();
  Static native void sleep(long);
  throws InterruptedException ;
        static void sleep(long, int)
  throws InterruptedException;
    ~~Sy~~ synchronized  void start();
    void run(); ✓
    final void stop(); ✓
   final synchronized void stop(Throwable);
    void interrupt(); ✓
   static boolean interrupted(); ✓
   boolean isInterrupted();
   void destroy();
   final native boolean isAlive();
final void suspend();
final void resume();
final void setPriority(int);
final int get " ();
final void setName(String);
final String getName();
final ThreadGroup
        getThreadGroup();
static int activeCount();
final void setDaemon(boolean);
final boolean isDeamon();
}
public interface Runnable {
    void run();
}
```

Eg:
```
class Test1
{ p s v m ( String args[])
{ Thread t = Thred.currentThread();
    S.o.p(t);
    s.o.p(t.getName());
    s.o.p(t.getPriority());
    t.setPriority(9);
    t.setName("Srinivas");
    S.o.p(t);
    s.o.p(t.getName());
    s.o.p(t.getPriority());
}
}
```

creating child threads (or) our own threads :

we can create threads in 2 ways.

1. By extending Thread class.
2. By implementing Runnable interface.

1. Creating Thread By extending Thread Class:

```
class MyThread extends Thread
{
   static int X=99;
   public MyThread(String tname)
    {
       setName(tname);
       start();
     }
   public void run()
   { for(int i=0;i<10;i++)
      { X++ ;
       s.o.p(X+".. "+getName());
       try {
         Thread.sleep(500);
        }
       catch(InterruptedException e) { }
    }
   }
  }
class Test
{  p s v m(String as[])
   {
```

```
new MyThread("pascal");
new MyThread("cobol");
for(int i=0;i<10;i++)
  { s.o.p("Sri"+"main");
    try{
      Thread.sleep(1000);
    }
    catch(InterruptedException e) { }
  }
 }
}
```

2. Creating the Threads by implementing Runnable Interface.

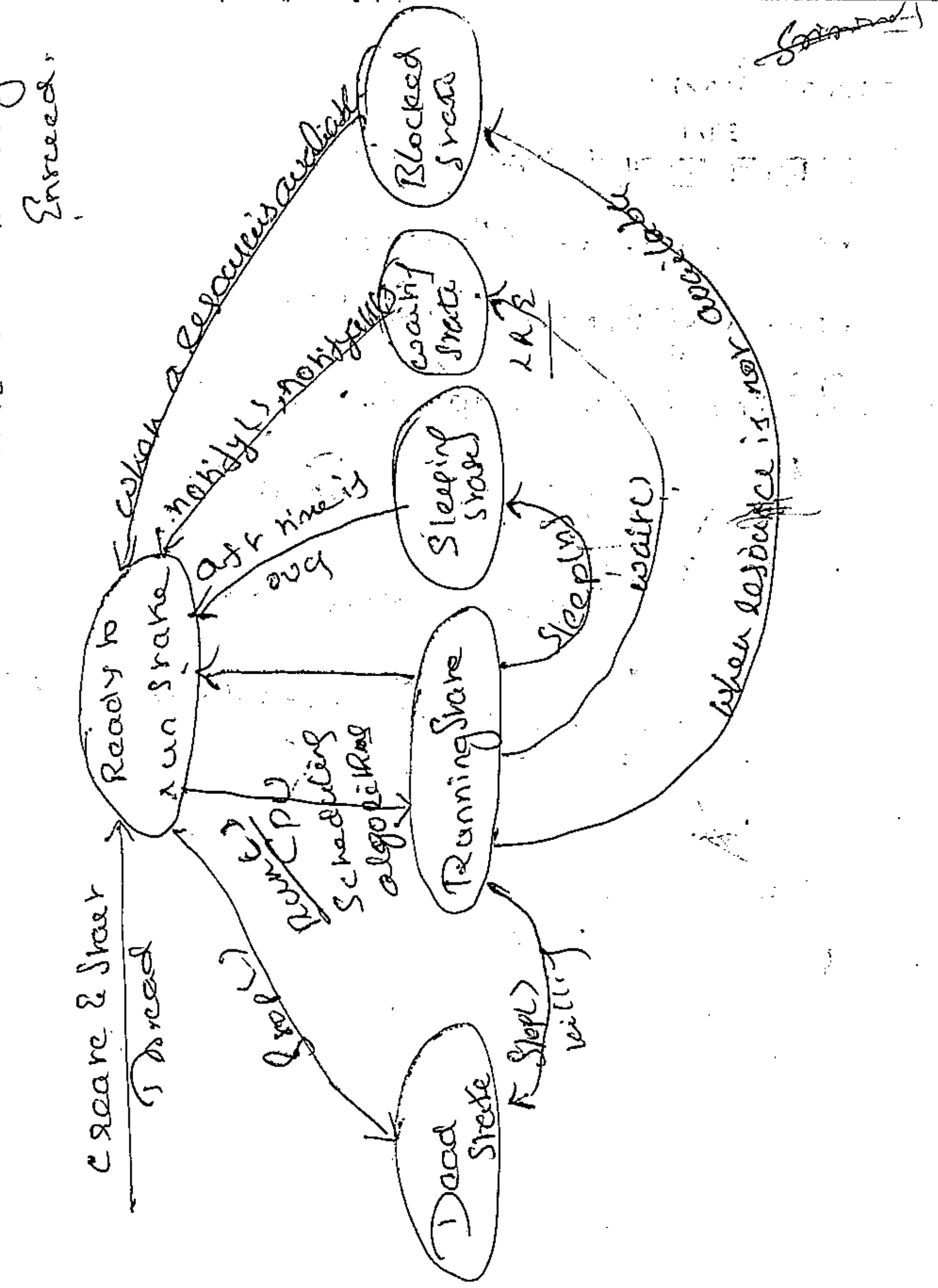
1) pascal - 101 - 200msec ✓ ~200m
   cobal - 102 - 500msec ✓
   Sri+main - 1000sec.

2. Creating the Thread by implementing Runnable interface: 21/06/06

```
class MyThread implements Runnable
{  static int x=99;
   Thread t= null;
   public MyThread(String tname);
 { t = new Thread (this, tname);
   t.start();
 }
public void run()
 { for (i=0; i<10; i++)
   { x++;
     s.o.p("x +" ..." + t.getname());
     try
   { Thread.Sleep(500);
     }
    catch(InterruptedException e)
    { }
 }
}
```

```
class Test1
{  p s v m (String as[])
   new MyThread("pascal");
   new MyThread("Cobol");
   for(int i=0; i<10; i++)
   {    s.o.p("Sri Thread");
        try{
          Thread.sleep(1000);
        }
        catch (InterruptedException e)
         { }
   }
  }
}
```

Thread Life Cycle



1. Aftr creating the Thread, we have to call the start mtd. It is mandatory

2. Aftr calling start mtd, Thread will be in Ready to Run State.

3. Based on the scheduling algorithms, Threads will get the CPU Time, after that run() mtd will be called by the JVM

4. Now Thread will be in Running State

5. We can call the sleep() mtd on the running thread, Then Thread will be moved from Running state to Sleeping state

6. After the specified elapsed time is over, Thread will be moved from sleeping state to ready to run state.

7. when we call wait() mtd on the running thread, Thread will be moved from running state to wait state.

8. We have to call notify() or notifyall() mtds to send the waiting Thread to ready to run state.

Here it will use least recently entered (LRE) alg.

8) when a running thread is waiting for a resource, which is busy, then thread will be entered into block state. Later, a blocked ~~thread~~ will be entered into ready to run state when ~~the~~ requested resource is available.

9. we can kill the Thread by calling stop() mtd.

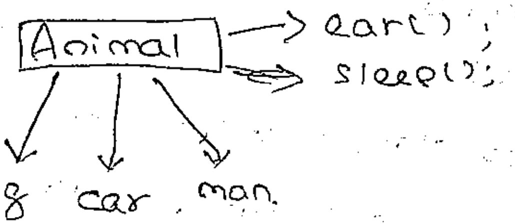
Continue...
from Back,

Difference betn

19/06/2006

abstract class	Interface
1. We can't do multiple inheritance with abstract classes	1. we can do multiple inheritance with interfaces.
2. Abstract class contains variables, constants, concrete mtds, abstract mtds & constructors.	2. Interface contains constants & abstract mtds.
3. we have to extend the abstract class, by using extends keyword.	3. we have to implements the interface with implements keyword.
4. when ur extending abstract class, we have to override all abstract mtds in sub class, otherwise declare the subclass as abstract class.	4. when ur implementing interface we have to override all abstract mtds (or) declare subclass as abstract.
5. we can't create obj for abstract class, but we can create	5. we can't create obj for interface, but we can create ref variable.

Polymorphism: ~~The pro~~ one form behaving differently in different situations is called polymorphism



```
abstract class Animal
{
  abstract void eat();
  abstract void sleep();
}
class Dog extends animal.
{
}
class cat extends animal        class
{
}.
```

we can assign subclassObj to superclass reference var. & reverse is not true.

```
man m = new Man() ✓
Dog D = new Cat() ✗
Animal A = new Dog() ✓
cat c = new Animal() ✗
```

we have 2 types of polymorphism one is compile time polymorphism and second one runtime polymorphism.

we can achieve compile time polymorphism using mtd overloading & we can achieve runtime polymorphism using ~~mtd~~ overriding.

we can decide which mtd will be invoked, only at runtime, because object will be created at runtime

→ we can decide at compile time because ~~be~~ based on ~~Signature~~ parameters of the mtd we can decide.

Eg:

```
      Class
absnacr Animal
 {
    Public absnacr void sleep();
    public absfract void ear();
 }
class Dog exrends Animal
 { public void Sleep();
   { s.o.p("dog sleep");
   }
   public void ear();
  { s.o.p("dog eating");
 } }
Class car exrends Animal:
 {
      P v S()
      { s.o.p ("car Sleep");
      }
      P v e()
      { s.o.p("car ear");
 } }
```

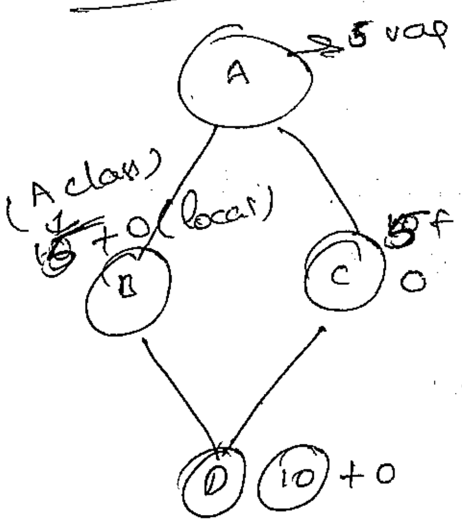
```
~~public~~ sranc
class ADemo
 { p s v m (String args)
   { Animal a = null;
     a = ~~new Dog();~~ mrd1~~(~~
          new Dog();
     mrd1(a);
     a = new car();
     mrd1(a);
   }
   public void mrd1(animal a)
   {
      a.sleep();
      a.ear();
   }
```

O/P :

```
dog Sleep
dog ear
car Sleep
car ear.
```

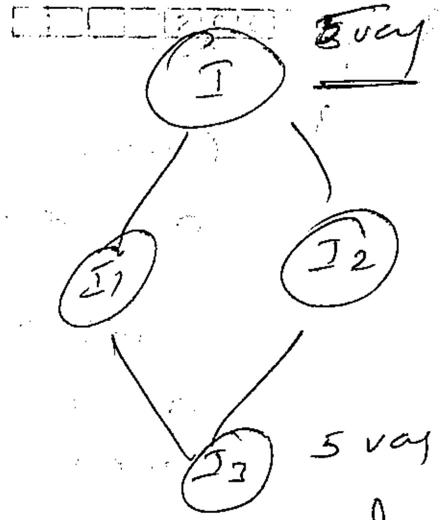
Q) why multiple Inheritance is not allowed?

classes



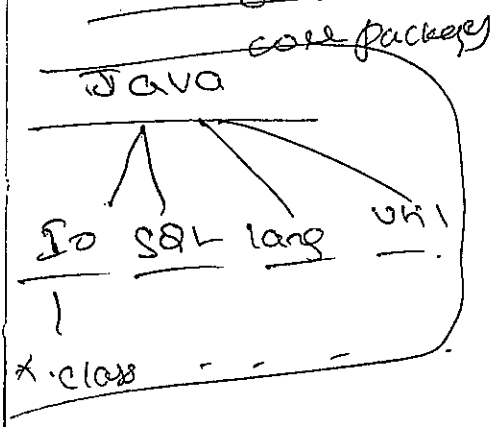
variables are allocated, when ever object is created

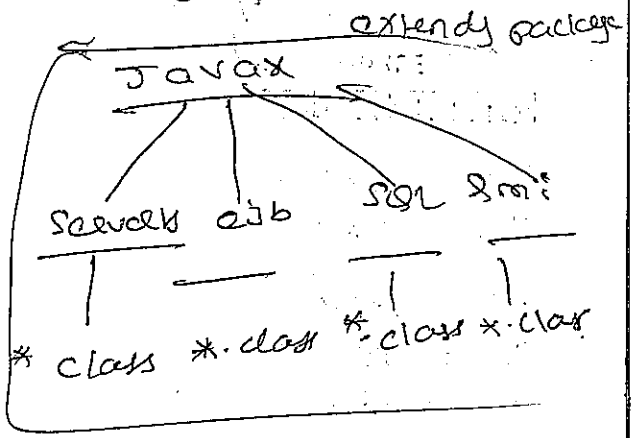
Interface

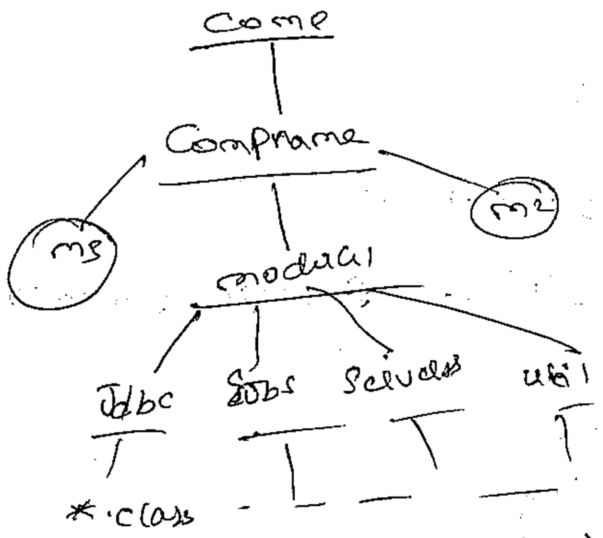


variable are final & static; so duplication not allowed. we can't create object for ~~class~~ interface.

packages: collection of classes.







~~the~~ following are the built in ~~4~~ core packages

Core packages

Java.applet }
Java.awt }
Java.io ✓
Java.util ✓
Java.lang ✓
Java.math
Java.net
Java.nio
Java.rmi ✓
Java.security
Java.sql ✓
Java.text Java.beans
Java.util.logging — Extension packages
Javax.rmi
Javax.sql
Javax.swing
Javax.security
Javax.naming ✓
Javax.xml
e.t.c.

we need only J2SDK for remaining

creating userdefined packages:

~~package~~

Hello.Java

```
package com.sdsoft.core;
public class Hello
{ public void display()
     { s.o.p("Helo");
     }
}
```

Hai.Java.

```
package com.sdsoft.core;
public class Hai
{
public void show()
     { s.o.p("Hai");
     }
}
```

Demo.Java:

```
package com.sdsof.use; import com.soft.core.Hello;
                              "       " .Hai;
class Demo
{ p s v m (String a[])
{ Hello h1= new Hello(); h1.display();
  Hai h2 = new Hai(); h2.show();
  }
}
```

(1) compile above 3 classes as follows

```
E:\b7new\pak> Javac -d . *.Java
```

(2) set the class path

```
E:\b7new\pak> set classpath=%classpath%;
              E:\b7new\pak;
```

(or)

```
E:\b7new\pak>
       set classpath = %classpath%;.;
```

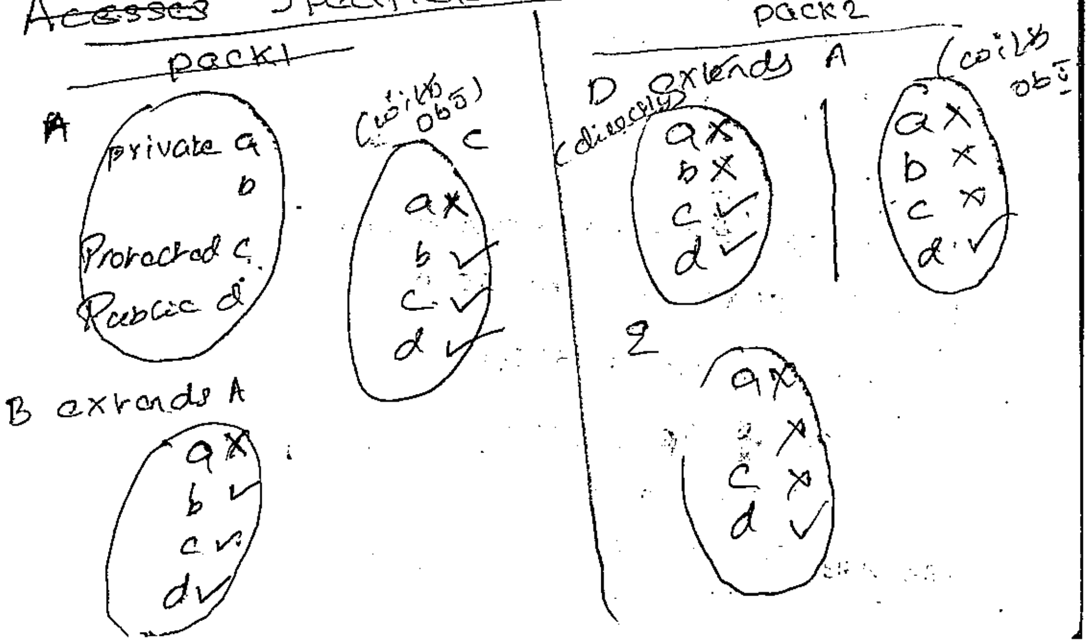
(3) run as follows

```
java com.sasoft.use.Demo
```

we can run this from any folder; bcoz now package is in class path.

1. package declaration stmt must be the first stmt in source file
2. Next (stmt) is one or more i/p stmts and then we can write one or more classes.
3. Only one package ~~stmt~~ declaration. stmt is allowed
4. Recommendation is use public mtds & public classes ~~always~~.

Access ~~Accesses~~ Specifiers: with packages



```
Eg : package com.javasree.pak1;
public class A
{
  private int a=10;
   int b=20;
  protected int c=30;
   public int d=40;
   public void disp()
    {
      s.o.p("A..Disp");
      s.o.p(a);
      s.o.p(b);
      s.o.p(c)
      s.o.p(d);
    }
}

package com.javasree.pak1;
public class B extends A
{
   public void disp()
  {  s.o.p "B...Disp");
   // s.o.p(a);
      s.o.p(b);          s.o.p(d);
      s.o.p(c);
```

```
package com.javasree.pak1;
public class C
{
  Pub void disp()
  {
    s.o.p("C...disp");
     A obj =new A();
   // s.o.p(obj.a);
     s.o.p(obj.b);
     s.o.p(obj.c);
     s.o.p(obj.d);
  }

package com.javasree.pak2;
import com.javasree.pak1.A;
public class D extends A
{  public void disp()          // with obj
  {  s.o.p("D...disp");          A obj = new A();
   // without object           // s.o.p(obj.a);
   // s.o.p(a);                 // s.o.p(obj.b);
   // s.o.p(b);                 // s.o.p(obj.c);
      s.o.p(c);               } s.o.p(obj.d);
```

```
package com.Javasree.pak2;
import com.Javasree.pak1.A;
public class E
{ public void disp()
  { S.o.p("E...Disp");
    A obJ =new A();
   // S.o.p(a);
  // S.o.p(b);
  // S.o.p(c);
  // S.o.p(d);
  }
}
```

```
package com.Javasree.pak3;
import com.Javasree.pak1.A;
   "         "          .B;
   "         "          .C;
   "         "     pak2.D;
   "         "     pak2.E;
class MyDemo
{ p.svm(Stringas[])
```

```
{
  A obJ1 = new A();
    obJ1.disp();
  B obJ2 = new B();
    obJ2.disp();
  C obJ3 = new C();
    obJ3.disp();
  d obJ4 = new d();
    obJ4.disp()
  e obJ5 = new e();
    obJ5.disp();
 }
}
```

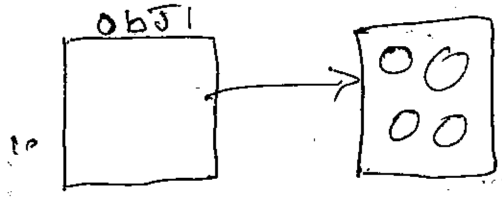
Garbage Collection: in Java we have the New operator to allocate the memory for all instance variables, But we don't have any funtionality to deallocate the mem' allocated using 'NEW' operator.

JVM will do this with the help of Garbage Collector.

GC is a ~~thread~~ service thread which is running behind the scenes & clears ~~the~~ memory. This thread will be created & started by JVM.

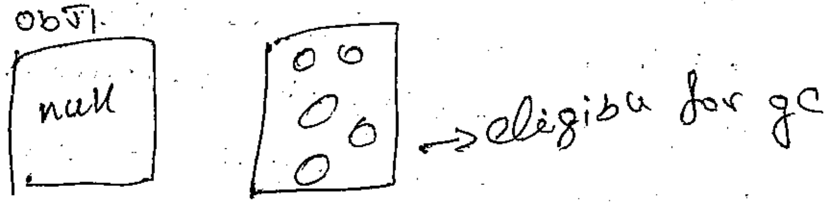
The following are the criterias to find whether the object is used or unused.

①



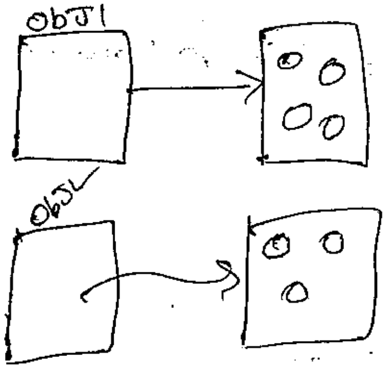
2. obj1=null;

3.

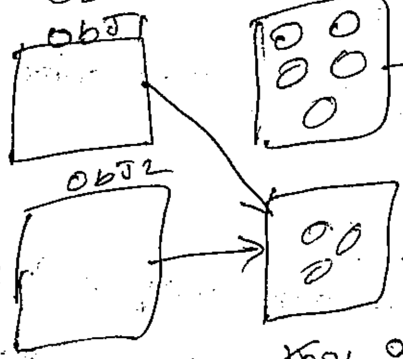


when u assign null to any reference var, then object which is referenced by that referenced var is eligible for garbage collection.

2.



obj1 = obj2



→ when u assign one obj. to another obj., then unreffered obj. will be eligible for the garbage ~~way~~ obj will be eligible for the garbage collection.

3. when ref' var' reaches out of scope, ~~then~~ that obj is eligible for gc

Generally ~~Ge~~ gc will be invoked by the JVM, if u want to invoke as a programmer, we can use the following mtds.

1. System.gc();

~~getRuntime~~

~~Runtime.gc~~

~~2. Runtime().getRuntime().gc()~~

2. Runtime.getRuntime().gc()

→ But JVM doesn't give any guarantee for these calls;

→ We can't force the gc

→ Sometimes some objects may refuse gc to reclaim the mem', bcoz those objects are holding some other resources like i/o, n/w sockets, database connections e.t.c.

So first we have to release the resources & then gc will clean the mem' without problems.

Has a programmer, ur responsible to write the cleanup code, bcoz u known better about the resources wat ur using in ur program.

- Generally we can't write this clean up code, inside the finalize() mtd which is invoked by JVM

- JVM first invokes finalize() mtds, then all resources will be released. then JVM invokes gc; which clears the mem' of unused objects.

If u want to invoke finalize mtd,

1. System.runFinalization(); System.gc()
2. Runtime.getRuntime().runFinalization();
   Runtime.getRuntime().gc();

Inner classes ·
Serializatn Prog'. } Swings
SQL
Javalang package (cloning) factory mrds).
Javautil package

1) creating packages
2) setting the classpath
3) importing the packages

~~package com~~.

-> Java. lang. Object

Object is the top most superclass for all the classes in Java.

toString():

when u override the toString() mrd u can written ur own String rep. when u are not overriding toString() mrd, default implementation of toString() mrd in the Obj. class will be executed and print something like: classname@hexadecimal rep'of Hashcode

Eg: Hello@11A9031

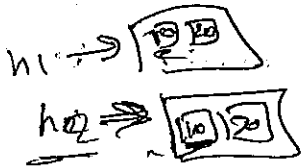
Hashcode: Hashcode is an identification no. for the obj. given by the JVM. Hashcode is used to search the objects very fastly in the heap. we can override hashcode mrd also in ur class, then u have implement some alg; to generate hashcode in ur own.

equals() mrd:

To compare any two objs of a ~~given~~ one class, we have to override the equals mrd in ur class.

```
class Hello
{ int a=10;
  int b=10;
  Hello(int a, int b)
  {
   this.a=a;
    this.b=b;
   }
public boolean equals(Object o)
{
  Hello h=(Hello)o ;
  if(h instanceof Hello)
 {
   if(this.a==h.a && this.b==h.b)
   {
     return true;
   }
  else
        return false;
   }
 public int hashCode()
 {
   return 99;
  }
 public String toString()
 {
  return "a="+a+"b="+b;
  }
}
```

```
public class TestHello
{
  p s v m(String as[])
 {
   Hello h1=new Hello(10,20);
   Hello h2=new Hello(10,20);
   Hello h3=new Hello(10,24);
   if(h1==h2)
   { s.o.p("yes:equa");
   }
   else
    { s.o.p("no:notequal");
    }
   System.out.println(h1);
   System.out.println(h1.toString());
     s.o.p(h1.hashCode());
     s.o.p(h1.equals(h2));
     s.o.p(h1.equals(h3));
     s.o.p(h1.hashCode());
   }
```



h1.equal(h2)

29.06.....6

getClass(): mrd gives class name of the invoked object.

clone(): clone()mrd gives another object, which is similar to the invoked object.

Hello obJ = new Hello (10, 20, 30)

① Hello obJ1 = new Hello(10,20,30)

(or)

Hello obJ2 = ~~clone obJ~~ obJ.clone()

→ Useful in Applets & Drawing images.

To clone any obJ. the class of the obJ. must implement clonable interface.

Clonable interface is a Markable Interface

Markable Interface is without any members

it seems to be like this.

```
public interface clonable
{
}
```

Ur are trying to call clone mrd, ~~to~~ but the class of the object is not implementing clonable interface, then JVM throws an exception called ~~clone not supported~~ "CloneNotSupportedException". ~~exception~~.

String & StringBuffer:

String objects are immutable. i.e we can't modify the string objects.

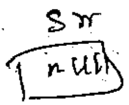
1. StringReference Variables are mutable i.e we can modify the reference var.

StringBuffer ref' var & obJ. are mutable.

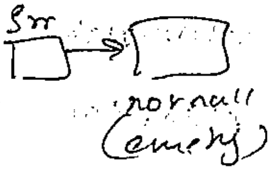
String & StringBuffer are 2 class in Java.lang packages & these 2 classes are final classes: (we can't extend).

String:

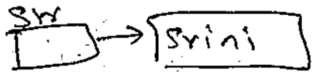
1) String str;
   String str=null } →



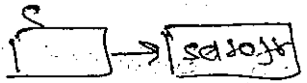
2) String str = new String() } →



3) String str = new String("Srini") } →



4) String s = "sdsoft";



- without using (New) ~~obj~~, we can't create String object.

frm above ③ & ④ we can conclude thr we have 2 ways to create String obj. 1. Using New operator.
2. without Using New Operator.

== vs equals():

① {
String s1 = new String("hello");
String s2 = new String("hello");
String s3 = new String("Hai);

② {
String s1 = "hello";
String s2 = "hello";
String s3 = "Hai";

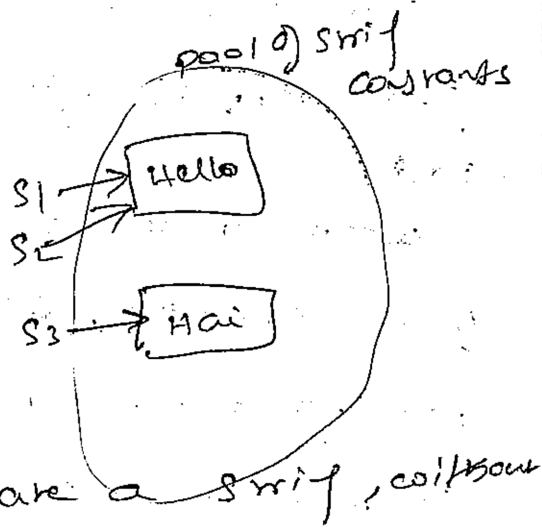
	①	②	equals
s1 == s2	Not	Not	not
s1 == s3	Not	Not	EQ
s1.equals(s2)	EQ	EQ	not
s1.equals(s3)	Not	Not ✗	

```
class SDemo
{
  p s v m( String as[])
{ String s = new String("Srinivas");
  String ss = new String("Srinivas");
  String s1 = "sdsoft";
  String s2 = "sdsoft";
S.o.p(s1.equals(s2));
S.o.p(~~s2~~.s.equals(s2));
if (s1==s2)
{  s.o.p("equals");     } → equals
}
else
{ s.o.p("not equal");
}
if (s==s2)
 { s.op("equal");

}
else
 { s.o.p("notequal")
 }
if (s==s3)
{ s.o.p("equal");

}
else
{ s.o.p("not equal");
}
```

```
S.o.p(s);
 s = s+"d";
 s.o.p(s);
}     Java = Java.lang.String
}
```

Pool of String Constants:

```
String s1="hello";
String s2="hello";
String s3="Hai";
```



when any u create a String, without new; it will check whether that String Constant is in the pool or not, If it is it will assign ~~to~~ ~~same~~ with same address for s2 also.

we can create String obj in 2 ways:

1. with new operator.

2. without New operator.

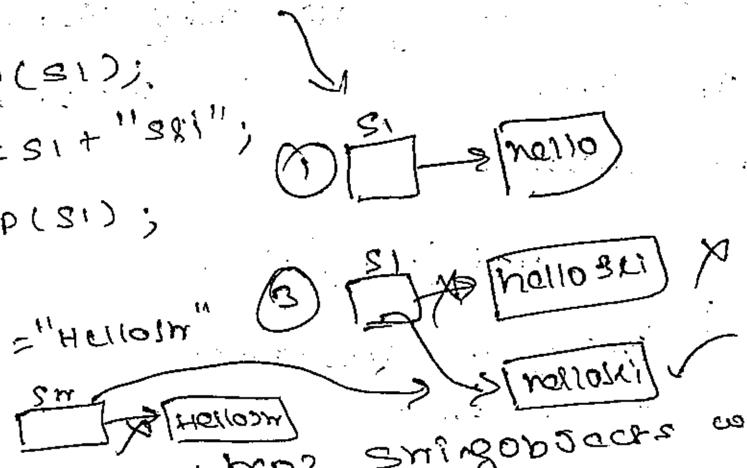
when u create a String obj with new operator, everytime mem' will be allocated.

when u create a String obj without New operator, JVM uses StringConstant pooling mechanism. i.e ~~S~~ when String Constant availaba in the pool, the same Constant address will be assigned to Reference var. It won't create the new String object.

when String Constant not available in the pool, JVM ~~c~~ creates new String Object and the same String constant will be placed in the pool.

→ whenever JVM sees the String which is in doublequotes, it will place in pool.

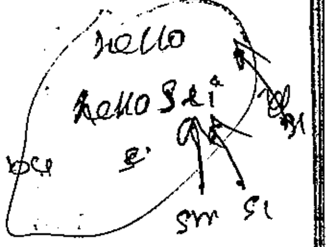
```
① String s1=new String("hello")
②      s.o.p(s1);
③      s1=s1+"sri";
④      s.o.p(s1);

⑤ String str="Hellostr"
```



bcoz String objects we can't modify; but we can modify (mutaba) String ref var.

For this purpose ~~they~~ use String ConstantPooling mechanism.

we can reuse pool for n-number of prog'. Bcoz it is in under JVM not under ur prog'. Using Hashcode we can access fastly the vals ~~[illegible]~~ which are in pool.



when u concatenate String, a new String object will be created everytime; bcoz we can't modify the String obj, once created

```
// String Demo                    30/06/06
class StrDemo
 {
   p s v m (String as[])
   {
     String str = "sri damda";
     s.o.p(str);
     s.o.p(str.length());
     s.o.p(str.charAt(2));
     char []ch = new char[20]; from which pos.
                                we have to place
                                (index)
             ending index of str
     str.getChars(0, 10, ch, 3); destination
             starting index of str
     for(int i=0; i<ch.length; i++)
      {
        s.o.p(ch[i]+"..");
      }
     byte []b = new byte[20];
      b=str.getBytes();
     char x=' ';
```

default value

0	0	0	S	r	i														

```
     for(int i=0; i<b.length; i++)
      { x = (char)b[i];
        s.o.p(x+"...");
      }
     String s1="hello";
     String s2="Hai";
     s.o.p("......."); s.o.p(s2.compareTo(s1));
     s.o.p(s2.compareToIgnoreCase(s1));
     s.o.p(str.startsWith("sri"));
     s.o.p(str.endsWith("de"));
     s.o.p(str.endsWith("vas"));
   }
 }
```

65 – 97
(A) (a)

```
       0123456789 10 ->10
O/p: Sri damda
     10
       i
     ..........s...r...i........d...a....
     ..........s...r...i........d...a...

     -32
     -4
     true
     true
     false
```

SIVA

```
// Sring Demo1
class SrDemo1
{
  p s v m (Sring as[])
  {
    String sr="srinivas";
    s.o.p(sr.indexOf("nivas")),   3
    s.o.p(sr.indexOf("i"));   2 (beginingpos)
    s.o.p(sr.lasrIndexOf("i")); 4  (lasr)
    s.o.p(sr.indexOf('s')); 0
    s.o.p(sr.lasrIndexOf('s')); 7
    s.o.p(sr.indexOf(97)); 6
                      (a)
    Sring x="abc";
     s.o.p(~~Sr~~ x.hashCode()); 96154
     s.o.p(sr.subSring(5)); vas
     S.o.p(sr.concar("dd")); srinivasdd
     s.o.p(sr.subSring(3,5)); ni
     s.o.p(sr);
      s.o.p(sr.raplace("vas"~~))~~,"abc"));
s.o.p(sr.replace(~~"[illegible]~~"i"),'x'); s.op(sr)
     S.o.p(sr.raplaceFirsr("i","x"));
   ~~S.o.pt~~ Sring ds[]=new Sring[10];
       ds=sr.Splir("i");
  for(inr i=0;i<ds.lengm;i++)
   { s.o.p(ds[i]);
   }
  Sring y=" abc ";
  s.o.p(y.lengm());
  s.o.p(y.trim().lengm());

  }

}
```

formula To calucculate the Hash Code of the O/P:

sring: abc

$a\times31^2+b\times31^1+c\times31^0$

$961a + 31b + c$

$961\times97+31\times98+99$

$+2838+99$

```
  6727
 8649
 93217
  2838
    99
 -----
 96154
```

StringBuffer:

```
// StringBuffer Demo
Class SBDemo
{ psvm ( String as[])
  {
    StringBuffer sb=new StringBuffer("sri");
     s.o.p(sb.length());
                    s.o.p(sb.append("vas");
     s.o.p(sb.capacity());
     sb.append("ni1234567890123456");
     s.o.p(sb.length());
     s.o.p(sb.capacity());
     sb.append("12345678901234567890l");
     s.o.p(sb.length());
     s.o.p(sb.capacity());
  }
// StringBuffer Demo1
 class SBDemo1
 { public static void main(String as[])
   { StringBuffer sb = new StringBuffer("srini");
s.o.p( sb.length());
s.o.p( sb.append("vas"));
s.o.p(sb.reverse());
 s.o.p(sb);
 s.o.p(sb.deleteCharAt(5));
 s.o.p(sb);
 s.o.p(sb.delete (2,4));
 s.o.p(sb);
 s.o.p(sb.insert(2,"ri"));
 Char ch[] = {'1','2','3','4'};
 s.o.p(sb.insert(7,ch,1,3));
 s.o.p(sb);
 Integer i =new Integer("9999");
  s.o.p(sb.insert(0,i));
  s.o.p(sb.replace(4,11),"99"));
                    4,11,"99"
 }
}
```

/* 5
srinivas
~~srinivas~~ savinius
 savinias
sr~~inivas~~
savinrs
savinas

sanas
sanas
savinas
savinas234
9999 savinas234
99991234

2/07/06

Wrapper Classes:

primitive type	wrapper class
boolean	Boolean
char	Character
Byte	Byte
Short	Short
int	Integer
long	Long
float	float
double	Double

we have 8 primitive datatypes, there are 8 wrapper classes corresponding to 8 primitive datatypes.

These 8 wrapper classes are useful for the following things.

1. we can add only objects to the collection classes, In this case we need obj rep. for the primitive var.

2. we can do variety of conversions using the mtds provided in the wrapper classes

I String toString(xxx) → primitive

II wrapper XXX valueOf(xxx) → primitive

III $\underset{\text{primitive}}{\text{xxx}}$ parse$\underset{\text{wrapper}}{\text{XXX}}$ (String)

IV

byte	byteValue()	mtds
shrt	shortValue()	
int	intValue()	
long	longValue()	
float	floatValue()	
double	doubleValue()	

String to wrapper   primitive to wrapper

```
V. Byte b1 = new Byte(10);  ✓
   Byte b2 = new Byte("99");  ✓
   Byte b3 = new Byte("A99"); ✗
```

Types of Conversions:

I. primitive type to wrapper object.

Eg: // Eg of Wrappers

```
class WDemo
{ p s v m (String as[])
  { // 1. primitive to wrapper
    int i=99;
    Integer ii1 = new Integer(i);
    Integer ii2 = Integer.valueOf(i);
    System.out.println(ii1);
  // 2. string to Wrapper
    String str = "123456";
    Long l1 = new Long(str);
~~Long ll2 = new~~
    Long l = Long.parseLong(str);
    Long ll2 = new Long(l);
    Long ll3 = Long.valueOf(l);
     S.o.p(ll2)
  // 3. wrapper to primitive
     int x = ii1.intValue();
     byte b = ii1.byteValue();
      S.o.p(x);
  // 4. primitive to String
     double d=99.99;
     String str = Double.toString(d);
      S.o.p(str);
  // 5. Wrapper to String
      Short ss = new Short("54jk");
      String str1 = ss.toString();
       S.o.p(str1);
}
}
```

Boolean & Character:

I
w
```
Character ch=new Character('a');
Boolean b1=new Boolean(true);
Boolean b2=new Boolean("false");
```

II
P
```
parseBoolean("false");
parseCharacter('c') ✗
```

III
W
```
valueOf(boolean)
valueOf (char)
```

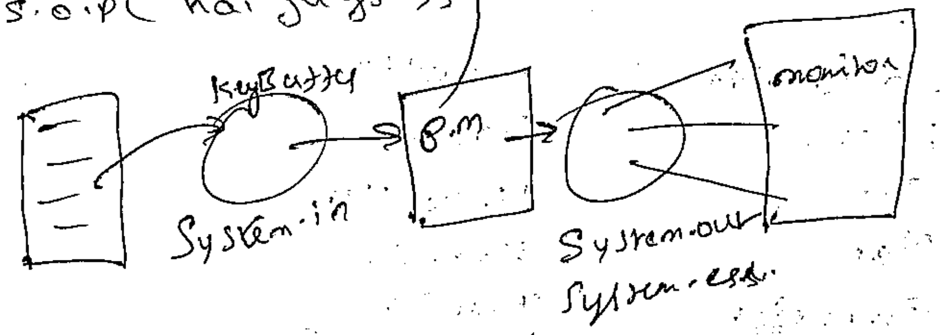
IV
```
String toString(Boolean);
String toString(char); ✗
```

System Class: import. java.util.*;

```
class SysDemo
{ p s v m( String as[])
{
properties p= System.get Properties();
s.o.p(p);
```

```
s.o.p("hello guys");
System.exit(0);
s.o.p("hai guys");
}
}
```

```
PrintStream ps = ~~System.out~~
System.out
ps.println("x");
```



Runtime Class: import java.util.*;

```
class RTDemo{
p s v m (String as[]){
//Runtime rt= new Runtime() ✗
Runtime rt=Runtime.getRuntime();
s.o.p(rt.freeMemory());
s.o.p(rt.totalMemory());
s.o.p(rt.maxMemory());
s.o.p("see the magic");
try{
Process p=rt.exec("notepad.exe");
process p1=rt.exec("calc.exe");
}catch Exception(e){ }
}
}
```

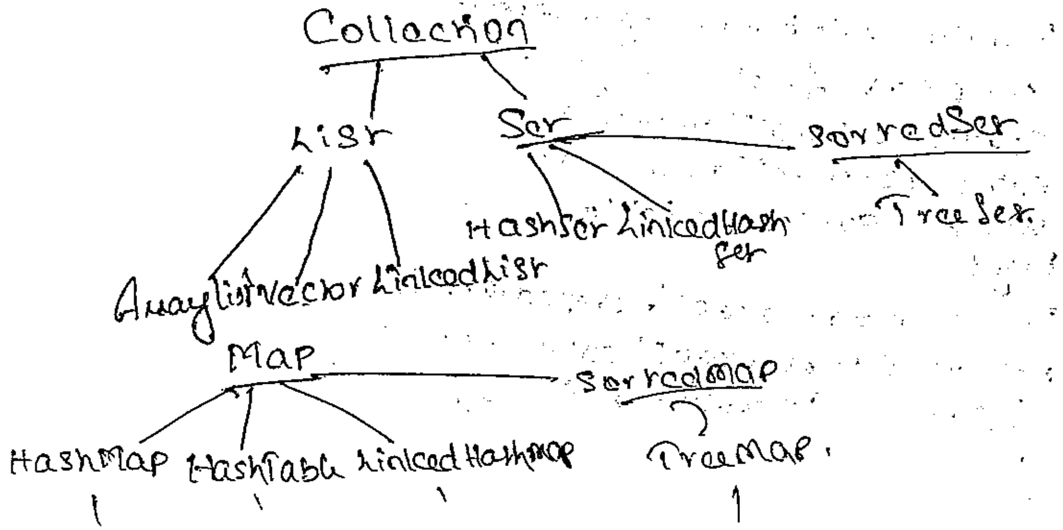
Java-util ~~Package~~ Package:

Interfaces (9)

Collection
List
Set
Map
SortedSet
SortedMap
Iterator
ListIterator
Enumeration

Classes: (11)

ArrayList
Vector
LinkedList
HashSet
TreeSet
LinkedHashSet
HashMap
TreeMap
Hashtable
LinkedHashMap
Collections



- List is a collectn of objects.
- List allows duplicates.
- Set is also collection of objects.
- Set Doesn't allow Duplicates
- Map is collection of key value pairs.

→ Collection frame work was introduced in Java 2. Before java 2, we have 5 legacy classes
1. Vector 2. Hashtable 3. Properties 4. dictionary 5. Stack. All these 5 legecy classes are synchronized by default. And accessing speed also very slow, becz of Synchronization. & They don't have any systematic approach. By keeping these pts in mind, Sun introduce Collection frame work in Java2.

Collection Interface:

```
① interface Collection extends Iterable {

int size();
boolean isEmpty();
boolean Contains(Object);
Iterator iterator();
Object[] toArray();
```

```
 boolean add(Object);
 boolean remove(Object);
 boolean containsAll(Collection);
 boolean addAll(Collection);
 boolean removeAll(Collection);
 boolean retainAll(Collection);
 void clear();
}
② Public interface List extends Collection {
   Object get(int);
   Object set(int, Object);
   void add(int, Object);
   Object remove(int);
   int indexOf(Object);
   int lastIndexOf(Object);
   ListIterator listIterator();
   List subList(int, int);
③ }
Public interface Set extends Collection {

   // Set interface doesn't contain any new method

in its own.

}
④ class ArrayList extends AbstractList implements
List, RandomAccess, Cloneable, Serializable {
  ArrayList(int);
  ArrayList();
  ArrayList(Collection);
```

```
 void trimToSize();
 void ensureCapacity(int);
 protected void removeRange(int, int);
}
⑤ Class Vector extends AbstractList implements
List, RandomAccess, Cloneable, Serializable {
 int elementCount;
   Vector(int, int);
   Vector(int);
   Vector();
   Vector(Collection);
   void trimToSize();
   void ensureCapacity(int);
   Enumeration elements();
   Object elementAt(int);
   Object firstElement();
   Object lastElement();
   void setElementAt(Object, int);
   void removeElementAt(int);
   void insertElementAt(Object, int);
   void addElement(Object);
   boolean removeElement(Object);
   void removeAllElements();
   List subList(int, int);
   void removeRange(int, int);
```

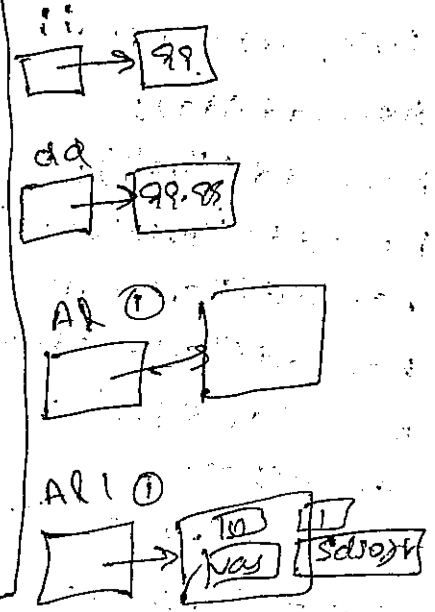
```
⑥ public interface Iterator {
{ boolean hasNext();
  Object next();
  void remove();
}
```

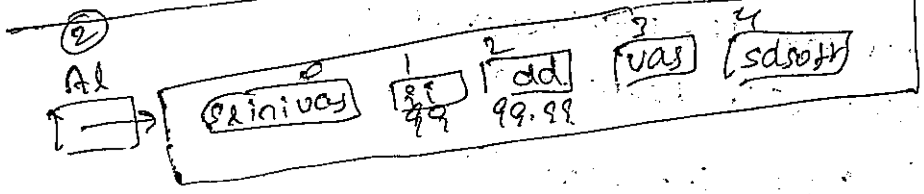
```
⑦ interface ListIterator extends Iterator {
  boolean hasNext();
  Object Next();
  boolean hasPrevious();
  Object Previous();
  int nextIndext();
  int previousIndex();
  void Remove();
  void set(Object);
  void add(Object);
}
```

ArrayList:

```
// ArrayList example
import java.util.*;
class ALDemo
{ public static void main (Sring asC])
 { integer ii=new Jnreger(99);
   Double dd = new Double("99.99");
   ArrayList all=new ArrayList ();
     all.add("vas");
all.add("sasofr");
S.o.p(al.isEmpry());
al.add("srinivas");
al.add(ii);
al.add(dd);
al. addAll(all);
 s.o.p(al);
 s.o.p(al.size());
 S.o.p(al.isEmpry());
 s.o.p(al.contains ("srinivas"));
 s.o.p(al.contains ("sri"));
 s.o.p( al. contains(ii));
 Object []o=al.toArray();
 for(int i=0;i<o.length;i++)
 { s.o.p( o[i].toString());

 }
 s.o.p(al.containsAll(all));
  all.add("abcd")
 s.o.p(al);
 s.o.p(all);
                contain
 s.o.p (al.~~Retain~~All (all));
S.o.p(al);
S.o.p(al1);
S.o.p(al.retainAll(al1));
// al.clear();
S.o.p(al);
S.o.p(al1);
// to itreare arraylist

al1.add(new Intreger(99));
Itreator ir=al1.itreator();
while (ir.hasNext())
{
  S.o.p (ir.next());
}
}
}
```

```
true
[srinivas,99,99.99,vas, sdsofr]
5
false
true
false
true
srinivas
99
99.99
vas
sdsofr
true
[srinivas, 99, 99.98, vas , sdsofr]
[vas , sdsofr,abcd]
true
[vas, sdsofr]
[vas, sdsofr, abcd]
false
[vas ,sdsofr]
[vas, sdsofr ,abcd]
vas
sdsofr
abcd
99
```





```
// Set Example.
import java.util.*;
class VDemo
{
  p s v m( String as[])
{
  HashSets hs=new HashSet(); TreeSet ts=new
                                     TreeSet();
   String ds=null;              // String ds=null;
   hs.add(ds);
   hs.add("99");
   hs.add("srinivas");   } duplicate not
   hs.add("srinivas");          allowed.
   hs.add("sri@sd.com");
   hs.add("9999");
    s.o.p(hs);
    s.o.p(hs.size());
    s.o.p("using list-ite");
    Iterator li=hs.iterator();
    while (li.hasnext())
     { s.o.p(li.next());
  }  }
}
```

O/P:

```
[sri@sd.com, 9999, srinivas, 99, null]
5
using list ite
sri@sd.com
9999
srinivas
99
null
```

	order	Sorted
Arraylist	index } given order	NO
vector	index	NO
Linkedlist	index	NO
HashSet ✓	no (Random)	NO
TreeSet	NO	Yes ✓
LinkedHashSet	givenOrder	NO
Hashmap ✓ (key)	Random (	NO
Treemap (key)	NO	yes ✓
Hashtable (key)	~~givenorder~~ Random	NO
LinkedHashmap	given order	NO

Diff betn Arraylist & Vector?

Arraylist	Vector
1. is a collection Class.	1. vector is Legacy class.
2. not synchronized bydefault	2. Vector is synchronized by default. (allowing one thread at a time).
3. not threadsafe	3. Thread Safe.
4. we can access the ele of arraylist fastly	4. Slowly.
5. we can use iterator on the Arraylist to visit ~~the~~ individual ele in the Arraylist	5. We use iterator & enumeration interface to visit ~~the~~ individual ele in the vector.

Diff betn HashSet & TreeSet

HashSet	TreeSet
1. Gives the ele in random order	1. Tree set gives the ele in Sorted order.
2. HashSet allows null values	2. doesn't allow null values.

Map: is used to store Collection of keys & values.

is not in a collection Interface.

but it is in a collection framework

```
interface Map{
 int size();
 boolean isEmpty();
 boolean containskey(Object);
 boolean containsValue(Object);
 Object get(Object);
 Object put(Object, Object);
 Object remove(Object);
 void putAll(java.util.map);
 void clear();
```

```
Set keySet();
Collection values();
 Set entrySet();

}
Eg: import java.util.*;
class MapDamo {
p s v m ( String args[])
{ TreeMap hm = new TreeMap();
  hm.put("sno","99");
  hm.put("abc","99");
  hm.put("sname","sri");
  hm.put("pqr","99");
  hm.put("email","sri@ed.com");
  s.o.p(hm);
  hm.put("sname","vas");
  s.o.p(hm);
  hm.put("email","vas");
  s.o.p(hm);
//hm.put(null,"hai");
```

```
Set s = hm.keySet();
Iterator i = s.iterator();
while( i.hasNext())
{ Object o = i.next();
  s.o.p(o + "::" + hm.get(o));
  s.o.p(" ");
}
Set ss = hm.entrySet();
Iterator ii = ss.iterator();
while (ii.hasNext())
{
 Map.Entry me = (Map.Entry) ii.next();
 S.o.p(me.getKey() + ":::" + me.getValue());
s.o.p(" ");
}
}
}
```

key	sno	abc	sname	pqr	email
value	99	99	~~sri~~ vas	99	~~sri~~ vas

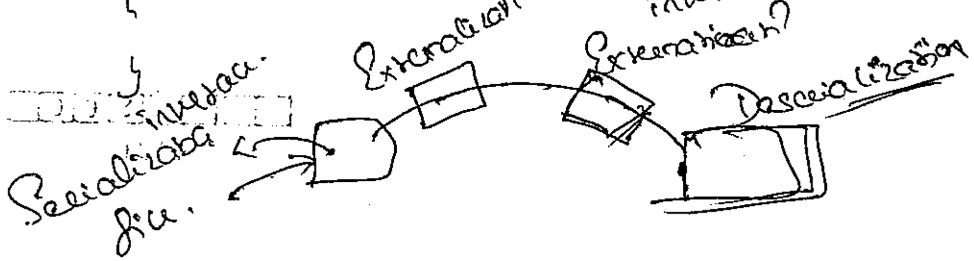
1) Deseeilization:

is a process of reconstructing the obj, from a seeilizable file at the remote m/c.

To seeilalize any object; ~~the~~ class of that obj must implement Java.io.seeializable. otherwise it will throw an exception.

When a class is implementing seeializable interface, the class is eligible for Seeilization. and also all subclass of the class are eligible for seeialization.

```
A implements Seeilizable
{
transient myclass obj=
transient int x;
   └> only for ~~variable~~/used to ~~no~~ non seeilialize.
}
B extends A
{
}
```



If u don't want to seeialize, some var inside the class, then declare the var as Transient variable
                                                         └> (modifier)

bcoz, Transient var can't be seeialize.

Externalization:

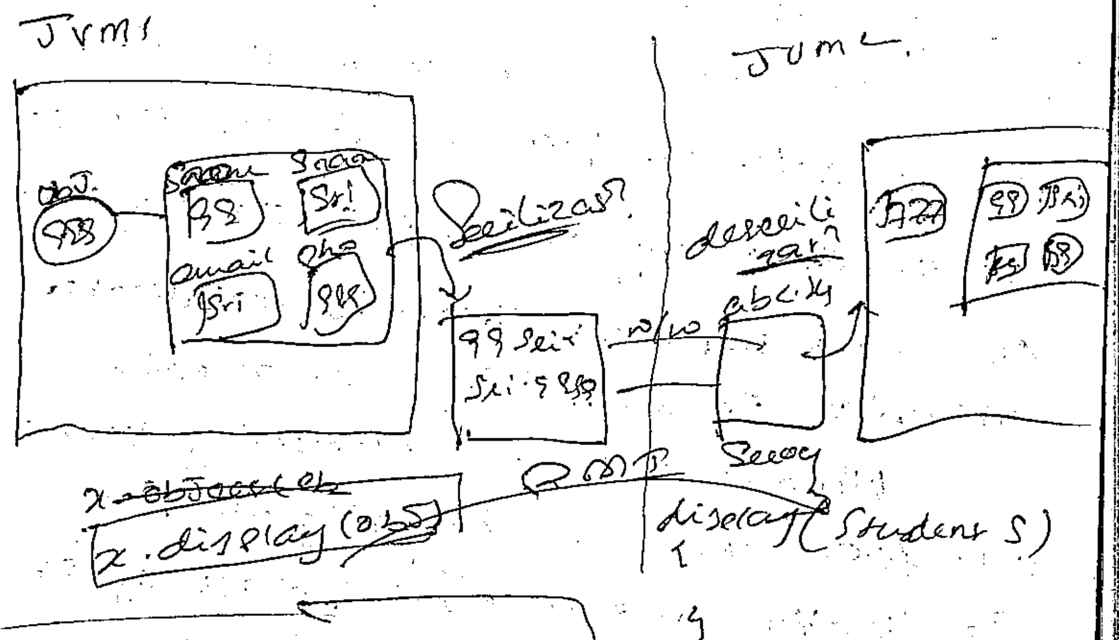
...when we using seeialization, data will ~~be~~ passed through the n/w as it is in the form of a file. we may get the prob. from the hackers.

To avoid these security problems, i want to encode the data at one end; and pass that encoded data to the remote m/c and decode it & reconstruct the object. Only for this purpose sun as introduce externalization concept & provided the interface called Externalizable.

```
(ow)
do { int x = fis.read();
    if(x!=-1)
    { fos.write(x);
    } while(x!=-1)
    ~~fos~~. close();
    fis.close();
}
```

Serialization:

is the machanism used to save ~~the~~ object state into a serializable file. Serializable may be a text file (or) another.

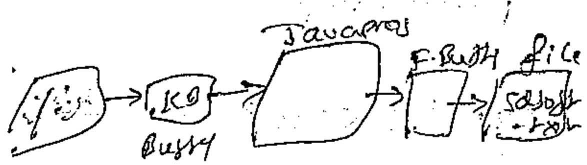


RMI
Cobra
EJB
DCom
} distributed m/c

passing objects through reference
passing promitive datatypes through value

```
S.o.p(c);
}while(ch!=-1)
```

Example to read the data from the keyboard write the data into a file called SD.txt.



```
import java.io.*;
class Ex2
{
  public static void main(String as[])
  {try{
   FileOutputStream fos=new FileOutputStream
                          ("sdsoft.txt");

   BufferedReader br=new BufferedReader
     (new InputStreamReader(System.in));
s.o.p("Enter student names");
   do{
      String str=br.readLine(); [user to entered]
      if(str!=-1) if(!(str.equals("end")));
         fos.write(str);
   }while(!str.equals("End"));
```

```
  catch(Exception e)
   { s.o.p(e);
   }
}
```

Eg: To the read the data from file Sdsoft.txt & write the same data to another file SCIMITAR.txt.

```
import java.io.*;
class Ex1
{ p.s.v.m(String as[])
  { try {        fis
     FileInputStream = new FileInputStream(
                         "Sdsoft.txt");
```

~~BufferedReader br = new BufferedReader (fos);~~

```
   FileOutputStream fos = new FileOutputStream("SCIMITAR.txt");
   do{ String str= fis.readLine();
       if(str!=-1)
       fos.write(str)
     }while(!str.equals("End");
}
```

In Java·io package they are given 2 abstract classes InputStream, Output Stream to manipulate the binary data. Lot of subclasses are available under these abstract classes to operate on binary data.

(-read()) InputStream

- BufferedInputStream
- DataInputStream
- FileInputStream
- ObjectInputStream (subclass
- pipedInputStream

(write()) OutputStream

- BufferedOutputStream
- DataOutputStream
- FileOutputStream
- ObjectOutputStream
- PipedOutputStream

CharacterStream: Operate on Characters, i.e

CharacterStreams are used to pass the characters

CharacterStreams use unicode representations

Bcoz of this, we can achieve Internationalization

i18n

In Java.io.package, there are 2 abstract classes call reader & writer. Lot of subclasses are implemented based on these abstract classes

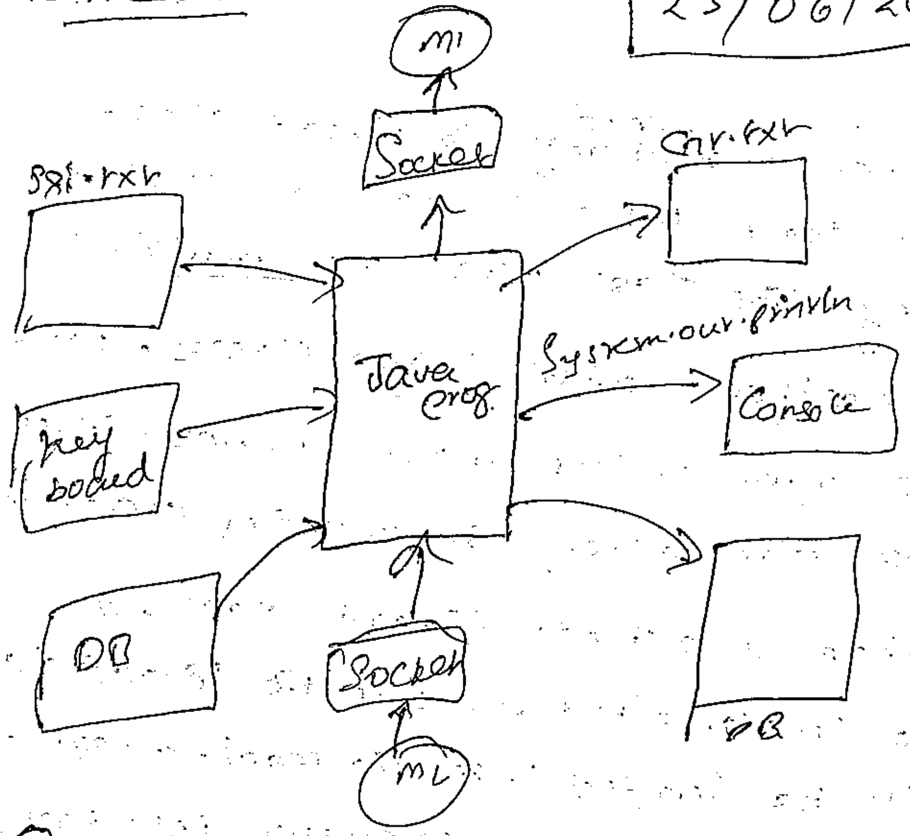
reader & writer read() reader	write() writer
BufferedReader [concrete class of abstract reader()]	BufferedWriter
pipedReader	pipedwriter
FileReader	FileWriter

Eg: to read the data from the keyboard & display it back to monitor.

```
Class Ex1
{ public static void main(String ar[])
  {
   BufferedInputStream bis = new BufferedInput
                                 Stream(System.in)

   BufferedReader br = new BufferedReader(bis)
    char ch = ' ';
    S.o.P("Enter yourname");
    do {
      ch = (char) br.read();
      if(ch!=-1)
```

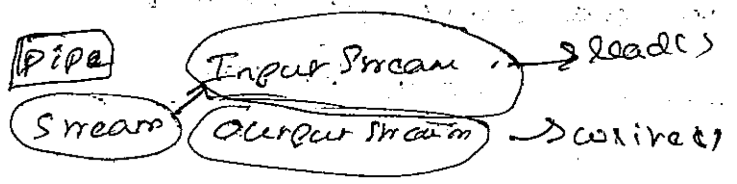
Streams:

23/06/2006



1 Input devices(I)

2 output devices (O)



Stream is an abstraction which is used to pass the data from ur Java prog' to other devices or from other devices to ur Java prog'.

Stream is a pipe which is connected to other physical devices. Based on the data flow, we can devide the streams into 2 types input streams, which are used to read the data. output streams which are used to write the data.

Based on the data we can divide the streams into 2 types.

1. Byte Streams

2. Charactr Streams

Byte Streams: used to pass byte informatn. Almost all the low level services like sockets, in Rmi calls will use byte streams only/-

on t1 & t2; than main thread waits until child threads finishes the task.

~~Dema~~

Daemon Threads: are service threads.

- Daemon Threads live as long as ~~child threads or parent~~ dependent main threads are running

U can make any thread as Daemon by using the following mthds.

void set Daemon(boolean);

Boolean isDaemon();

we can check whether the ~~B~~ thread is Daemon thread or not.

→ we can create the threads in 2 ways 1. By extending thread class
2. By implementing runnable interface

- Implementing runnable is Best.

Reasons:

1. when u ~~extchange~~ extend the thread class, we don't have chance to extend any other class, bcoz no multiple inheritance with Java classes. where as when u implement runnable interface we can extend some other class also

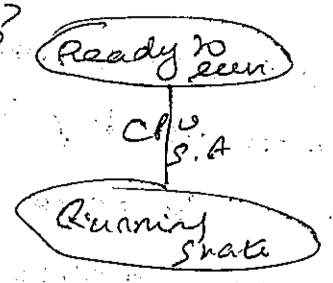
2. when u extends thread class, all the implemented mthds in the thread class will be loaded into the mem. where as when u implementing runnable interface ~~as~~ ~~im~~ we don't get such problem.

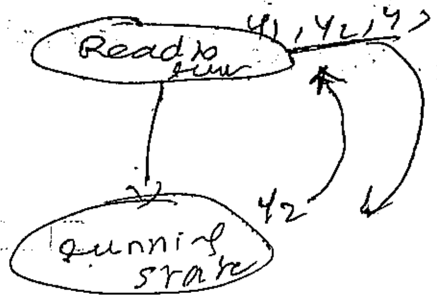
End of Threads

isAlive()
yield()
Join()
Sleep()
start()
stop()
run()
} are in Thread class

notify()
notifyAll()
wait
} are in object class

State





Boolean isAlive()
T F

t2.yield()
no guarantee

isAlive(): mtd is used to check whether the Thread is still alive or dead (kill).

yield(): when u call yield mtd on the running thread, running thread will give a chance to any other thread which is in readytorun state i.e running thread will be placed back to readytorun state, some other Thread which has the heighest priority will be moved from readytorun state to running state. But no Guarantee from JVM side.

Join(): when u call Join() mtd on the threads, than Joined ~~mtds~~ threads will be completed first & then parent thread from where other threads are started. Eg: from the main thread 2 child threads t1 & t2 are started, when i call Join mtd

```
Eg: class abc
 {
   void m1();
   ~~{~~
   s.o.p1();
   s.o.p2();
   }
 Synchronized void m2()
  { s.o.p3();
    s.o.p4();
  }
 void m3()
  {  stmt5();
     synchronized(x)
      { stmt6();
       }
  }
 }
Class
```

Note: In the above eg there is a class ~~abc~~ abc with 3 mtds m1(), m2(), m3().

m1() is ~~not syn~~ nonsynchronized mtd.

m2() is synchronized mtd

m3() is nonsynchronized mtd with synchronized blk.

- when one Thread calls mtd m2(),. than it is not possible to call m2() by other threads, but other threads can call m1() & m3().
- when one thread calls mtd m3(), stmt6 will be executed by that thread only. Other Thread won't access stmt6. But other Threads can access stmt5; bcoz it is not inside the synchronized blk.

```
this.amt=amt;
 this.name=name;
 setName(name);
  start();
}
public void run()
{ for(int i=0;i<=6;i++)
 { checkWithdraw(amt);
  try{
     Thread.sleep(200);
   } catch(Exception e) { }
  }
 }
public void checkWithdraw(int amt)
{
Account act=new Account();
 if(amt > act.getBalance())
  { s.o.p("Hello"+name+"no bufs in.
                    the amt");
}
```

```
else
{ act.withdraw(amt);
  s.o.p(".."+name+"..completed"+act.
                        getBalance());
  }
 }
}
class ATest
 {
  Public s v m( String as[])
  {  AThread t1=new
     AThread("Srini",1000);
     AThread t2=new
     AThread("vas",1500);
   }
 }
```

when u call any synchronized mtd with an object, that object will be locked, then no other threads use that object

Blocklevel Synchronization:

```
public void m1()        public void m2()
{                        {
  ===
            obj
  Synchronized (a1)            synchronized(this)
  {            ↳ locked.  {
     al.add(x);
     al.add(y);
                          }
  }
```

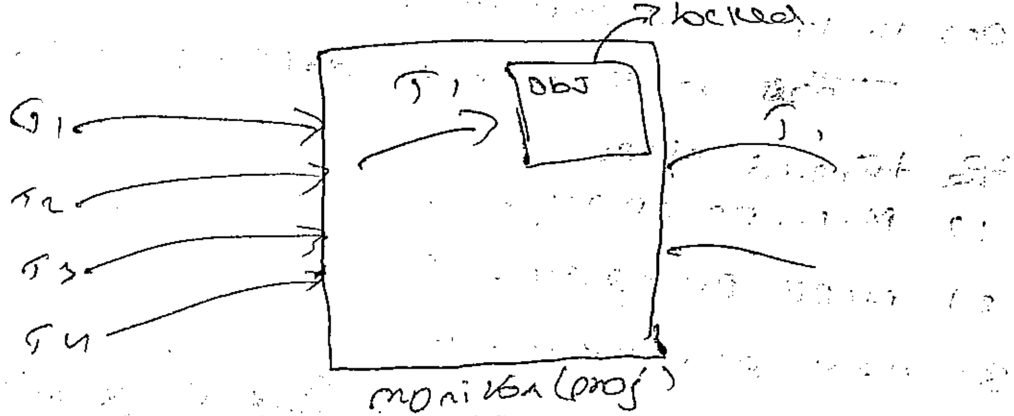
1. When ur using blk level synchronizatn u should pass any object to the blk; that passed object will be synchronized (or) locked

<u>NOTE</u>: In mtd level synchronization, obj of the class which contains synchronized mtd will be locked; where as in blocklevel synchronization, the object which ur passing to ~~the~~ synchronized block (any obj) will be locked.

Eg: for Synchronized.

```
class Account
{
 private static int bal = 10000;
 public int getBalance()
 {
 return bal;
 }
 public synchronized void withdraw(int amt)
 { bal = bal-amt;
 }
}
class AThread extends Thread
 {  String name = null;
    int amt=0;
  Public AThread(String name, int amt)
  {
```

acces the object. This is what is known as "Synchronization".



when i synchronize the object, obj will be locked & will be in monitor. Threads which would like to used locked object. will enter into the monitor & uses the locked obj. Afrer using the object, come out of the monitor. The threads out side the monitor are unable to access the locked obj. we can do the synchronization, ~~using the~~ modifiy called synchronized modify.

we can use this modify in 2 ways one is mtd level synchronization; & 2nd is blk level synchronization.

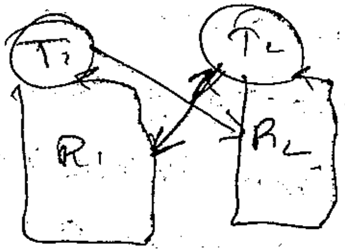
synchronized modify is not allowed for classes, interfaces & variables.

Mtd level Synchronization:

```
public Synchronized void m1()
{
    // body

}
    obj.m1()
     ↳ locked
public synchronized void withdraw()
{
    if (amt <= bal)
        bal = bal - amt
    give cash
}
    obj.withdraw()
         ↑ [illegible]
```

Ran state and another thread will be entered into running state & so on..

Deadlocks:



Eg: Thread T1, which is holding resource R1 is waiting for a resource R2 and Thread T2 which is holding a resource R2 is waiting for a resource R1.

T1 releases the resource R1 after getting R2 & T2 releases the resource R2 after getting R1. They won't release the resources & they won't get the resources also simply T1 & T2 will be blocked. No chance to move out of block state. This Situation is called as deadlock Situation.

Thread Priority:

we can give an integer no. ranging from one to 10 as a priority to the threads.

There are 3 constants defined inside the thread class.

1) MIN-PRIORITY — 1
2) NORM-PRIORITY — 5
3) MAX-PRIORITY — 10

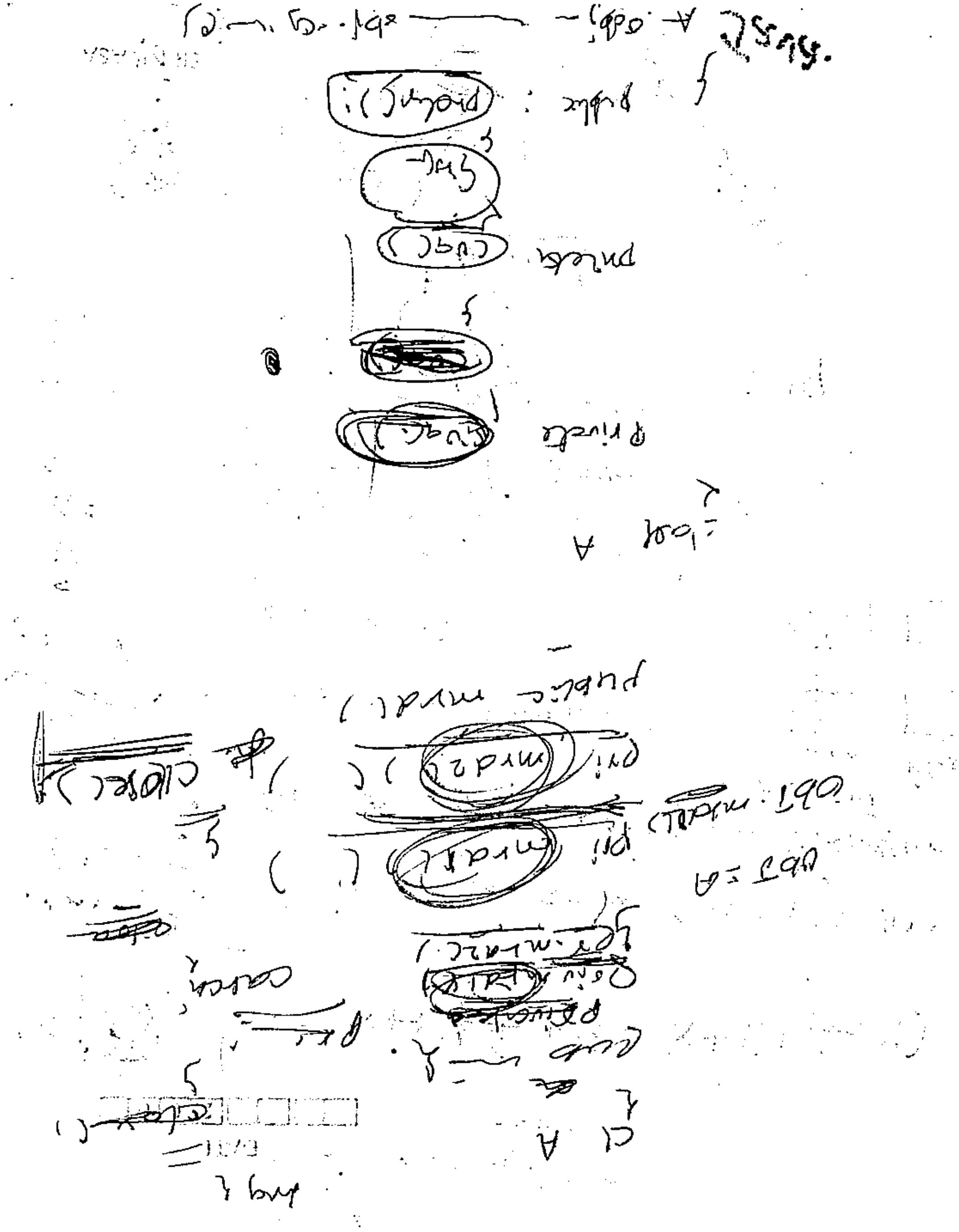
we have 2 methods, to find the priority of the thread & to change priority of thread

1. int getPriority()
2. void setPriority(int)

Synchronization:

In a multi threading environment when all the threads are accessing the obj concurrently, some times we can get inconsistent results. To avoid inconsistent results I don't want to allow all the threads to access the object concurrently.

Eg. I want to allow one by one to



Scheduling Algms:

1. priemtive scheduling alg:

Suppose 2 threads T1 & T2 entre into ready to run state with the priorities 4 & 6 respectively. Now T2 gets the cpu time, bcoz it has heighest than T1. Now T2 is running & T1 is in ready to run state. Another thread called T3 with the priority 9 entered into ready to run state & T3 has heighest priority than the running thread, and then T2 will be preempted & T3 will entre into running state.

2. Time Slice or Round Robin Alg:

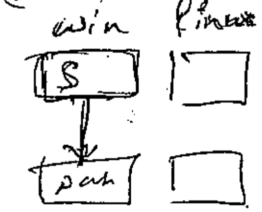
In this algorithm each thread will be given fixed amt of Time and cpu time will be allocated for each thread based on FCFS (or) premtive alg and thread will be in running state for specified Quantum time, afr the time is over, thread will be back to ready to x = 4

a = ++x * ++x * ++x

100

a = (x++) + (++x) + (x++) + (++X)

- Byte code same (all platforms)
- C1 compiler platform dependent (or) not
- arm amenacable no.
- amstrong no.



cls A

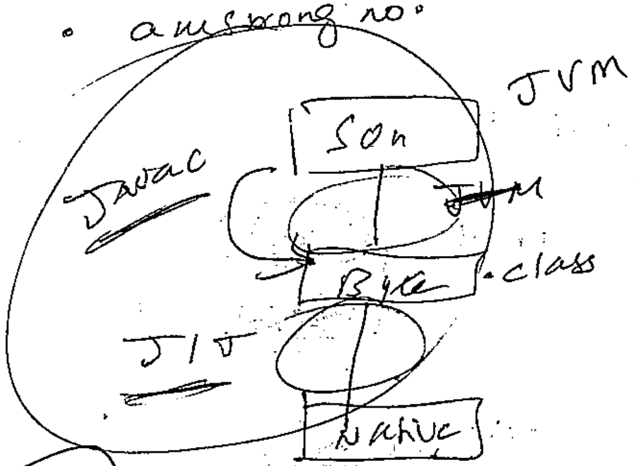
A1

class B

{ B1

B2

}



C1

interp

m1()

{ vari

class

{ vari

} }

[illegible]

Java.

8119323640

8509653083 → CNF

Siva

x = 9

a = x++	a = 9	x = 10
b = x--	b = 10	x = 9
c = ++x	c = 10	x = 10
d = x--	d = 10	x = 9
e = x++	e = 9	x = 10
f = ++x	f = 11	x = 11

CNF

CNF

CNF

Jaum(()

m1()

m1()

m1()

20

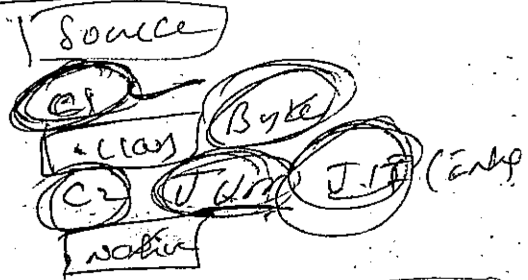
x = 9;

x++

x = x+1

a = x;

a = x++ + ++x + ++x + ++x



Byte code Same for all platform

JVM dependent on platform

first compiler dependent on (C1) platform

SRINIVASA