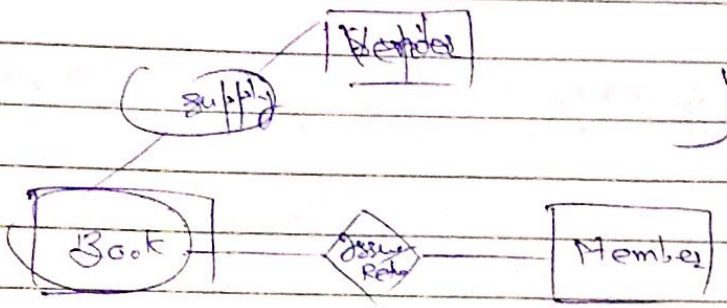
OOP

→ Reusability of code.

```
struct date
{
  short dd : 5
  short mm : 4
  short yy : 7.
}
```



UML → Unified Modelling Language.

02/08/17

Software development fundamentals / process

System analyst

Steps

(1) Requirement gathering / survey / study
→ record them using tools like symbols, UML

Engineer Architect
↑
Project Manager
↑

(2) Feasibility study (Analyse the ~~info~~ feasibility of requirements)
→ Technical
→ Financial
→ Operational

System Analyst
↑
code/debug — Programmer / Tester
↓
find exceptions in code / problem statement

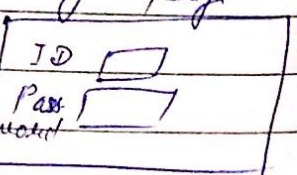
Technical: whether resources are available.
Operational: Functionality and utility report of product.

(3) System Design.
Imp. {
→ Input design / Form
→ Process design / Algorithms
→ Output design / Report
→ Database design / File design (storage)

Input → Process → Output

Input : Language for input
Process : ~~Validation verification~~
Output : present the result to user
I/O : UI (User Interface)

Log in page



Validation and verification

Validation: whether the info is allowed / permissible
Verification: Matches with the data in file.

Pie chart using C

224 - CSE   48 - ECE/ME/...
121 - ECE

Recruitment Ranking
Progs. Language

* User Interface is important and comes under Feasibility Study
UI is designed so as to make users feel comfortable.

Project Manager
→ uses mathematical tools to schedule projects
→ sequencing of projects in software development
→ not very technical
→ CPM (Critical Path Method) chart : if any crisis arises how to handle it (if any programmer quits the project)
→ Req. of resources, monetary assignments : chart.

Engineer architect
→ Predict long-term problems and issues and get solutions
→ research (data analytics, parallel processing,

⇒ Waterfall model
all events happen in a sequence and we can't repeat any finished event

~~Structured progs. lang.~~
- Sequencing
- Selection
- Iteration

structure chart : function calling another function
→ required for maintanence

* functions not returning anything are preferred.
* min. dependencies
* if there's a bug it will affect only a single f^n.
cohesion and coupling
↓ high, ↓ Loose.
even if it's a single line f^n, make it.

→ we can iterate any f^n any numbers of times till implementation is completed

⇒ structured chart

bottom up : functions then main
develop subsystem and integrate it

→ bottom up method - we don't have any idea about the whole view, big picture just by making small modules

⇒ coding is based on systematic approach

⇒ debugging trivial testing

Testing
→ 15% - 20% of time

↳ Unit testing (every f^n is performed properly as expected)
↳ Sub system Testing
↳ System testing
↳ Quality Assurance testing

Robust
→ Linux is a robust software
→ Robust programming should take care of exceptions and not crash even in unexpected cases

```
main()
{
}
```

it should give warning proper messages and pop-ups

```
main();
```

Sub-sys

→ integrate testing
one module developed by one person
and another module developed by other person

Sys.

→ whole integrated module testing / string testing

Quality Assurance testing

→ done by organizations / certificate
→ value increases for the product post this testing
→ gives certificates
→ STTPS : security.

09/08/17

User Acceptance testing

→ Demo of product to the user / client.
→ Signature for agreement → Implementation can start

Implementation / Deployment

→ Direct changeover : Replace the old system with a new system.
→ User training, designing user manual
→ Platform dependency : In case of software and hardware requirements must be specified.
→ configuration should be provided
→ Pilot run
→ Parallel run.

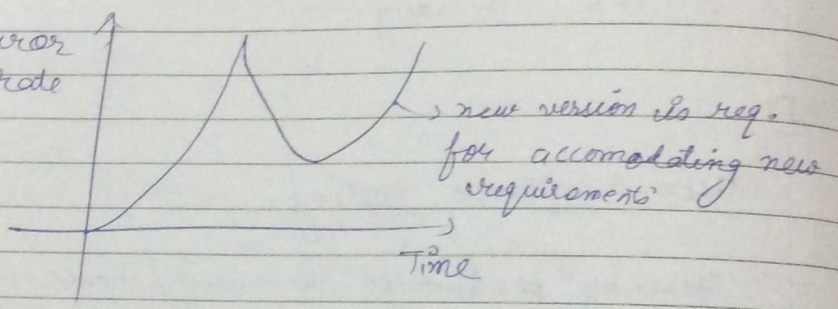
COM
PEER

Pilot run: deployed on hypothetical data and not on actual data for a specified time duration so that actual data isn't affected.

→ real time system: railway reservation, satellite

Parallel run: both the existing and new systems run simultaneously.
not possible in case of real-time system.

⇒ schedule of implementation is also prepared by product manager



→ Post implementation maintainence.

* Logic and functional programming used in AI (LISP/Prolog)
* Small talk (first pure OOP)
* JAVA is strict in its grammar.

* structured prog. (C) [illegible] ... functional progs ...

OOP concepts

Object Oriented paradigms

(1) Class and object
(2) Encapsulation
(3) Data hiding Abstraction
(4) Polymorphism
(5) Inheritance
(6) Resistance

If any progs. language uses these concepts is called OOP if it provides struct. for implementation.

1. Class and object
→ Object has its existence
→ Collection of similar types of objects is called a class
→ Object is one instance of class.
→ Data-type : Domain
operations performed on this data type.
→ we can not standardize some data types. eg. student, book

2. Encapsulation
→ put all the properties for a class in one place,
the operations
the data
→ We can achieve data hiding using this as we can specify that only these operations can access the data.

3. Data hiding
→ achieved using encapsulation

* can't be done using C language - encapsulation

Camlin Page
Date 11 / 08 / 17

(3) Encapsulation Abstraction
we know important details and other details are hidden.
→ signature knowledge / signature object function - important details.
→ definition of function of a class can't be known by other programmers.
→ achieved using encapsulation.

4. Polymorphism
→ in C we can do using pointer to function

add(a, b)
(a, b) rational
(c, d) complex

⇒ Function overloading
⇒ operation overloading
⇒ dynamic binding

Function overload: Using OOPs we can have functions with same name
→ using this we won't have to remember large no. of functions.
→ flexibility

Dynamic binding: → distinction is made by no. of parameters and types of parameters.

→ we can extend the domain of an object.
$r_3 = r_1 + r_2$ (& on matrix, string)
→ not supported by JAVA
→ supported by C++

IEEE 754

float no representation

→ design class
→ write code in JAVA

* Math class all mathematical functions

Math.pow(10,1)

5. Inheritance

→ hierarchy of classes.
→ general properties
→ specific properties

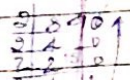
6. Persistence

If an object persists in memory, even if we come back later to that application

→ DBMS supports it

Ques Write a program to convert decimal into binary using bitwise operator.

5:4 →

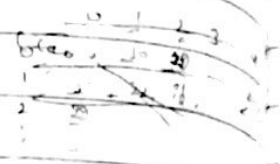


```
class convert
{
    public static void main(String args[])
    {
        byte a=23;
        long b;
        while(a!=0)
        {
            b=(a-a>>1)*x
        }
    }
}
```

Camlin Page
Date

Ques Write a program to find the transpose of a matrix.

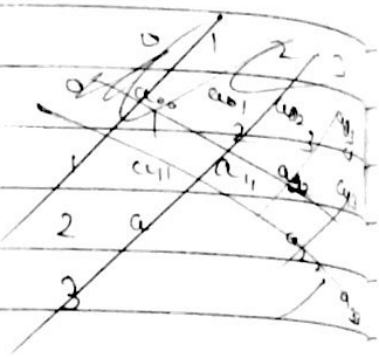
```
public transpose
{
    public static void main (- -)
    {

    }
}
```



a[n][n]

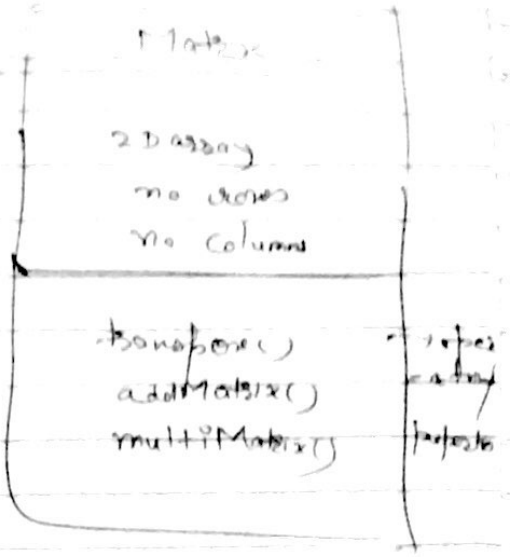
a_{11} a_{12} a_{13} a_{14}
a_{21} a_{22} a_{23} a_{24}
a_{31} a_{32} a_{33} a_{34}
a_{41} a_{42} a_{43} a_{44}

```
for (i = 0; i < n; i++)
    for (j = i+1; j < n; j++)
    {
        temp = a[i][j] = a
        a[j][i] = a[i][j]
        a[i][j] = temp;
    }
```

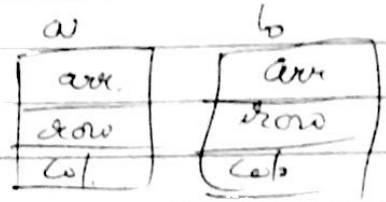


→ can be used as a datatype

```
class Matrix
{                    → object
    int arr[][];     → NULL
    int row, column;
    void
    public initialize(int r, int c)
    {
        arr = new int[r][c];
        row = r;
        col = c;
    }
}
```




```
Matrix a;        → object
                 → NULL
a = newMatrix()
a.initialize(3,4)

Matrix b = newMatrix
b.initialize(5,5);
Matrix c = a.transpose();
```




```
public Matrix transpose()
{
    int i,j;
    Matrix t = newMatrix();
    t.initialize(col, row)
    for(i=0; i<col; i++)
        for(j=0; j<row; j++)
            t[i][j] = arr[j][i];
    return t;
}
```

* no automatic initialization, gives error.

Camlin | Page
Date / /

Ques) Write or Design a class named as Rational. Use this class to perform basic operations on rational numbers.

```
Class Rational
{   int nemo.
    int deno
    public void initialize()
    {
        nemo = 0;
        deno = 1;
    }
    public void initialize(int n, int d)
    {
        nemo = n;
        deno = d;
    }
    public void initialize(int n)
    {
        nemo = n;
        deno = 1;
    }

    public Rational addRational(Rational r)
{
    Rational t = new Rational();
    t.nemo = r.deno * nemo + deno*r.nemo;
    t.deno = r.deno * deno;
    int g = gcd(t.nemo, t.deno);
        t.nemo = t.nemo/g;
        t.deno = t.deno/g;
}
```

```
public void displayRational()
    {
    System.out.println(nemo + "/" + deno);
    }
}

class Rtest
{
public static void main(String args[])
    {
        Rational r1, r2, r3;
        r1 = new Rational();
        r2 = new Rational();
        r1.initialize(3,5);
        r2.initialize(2,7);
        r3 = r1.addRational(r2);
        r1.displayRational();
        r2.displayRational();
        r3.displayRational();
    }
}

int gcd(int n, int m)
{
    if(n%m == 0)
    return m;
    else
        return gcd(m, n%m);
}
```

Creates r_1 and r_2 in memory physically.

→ class file is created in Java.
→ save as Rtest.java
→ no need to do extend
→ we should decompose the problem in smaller parts.
→ when any object is connected to

Constructors

Rational r1 = new Rational();

r1 → referring → num, deno

- Same name as class
- It's very specific method of class
- It never returns any value
- Constructors are used to create the object
  - allocate memory
  - initialize the object with its values
- They are called automatically

Default Constructor

```
Rational()
{
    num = 0;
    deno = 1;
}
```

not receiving any parameters

⇒ num, deno → instance variables

object is an instance of class.

⇒ between instance & local variables, compiler gives preference to local variable.

local variable

```
public Rational (int num, int deno)
{
    num = num;
    deno = deno;
}
```

→ instance variable are still uninitialised

* 'this' is a keyword in Java
  It always keeps the track of current object / invoking object

```
r1.displayRational();
```

'this' keeps track of r1 as r1 is the invoking object

```
num = n;
deno = d;
```

```
printf("%d/%d", num, deno);
```

'this' specifies num & deno to r1 as it's the referred object

* we can't have 2 methods with same no of parameters and same type of parameters with the same name

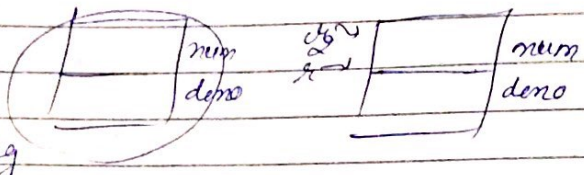
```
public Rational (int num, int deno)
{
    this.num = num;
    this.deno = deno;
}

public Rational (int d, int n)
{
    this.num = n;
    this.deno = d;
}
```

not allowed in method overloading

$r_3 = r_1.addRational(r_2)$;

- no duplicate is made / no fresh copy is made
- r_3 is referring to the same location
- local variables in a method is not destroyed if a reference is existing for those objects.

$r_1 = r_2$



dangling block.

garbage collection mechanism in Java

- Memory optimization is automatic by garbage collector which is active as long as the program is being executed.
- the dangling blocks are freed.

- If we haven't written any constructor in our class the compiler calls its own default constructor.

- In a cl
method
- Create stack
- Create Inde

Node
integer s nu
Node s ne
Create Node
ddd to Node
set Value()
get Value()
search Value(

- is
kind
aggreg
- part
has a
contain

Linked list

Camlin Page
Date 23 | 09 | 17

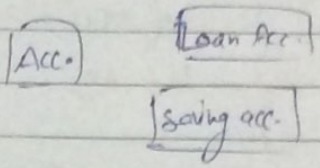
→ In a class, we can call one method in another method

→ Create stack in Java.
→ Create linked list.

Node
integer : num Node $ next
Create Node() deleteNode() setValue() getValue() searchValue()

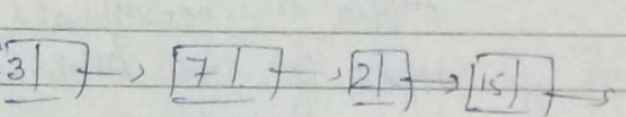
aggregation:
one class contains objects of other class

Inheritance



→ is a kind of aggregation } specialization or generalization

- part of
- has a
- contains



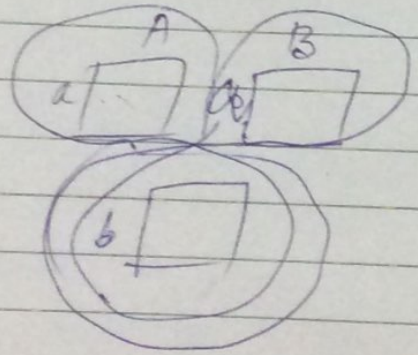
Static in java

→ Instance variables are object variables (created and stored by all objects)

→ Static variables are class variables / not the property of object

```
class Alpha
{
    int a;
static int b;

    void putAlpha()
    {   a=10;
        b=20;
    }
}
```

```
Alpha A = new Alpha();
Alpha B = new Alpha();
```



```
static
{
    b=20;
}
```

→ static variables are initialized in a static variable.
→ static variables are initialized with zero if not by the user.

→ we don't declare instance variable as public.
→ If the access specifier is not specified then default is used (private, public, protected)

→ every class is a part of some package.
→ we can access the static variable using object reference but we access it using class reference

```
public static void main(String args[])
{
    Alpha a=10;
    Alpha b=20;
```

→ eg. If we want to keep track of times of login then we make counter using static variable

eg. interest rate in a bank: static variable.

* JVM → Java Virtual Machine

$ java Test ↵

* JRE bifurcates the RAM.

JVM → to invoke → call main line

→ retrieve class from memory

⇒ JVM calls main without object reference

Class containing main method.

The the reason:

main is static

→ Any static method can't access non-static properties of same class

→ It can't call non-static methods.

```
class Test
{ private int a=10;

  public static void main(String args[])
  {
      a=20;          X     Test t= new Test();
      put_a();             t.a=20;
                           t.put_a();   ✓
  }

  void put_a()
  {
      S.o.p("a=" + a);
  }
}
```

Access specifiers (private / protected / public / default)

In the same class access specifiers don't make any difference.
→ we can call any type of method
→ access any properties

→ Private ∧ can't be accessed outside the class.
→ scope is within the class.

```
class Alpha
{
default ← int b;
    void putAlpha()
    {
        a = 30;
        b = 20;
    }
}
```

If 'a' were to be a default variable still we can't access it in other class without object reference.
Although we can access properties of same class without object reference.

→ Public is accessible outside program.

Scanner

java.util. → package

if package name is specified then it's default package.

→ java.lang → default package contains predefined classes

import java.util.Scanner;

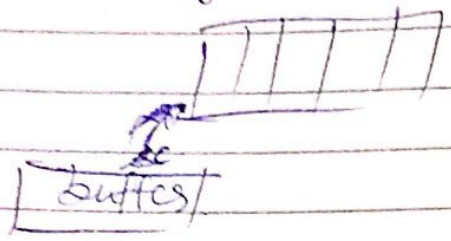
import java.util.*
(all classes are included)

→ Math is a static class

Camlin Page
Date / /

Constructor call → Scanner sc = new Scanner(System.in)

Request to Operating System.
It provides link with the keyboard
A buffer will be created inside the RAM with the name System.in



To access the value in buffer, there are methods

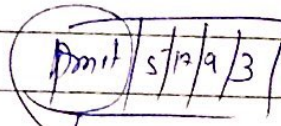
sc.nextInt()
sc.next()
sc.nextFloat()
hasNext()

if sc. (integer specified) → execution will stop

→ String is used to receive string

→ String is in the package java.lang class.

while (sc.hasNext()) → true/false.
{
a[i] = sc.nextInt();
}



Removed from the buffer

till 10 pm.

pixel :- colour
brightness
sharpness
*contrast

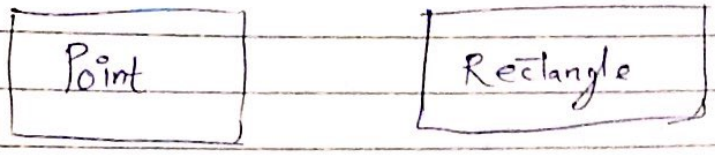
1024 x 1024

JPG MPEG

→ Compress
→ decompress

30/09/17

Inheritance



Point is 'a part' of rectangle.
Rectangle 'has a' point

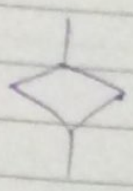
Book 'has an' author

→ IS-A or kind-of (Generalization or Specialization
→ Has-A or Part-of or Contains (Aggregation or Containership)

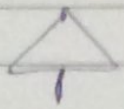
→ Private properties can't be Inherited

Inheritance represents the IS-A relationship also known as parent-child relationship

1. Reusability
2. Extendibility



Aggregation



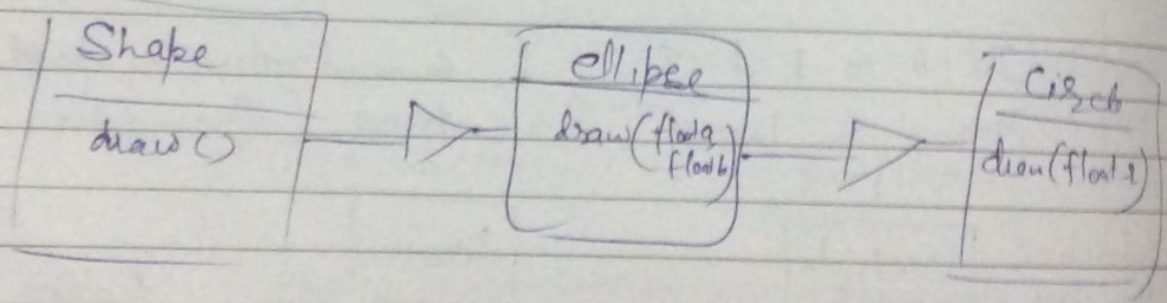
Inheritance

→ In inheritance, we aren't concerned about the super class but the derived class.
In aggregation, existence of both the classes are independent.

→ Method overloading

→ Method overriding

The definition for a method in superclass and derived class can be different although the method name is same. The definition can even be extended in ~~defini~~ derived class.



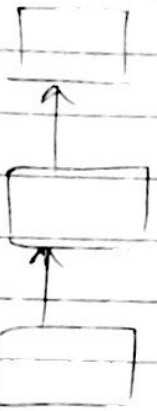
Can a class extend more than one classes

Camlin Page
Date / /

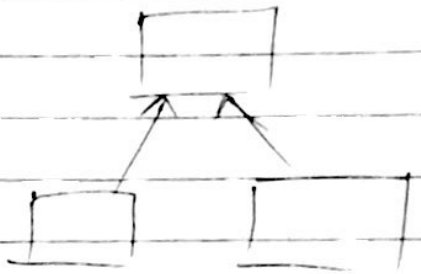
dynamic binding
or run-time binding
or late binding

Types of inheritance

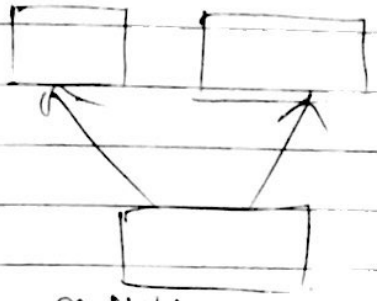
single



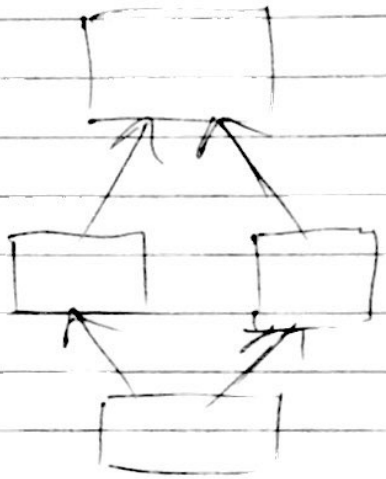
multilevel



hierarchy



Multiple



hybrid

Monday

Camlin Page
Date / /

1. Products
Services
Member
Membership : Premium / Gold / Silver

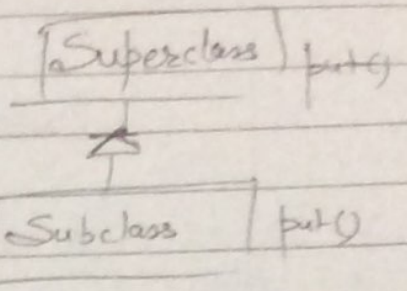
Products Services

Member

Premium Gold Silver

Dynamic binding

Superclass sp = new Superclass();
Subclass sub = new Subclass()



sp.put() → method in superclass

sp = sub; → Superclass can refer to subclass object

sp.put() → method in subclass

sub = sp; ✗

→ method overriding occurs

→ Instance variables can't be over-ridden in subclasses

→ Static method can't be overridden.

Final methods can't be overridden

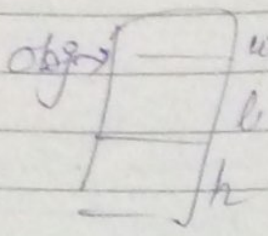
final put() → superclass

void put() → sub class.
{
super.put();
}

→ Any method can't initialize final variable except constructor

→ newHonda().run()
↳ anonymous object is created.

→ There is no address in Java but reference (memory)



hexadecimal number

toString()

→ Object class is the superclass of all the classes. It has a method toString()

String toString() → override this method in class box. It'd be automatically called in System.out.println

{
return width + " " + height + " " + length;
}

println accepts only string

(1)

```
class box
{
  int width;
  int height;
  int length;
}
Class mainclass
{
  public static void main(String args[])
  {
    box obj = new box();
    System.out.println(obj);
  }
}
```

Java: but, reference memory
hexadecimal number

... of all the classes
...

...side this method in class box. It'd be automatically called in System.out.println(obj)
length.

(2)

```
void meth ( test o)
{
    o.a *= 2 ;
    o.b *= 2 ;
}
```

obj.meth(obj) → call be reference

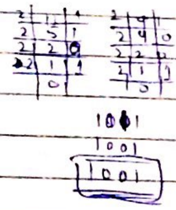
(3)

```
class Main
{
    public static void main ( String args[])
    {
        System.out.println( fun());
    }

    int fun()
    {
        return 20 ;
    }
}
```

→ It should be static or fun() should be called by object reference.

(4)

```
Class Bitwise
{
    public static void main ( String args[])
    {
        int x = 189 ;
        int y = x ^ 3 ;
        System.out.println (y|12);
    }
```



(5)

```
class Test
{
  public void display (int, double b)
  public void display (int a, float b)

  t.display (10, 5) ---> by default it's converted to double
```

(6)

```
public class A
{
  void A()  ---> compile time error as constructor can't have void
  {
    System.out.println ("This is A");
  }

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

(7)

short = 0x8000; ✗

it can't be stored in short.

0x8000

0x → 8000

1000 0000 0000 0000

-2^{15}

2^{14}

Comlin
Date / /

10+4=14 ⟹ type cast

4×10=40

14+(49/7) = 14+13 ... 27.26

17×7 ... 26 ... 56

26×7=182 ... 26 ... 182 ...

Camlin Page
Date 11/9/17

Abstract Class

→ If we are not able to define any class or method in a class then the class is called abstract

→ If a method is abstract then the class must be defined as abstract

→ we can't instantiate an object from an abstract class

→ The skeleton of the abstract class can be useful when we are working on some project and we need the property & method

```
abstract class shape
{
    int color;
    abstract double calArea();
}
```

It's a must to define abstract method in derived class

```
class Rectangle extends shape
{
    double length;
    double width;
    Rectangle( double length, double width)
    {
        this.length = length;
        this.width = width;
    }
    double calArea()
    {
        return length * width;
    }
}
```

derived class can also be abstract

abstract class Triangle extends Shape → calculation of area is not generalized as we can use base/height (right Δ), eq. string

```
{
    abstract double calArea();   → either define fully we have to define it again
}

class RightTriangle extends Triangle
{
    double base;
    double height;

    double calArea()
    {
        return (1/2) * base * height;
    }
}
```

→ sending message to different subclasses of superclass is called dynamic dispatch.

Camlin Page
Date 13/9/17

Garbage Collection / Deallocation of dynamic memory

Student *ptr = malloc(10); } free(ptr);
Student *str = new Student(); } delete str;

→ If an object is explicitly assigned as null, the garbage collector is invoked

(1)
```
Class Test
{
    psvm(String args[])
    {
        Object obj = new Object();
        obj = null;
    }
}
```

(2)
```
Object obj = new Object();
Object obj1 = new Test();
obj = obj1;
```

(3)
```
Object obj = new Object();
obj = new Test();
```
→ can be declared

(4)

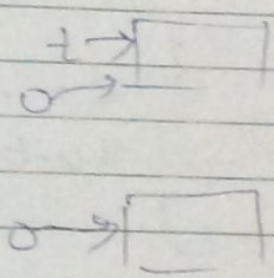
```
Object obj = new Object();
Object obj1 = new Test();
obj.change(obj);
```

```
P.V change(object o)
{
    Object t = new Object;
    t = ...
}
```

local object is created and it'll be destroyed after the method is over

(5) 2

```
public Object change(Object o)
{
    Object t = new object();
    o = t;
    return o;
}
```




```
Object obj = new Object();
obj = obj.change(obj);
```

~~t is not~~ For the memory space assigned to t, object reference still exists so the memory space is not dangling

3

```
obj.change(obj);
```

↓ not receiving

then the garbage collector is invoked for memory block assigned to t after the method ends

(5) Anonymous object

new Object();

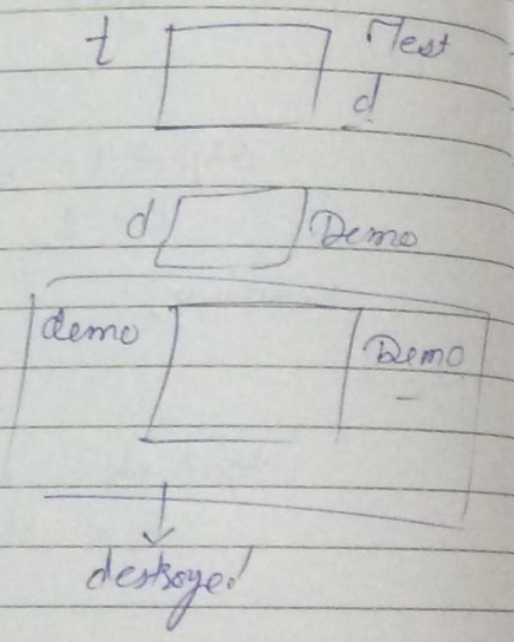
If in the constructor

```
Object()
{
    this. ...
}
```

→ some other reference then won't be destroyed

```
Class Test
{
    private Demo d;
    void start()
    {
        d = new Demo();
        this.takeDemo(d);
    }

    void takeDemo(Demo demo)
    {
        demo = null;
        demo = new Demo;
    }
}
```

Class object →

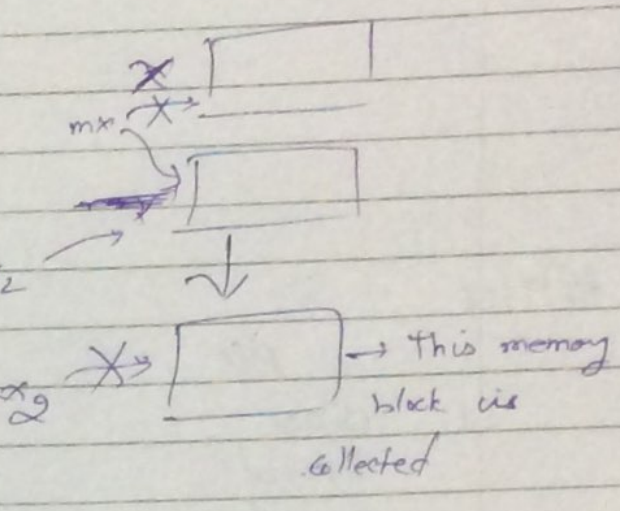


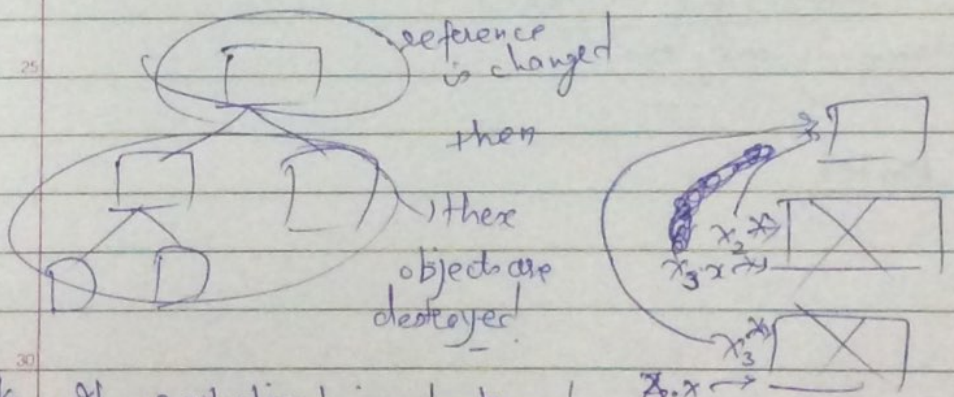
t object has instance variable d. of Demo class

~~t.start~~

We can't say about t object from this method as it might have called from main or some other class.

```
public class Alpha
{
  psvm (String args[])
  {
    Alpha x = new Alpha();
    Alpha x2 = fun(x);
    Alpha x4 = new Alpha();
    x2 = x4;
  }
  void Alpha fun (mx)
  {
    Alpha mx = new Alpha();
    return mx
  }
}
```





* If root object is destroyed then all objects of sub classes are also destroyed

protected void finalize()
{
sc.close(); — free the resources used by the object as method
}

resource → file
video
server

15/9/17

(1) dBase
Derived
Derived

(2) In Java, protected method can be overridden as public method.

Can't be more restricted.

→ We can't refer like
super.super.method()
If inheritance is multilevel

(3) G ≠ G is fronted