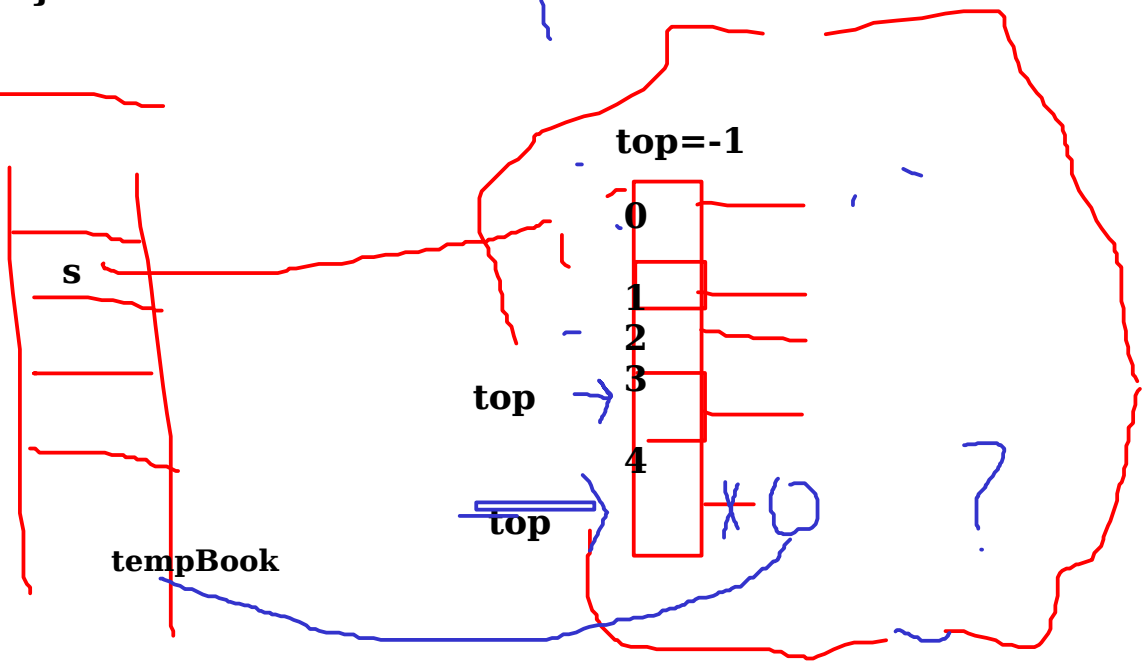
DAY -3&4

```
public class Stack {
    private final int SIZE = 5;
    Book arr[];
    int top;

    Stack() {
        arr = new Book[SIZE];
        top = -1;
    }
    public Book pop() {
        if (!isEmpty()) {
            return arr[top--];
        }
        return null;
    }

    public void push(Book book) {
        if (!isFull()) {
            arr[++top] = book;
        } else {
            System.out.println("stack is full");
        }
    }

    public boolean isEmpty() {
        return top == -1;
    }

    public boolean isFull() {
        return top == SIZE - 1;
    }
}
```

Memory leak?

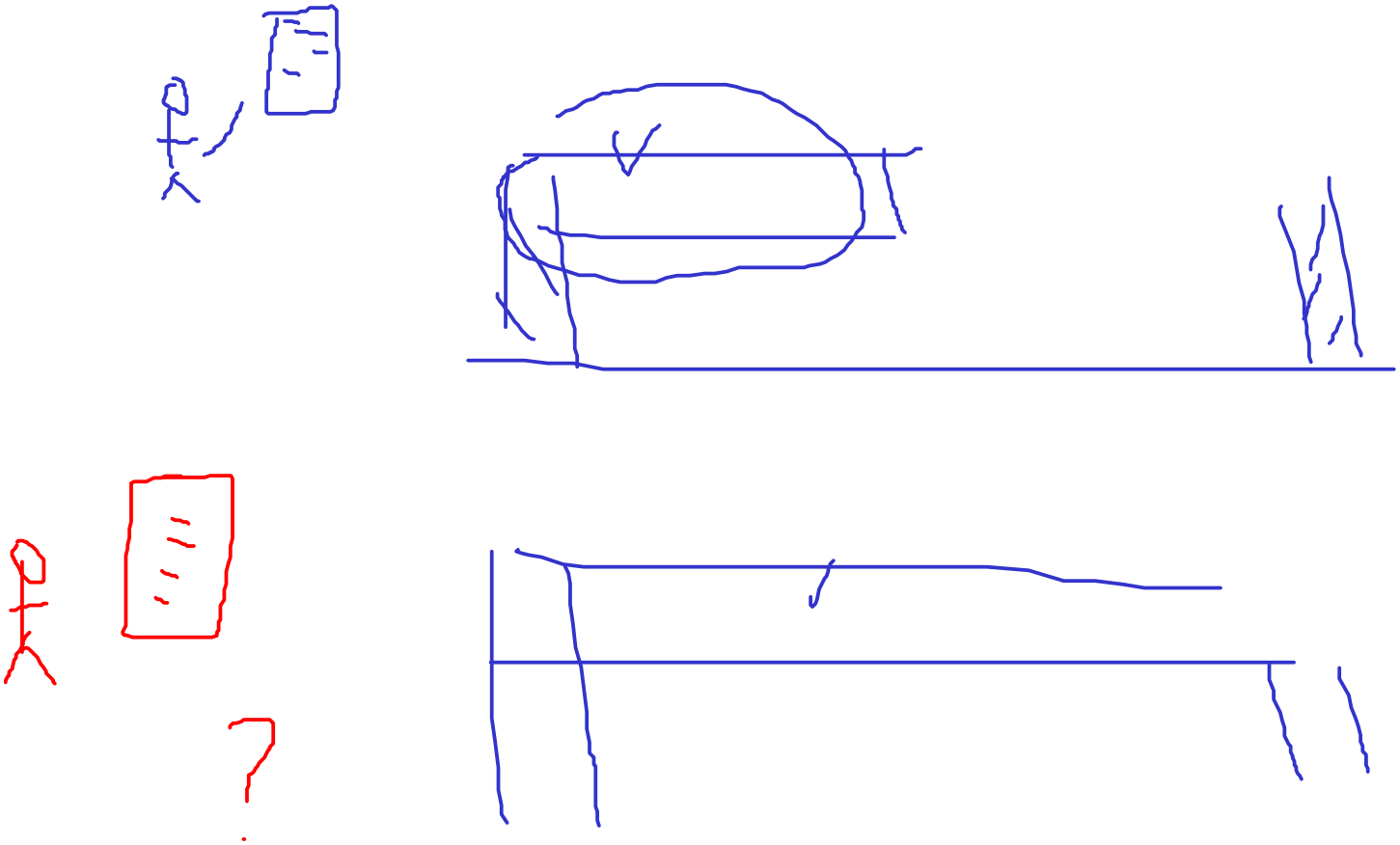
```
public Book pop() {
        if (!isEmpty()) {
                return arr[top--];
        }
        return null;
}
```

```
public Book pop() {
        Book tempBook=null;

        if (!isEmpty()) {
                tempBook= arr[top];
                arr[top]=null;// imp?
                top++;
        }
        return tempBook;
}
```

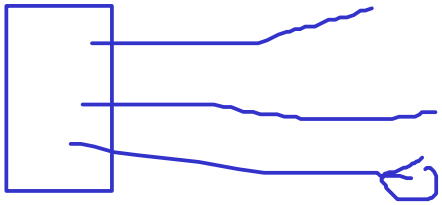



```
stack.push(new Book(121, "let us c", 300));
        stack.push(new Book(12, "thinking in java", 400));
        stack.push(new Book(199, "think and grow rich", 200));
        stack.push(new Book(17, "monk who sold farrari", 150));

        stack.push(new Book(7, "thinking in java", 10));
```

```
public void push(Book book) {
        if (!isFull()) {
                arr[++top] = book;
        } else {
                System.out.println("stack is full");
        }
}
```



Writing flexible sw is very imp...



Day 3: Advanced Class Features

- Loose coupling and high cohesion
- composition, aggrigation, inheritance, basic of uml
  - Abstract classes and methods
  - Relationship between classes- IS-A, HAS-A, USE-A
  - Interface Vs Abstract, when to use what?
  - Final classes and methods
  - Interfaces, loose coupling and high cohesion
  - SOLID principles, Square – rectangle problem
  - Hands On & Lab

A bit about UML diagram…

- UML 2.0 aka modeling language has 12 type of diagrams
- Most important once are class diagram, use case diagram and sequence diagram.
- You should know how to read it correctly
- This is not UML session…

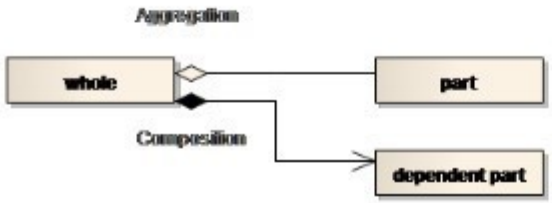
UML	Implementation

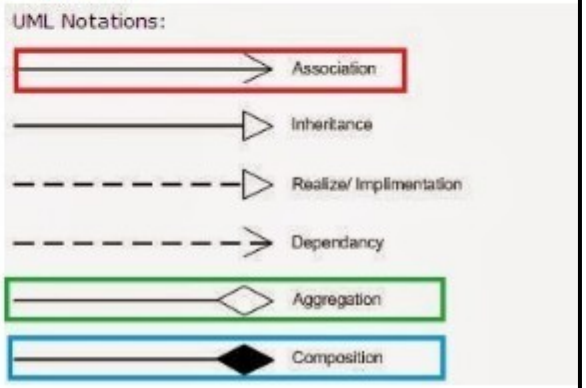
Person
- name : String
- age : int
+ Person(name : String)
+ calculateBMI() : double
+ isOlderThan(another : Person) : boolean
	`public class Person {`
	`  private String name;`
	`  private int age;`
	
	`  public Person(String name) { ...}`
	
	`  public double calculateBMI() { ... }`
	
	`  public boolean isOlderThan(Person another) {...}`
	`}`

rgupta.mtech@gmail.com

Relationship between Objects

- USE-A
  - Passanger using metro to reach from office from home
- HAS-A (Association)
  - Compostion
    - Flat is part of Durga apartment
  - Aggregation
    - Ram is musician with RockStart musics group
- IS-A
  - Empoloyee is a person



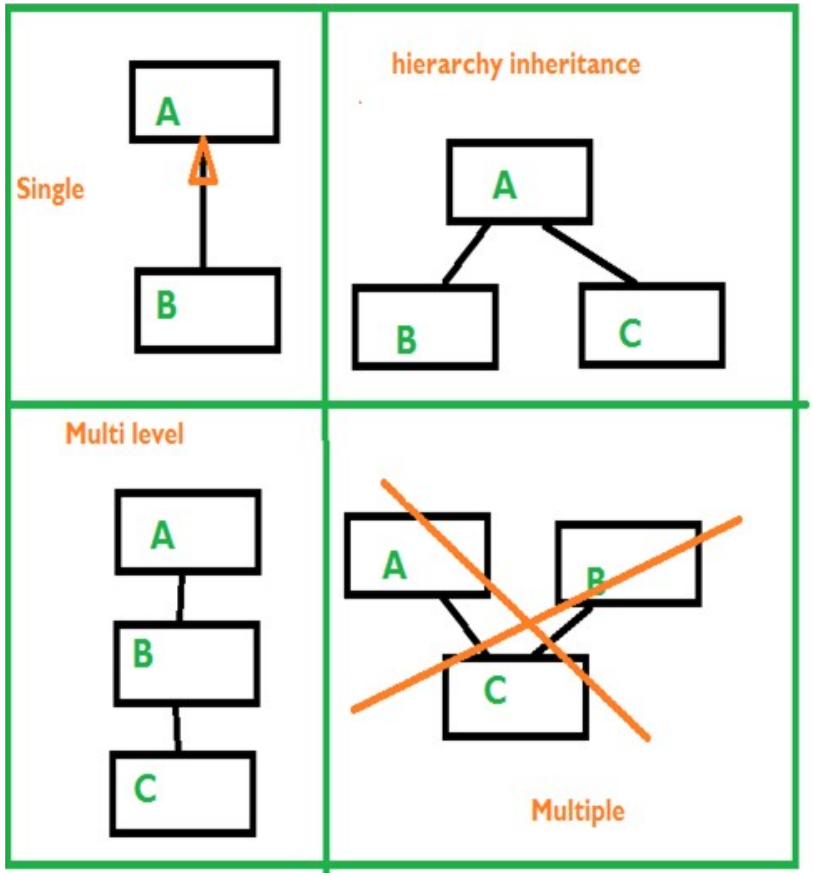


Inheritance

- **Inheritance is the inclusion of behaviour (i.e. methods) and state (i.e. variables) of a base class in a derived class so** that they are accessible in that derived class.

- **code reusability.**

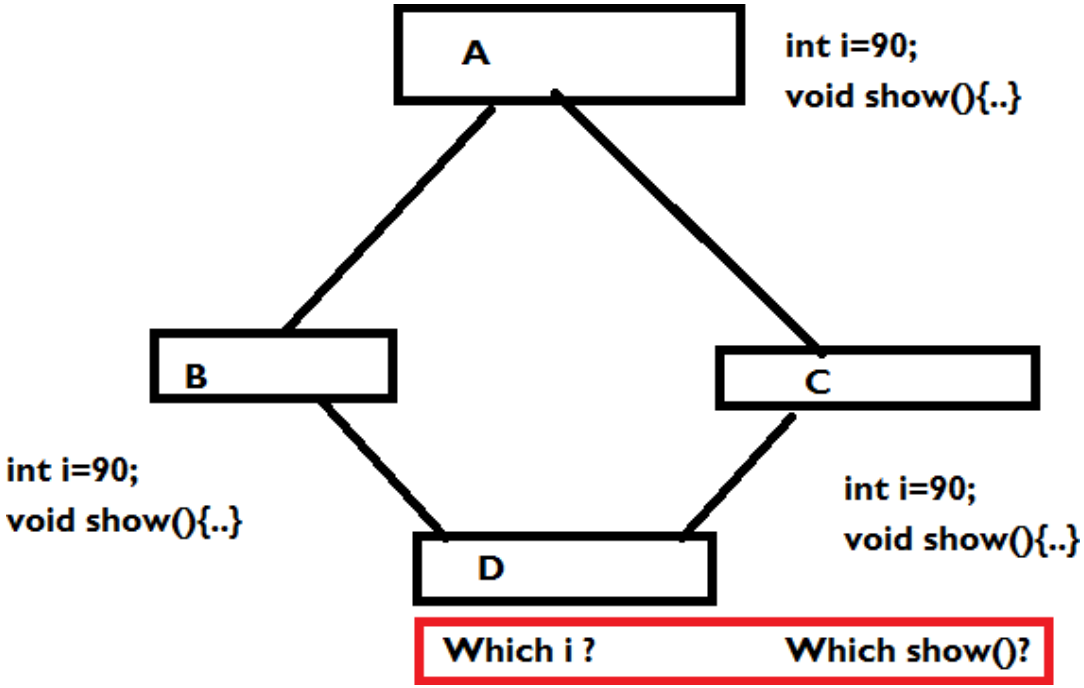
- **Subclass and Super class concept**



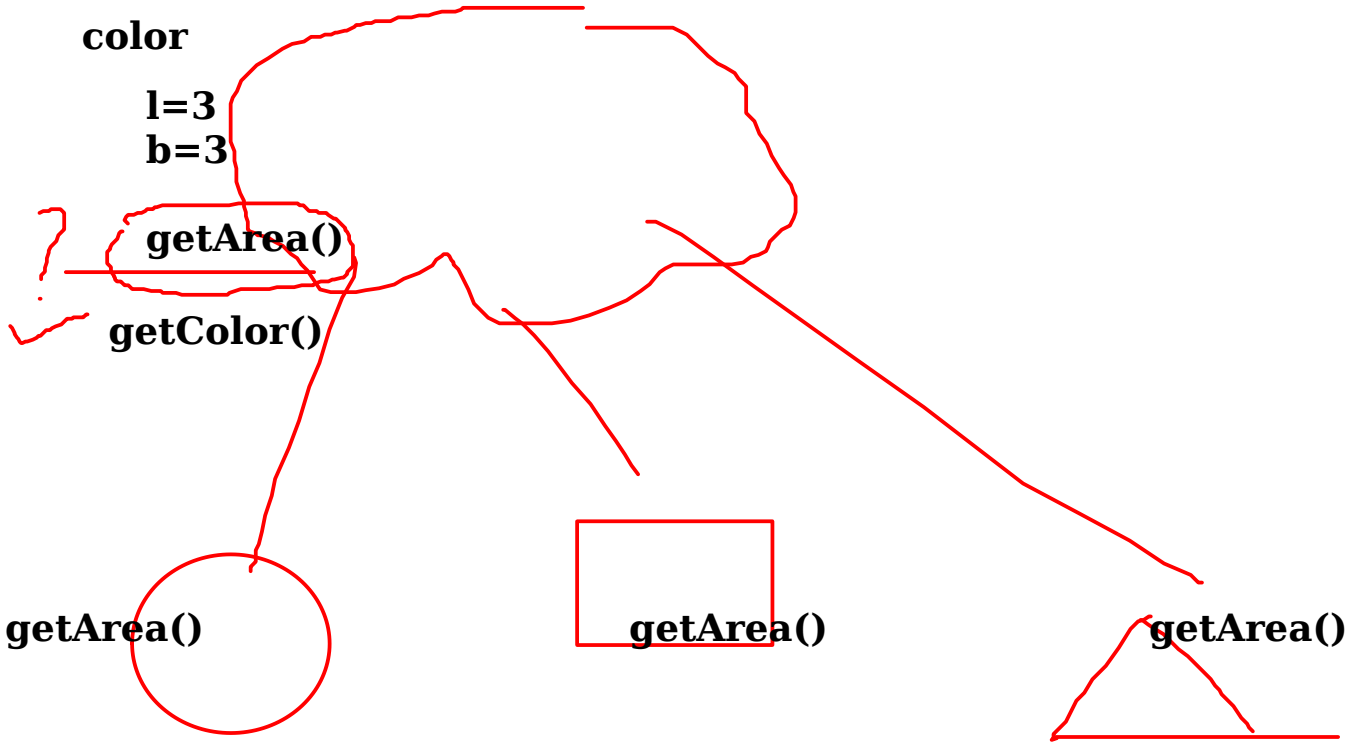
rgupta.mtech@gmail.com

Diamond Problem?

- Hierarchy inheritance can leads to poor design..
- Java don't support it directly…(Possible using interface)



rgupta.mtech@gmail.com



What is abstract class?

aka incomplete class?

```
abstract class Shape {

        private String color;

        public Shape(String color) {
                this.color = color;
        }

        public abstract void drawShape() ;

        public void showColor() {
                System.out.println("color : " + color);
        }
}
```

```
class Shape {

        private String color;

        public Shape(String color) {
                this.color = color;
        }

        public void drawShape() {

        }

        public void showColor() {
                System.out.println("color : " + color);
        }
}

class Rectangle extends Shape {
        public Rectangle(String color) {
                super(color);
        }

        @Override
        public void drawShape() {
                System.out.println(" it is a Rectangle.");
        }

}
```

Aman

mcq

```
abstract class A{
        void foo(){
        }
}
```

A class can be abstract class even if dont have any ab method but even it have one abs method then should be abstraction

You can only crete the type ie ref but not the object

```
abstract class Shape {

    private String color;

    public Shape(String color) {
        this.color = color;
    }

    public abstract void drawShape() ;

    public void showColor() {
        System.out.println("color : " + color);
    }
}
```

can have instance varible

can hv ctr

abs	interface
used to create common hierarchy	acting contract bw two modules
Account SavingAcc CurrntAccount	interface break the hierarchy*
instance variable common function	java 7: inteface was 100% abs
	java 8: u can also defind method
can have ctr can hve instance vari can have imp of method	u can not declare instance varaible
only u can not create object u can only crete ref	final static variable

Key word?

```
interface A{

}
```

```
interface A{
    int i=66;
}
```

public static final

GPP

```
interface A{
 public static final int i=66;
}
```

because we dont have instance varaible

ctr is of no use

```
interface A{
    default void A(){}
}
```

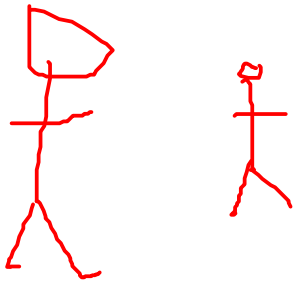
```
interface Foo{
    void foo();
}
```

```
interface Foo{
    public abstract void foo();
}
```

```
interface Foo{
    void foo();
}
```

extends

implements



```
inteface Foo{
    void foo();
}
 abstract class Foo2 implements Foo{

}
```

Why interface support multiple inheritance?

```
interface Foo{
        int i=66;
        void foo();
}
```

```
interface Bar{
        int i=66;
        void foo();
}
```

```
class C{

void demo(){}
}
```

```
class MyImp extends C implements Foo, Bar{

        public void foo(){
                Sysout(Bar.i);
        }
}
```

```
interface Foo{
        public abs void foo();
}
abstract class MyImp implements Foo{

}
```

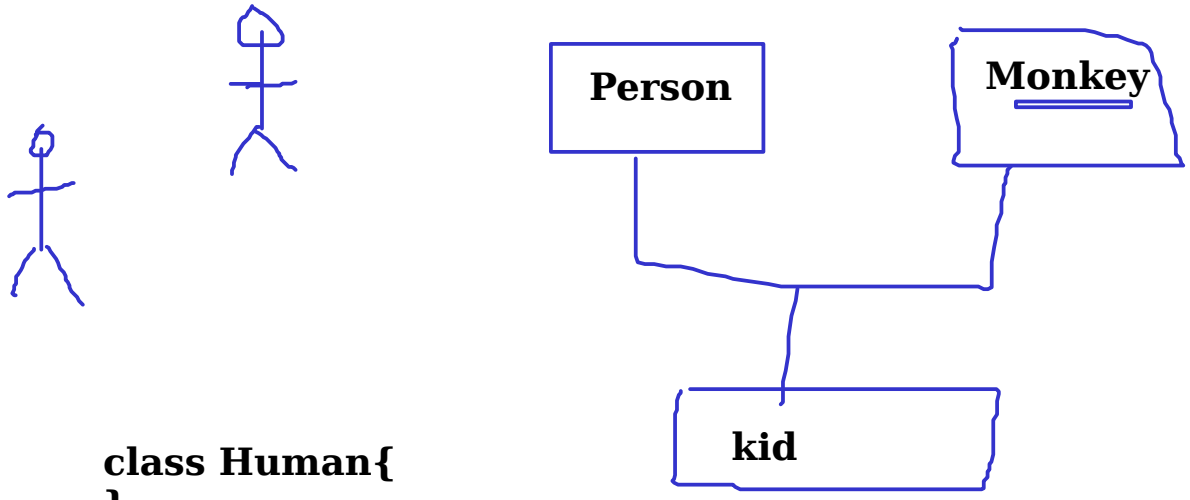
final class | normal class | ab class | interface

Note in java 8
we have diamond problem with interface*

interface break the hierarchy*

Abstract classes are for "is a" relationships and interfaces are for "can do".

Abstract classes let you add base behavior so programmers don't have to code everything, while still forcing them to follow your design.



jump()
stay()

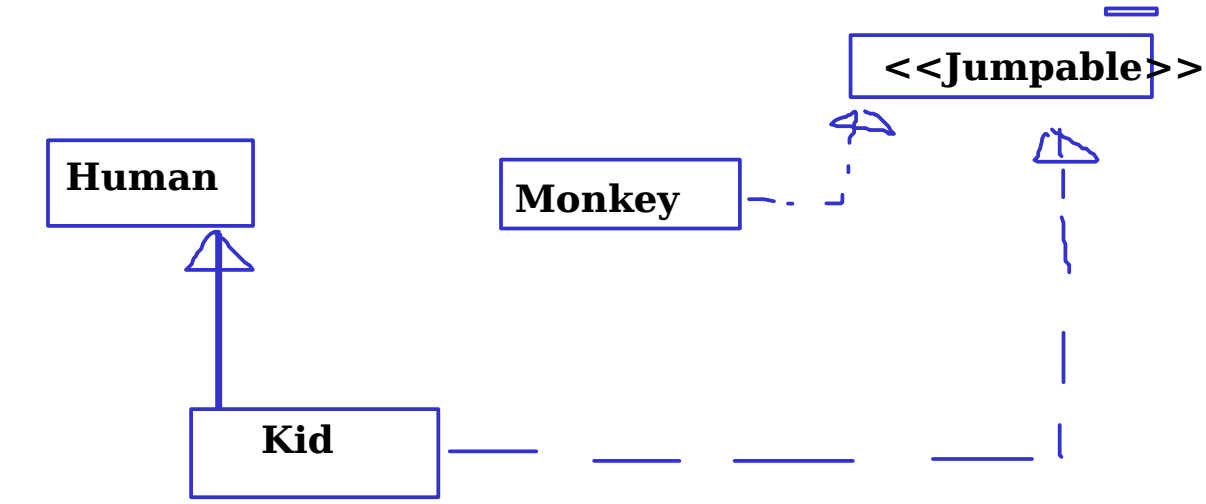
```
class Human{
}

class Monkey{
}

class Kid extends Human {
}

class Kid extends Monkey{
}
```

Runnable
Callable

Payment system:

Interface can improve the design of application

```
class Bird{
        public void eat() {
                System.out.println("eating ....");
        }
        public void swim() {
                System.out.println("swimming  ....");
        }
        public void fly() {
                System.out.println("flying........");
        }
}
class Eagle extends Bird{
        public void fly() {
                System.out.println("flying like anything.......");
        }
}
```



loose compling

Data transfer object

data access object

real life ex:

3 tier app! bankapp

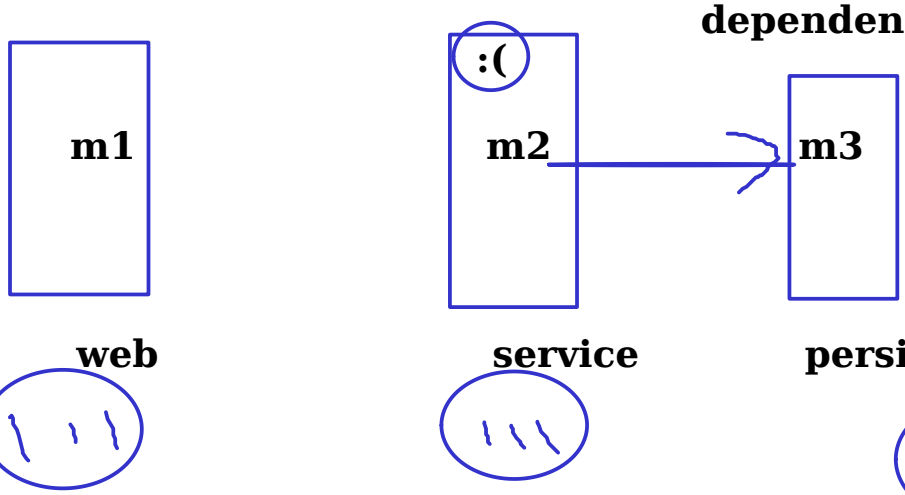
Design pattern

Dao / Dto

:)

dependency

:(



loose coupling:

sw module should

sahil

ravi

```
class A{
}

class B{
 A a=new A();

}
```

MVC
model view controller

B-----> A

Design pattern *

proven way of doing thing.....

In software engineering, a design pattern is a general repeatable solution to a commonly occurring problem in software design.
A design pattern isn't a finished design that can be transformed directly into code.
It is a description or template for how to solve a problem that can be used in many different situations.

Reuable comp of sw

head first design pattern

23 pattern GoF pattern
c++, java, python

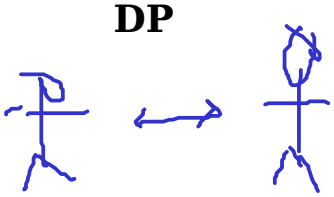
EIA design

spring patter

j2ee pattern

GoF

DP



Book store app

service layer
BL

data access
CRUD method

SOLID

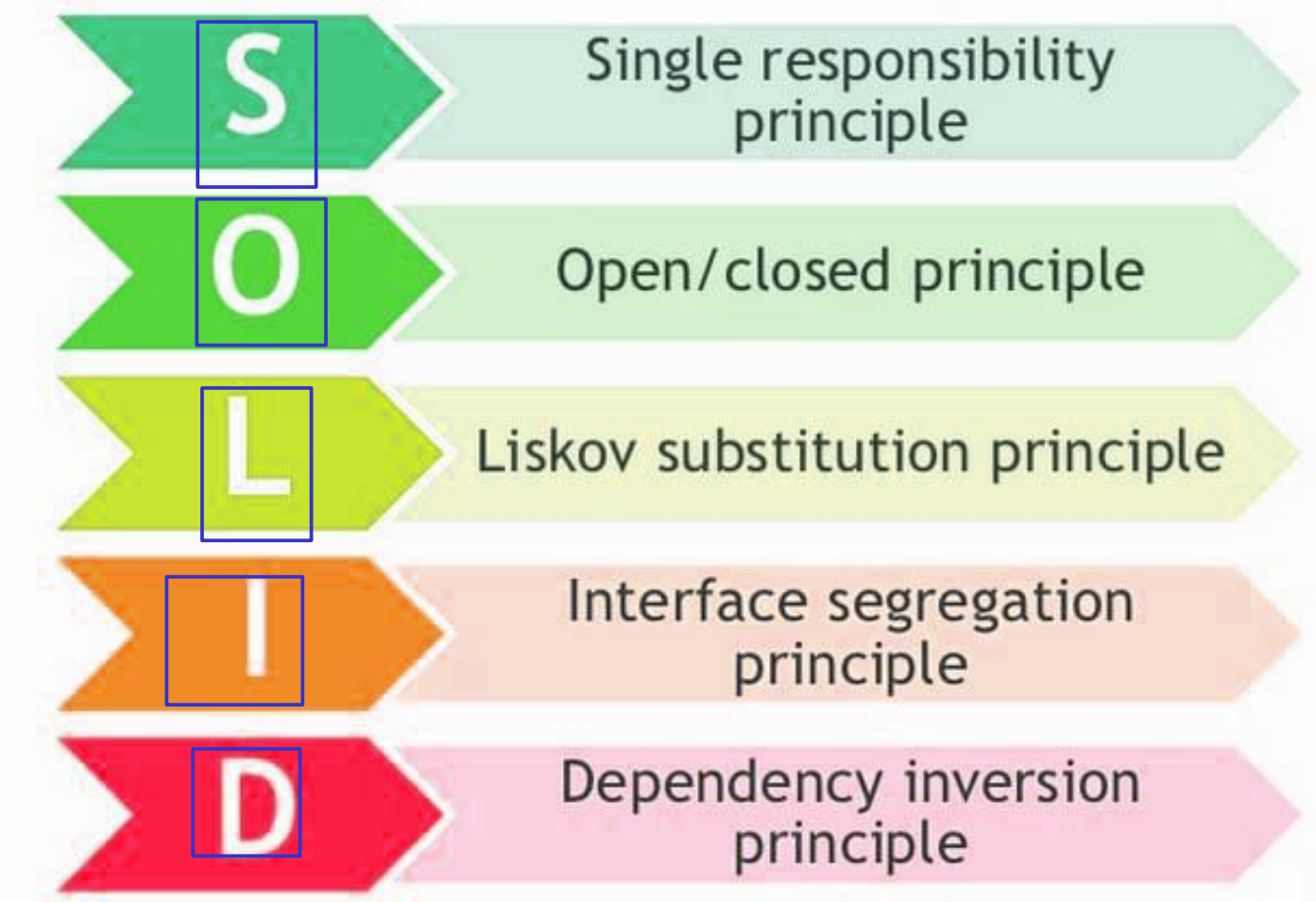
---

gof

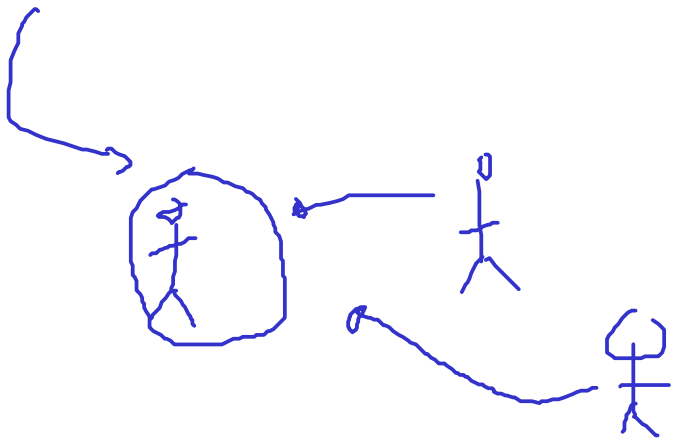
solid

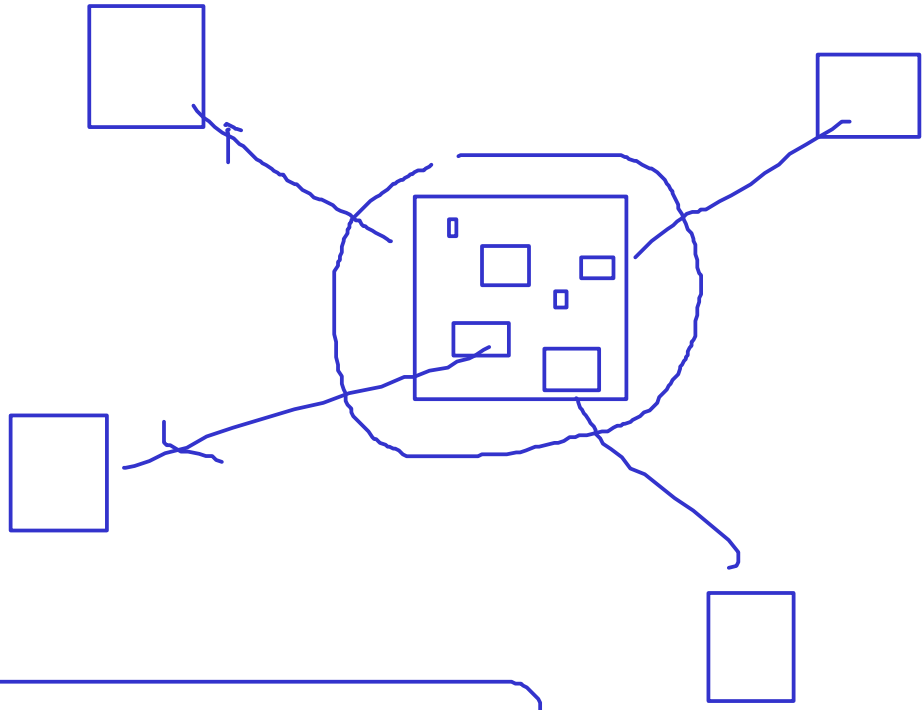
loose coupling and high cohesion

piller oo

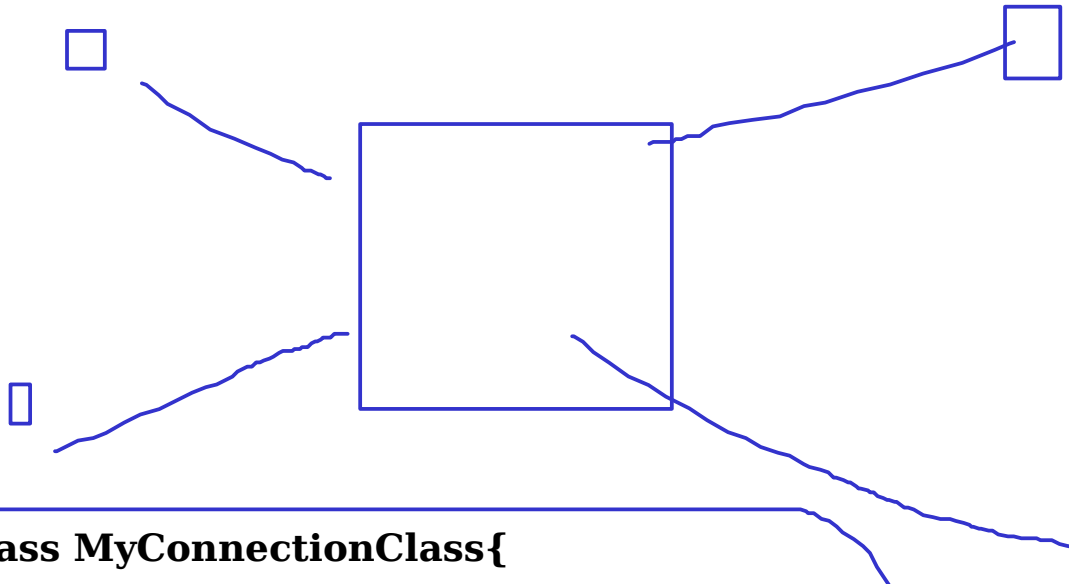


loose coupling and high cohesion





Never write the God class
Never write the bankruppt classes



```
class MyConnectionClass{
    public Connection jdbcConnection(){
    }
    public Connection jmsConnection(){
    }
    public Connection hibConnection(){
    }
}
```

:(

SOLID

loose coupling ....

oops

SRP (Single resp principle)

There should be only one reasion to change the class...

Open close prin...

a sw module should be open for extension
but close for modification!

code OCP

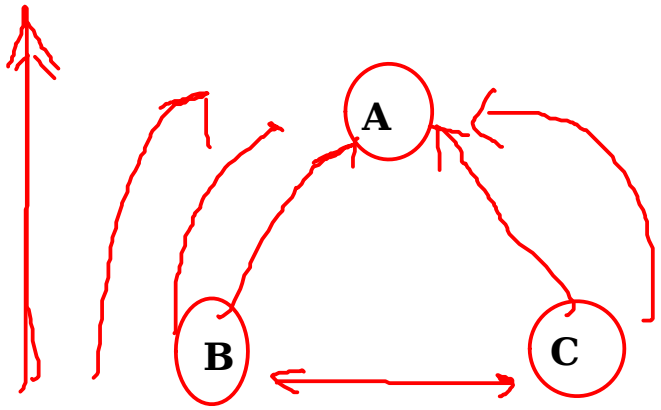
Inheritance example

- Use extends keyword
- Use super to pass values to base class constructor.

```
class A{
    int i;
    A()   {System.out.println("Default ctr of A");}

    A(int i) {System.out.println("Parameterized ctr of A");}
}

class B extends A{

    int j;
    B()   {System.out.println("Default ctr of B");}

    B(int i,int j)
    {
        super(i);
        System.out.println("Parameterized ctr of B");
    }

}
```

rgupta.mtech@gmail.com

Upcasting and downcasting



Upcasting

A a=new B();

object of drived class

ref of base class

A a=new B();

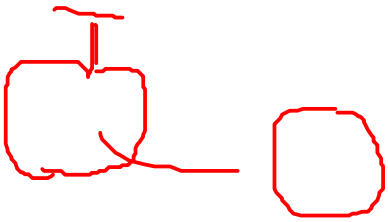
B b=a;

downcasting

B b=(B)a;

Shape s=new Circle();

ClassCastException



Overloading

- Overloading deals with multiple methods in the same class with the same name but different method signatures.

```
class MyClass {
    public void getInvestAmount(int rate) {…}

    public void getInvestAmount(int rate, long principal)
    { … }
}
```

- **Both the above methods have the same method names but different method signatures, which mean the methods are overloaded.**
- **Overloading lets you define the same operation in different ways for different data.**
- **Constructor can be overloaded ambiguity**
- ***Overloading in case of var-arg and Wrapper objects**…

Overriding…

- Overriding deals with two methods, one in the parent class and the other one in the child class and has the same name and signatures.

- Both the above methods have the same method names and the signatures but the method in the subclass *MyClass* overrides the method in the superclass *BaseClass*

- Overriding lets you define the **same operation in different ways for different object types.**

```
class BaseClass{
    public void getInvestAmount(int rate) {…}
}

class MyClass extends BaseClass {
    public void getInvestAmount(int rate) { …}
}
```

rgupta.mtech@gmail.com

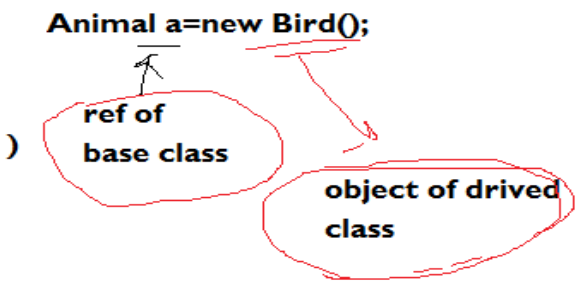
Polymorphism

- Polymorphism=many forms of one things
- **Substitutability**
- **Overriding**
- **Polymorphism means the ability of a single variable of a given type to be used to reference objects of** different types, and automatically call the method that is specific to the type of object the
- variable references.

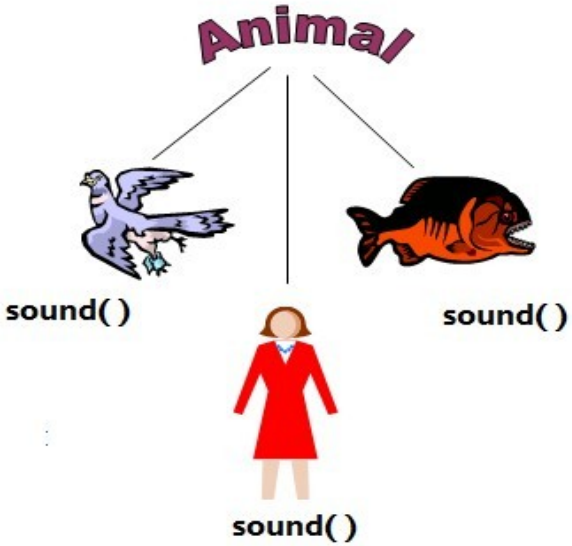
rgupta.mtech@gmail.com

Polymorphism

- Every animal sound but differently…
- We want to have Pointer of Animal class assigned by object of derived class



Which method is going is to be called is not decided by the type of pointer rather object assigned will decide at run time



rgupta.mtech@gmail.com

Example…

```
class Animal
{
    public void sound()
    {
        System.out.println("Don't know how generic animal sound.....");
    }
}
class Bird extends Animal
{@Override
    public void sound()
    {
        System.out.println("Bird sound.....");
    }
}
class Person extends Animal
{@Override
    public void sound()
    {
        System.out.println("Person sound.....");
    }
}
```

rgupta.mtech@gmail.com

Need of abstract class?

- Sound( ) method of Animal class don't make any sense ...i.e. it don't have semantically valid definition
- Method sound( ) in Animal class should be abstract means incomplete
- Using abstract method Derivatives of Animal class forced to provide meaningful sound() method

```
class Animal
{
    public void sound()
    {
        System.out.println("Don't know how generic animal sound.....");
    }
}
class Bird extends Animal
{@Override
    public void sound()
    {
        System.out.println("Bird sound.....");
    }
}
class Person extends Animal
{@Override
    public void sound()
    {
        System.out.println("Person sound.....");
    }
}
```

rgupta.mtech@gmail.com

Abstract class

- If an class have at least one abstract method it should be declare abstract class.
- Some time if we want to stop a programmer to create object of some class…
- Class has some default functionality that can be used as it is.
- Can extends only one abstract class

```
class Foo{
        public abstract void foo();
        }
```

```
abstract class Foo{
        public abstract void foo();
        }
```

```
abstract class Foo{
        }
```

rgupta.mtech@gmail.com

Abstract class use cases…

- Want to have some default functionality from base class and class have some abstract functionality that can't be define at that moment.

```
abstract class Animal
{
        public abstract void sound();
        public void eat()
         {
                System.out.println("animal eat...");
         }
}
```

- Don't want to allow a programmer to create object of an class as it is too generic

Interface vs abstract class

More example…

```
public abstract class Account  {
   public void deposit (double amount) {
        System.out.println("depositing " + amount);
   }

   public void withdraw (double amount) {
        System.out.println ("withdrawing " + amount);
   }

   public abstract double calculateInterest(double  amount);
}
```

```
public class SavingsAccount extends Account {

    public double calculateInterest (double amount) {
        // calculate interest for SavingsAccount
        return amount * 0.03;
    }

    public void deposit (double amount) {
        super.deposit (amount); // get code reuse
        // do something else
    }

    public void withdraw (double amount) {
        super.withdraw (amount); // get code reuse
        // do something else
    }
}
```

```
public class TermDepositAccount extends Account {

    public double calculateInterest (double amount) {
        // calculate interest for SavingsAccount
        return amount * 0.05;
    }

    public void deposit(double amount) {
        super.deposit (amount); // get code reuse
        // do something else
    }

    public void withdraw(double amount) {
        super.withdraw (amount); // get code reuse
        // do something else
    }
}
```

rgupta.mtech@gmail.com

Final

- What is the meaning of final
  - Something that can not be change!!!
- final
  - Final method arguments
    - Cant be change inside the method
  - Final variable
    - Become constant, once assigned then cant be changed
  - Final method
    - Cant overridden
  - Final class
    - Can not inherited (Killing extendibility)
    - Can be reuse

Some examples….

rgupta.mtech@gmail.com

Final class

- Final class can't be subclass i.e. Can't be extended
- No method of this class can be overridden
- Ex: String class in Java…

Real question is in what situation somebody should declare a class final

```
package cart;

public final class Beverage{

     public void importantMethod(){
          Sysout("hi");
     }
}

-----------------------------------------------
package examStuff;
import cart.*;

class Tea extends beverage{

}
```

rgupta.mtech@gmail.com

Final Method

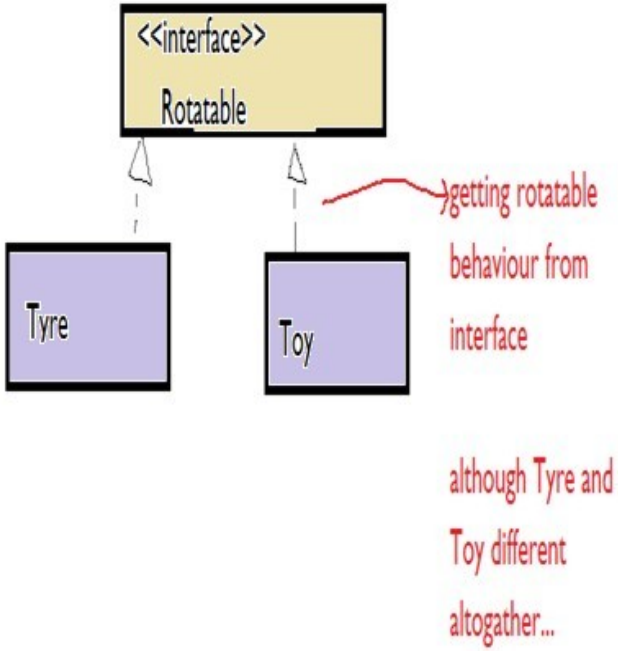
- Final Method Can’t overridden
- Class containing final method can be extended
- Killing substitutability

```
class Foo{

        public final void foo(){
                Sysout("i am the best");
                Sysout("You can use me but can't override me");
        }
};

class Bar extends Foo{
        @Override
        public final void foo(){


        }
}
```

rgupta.mtech@gmail.com

Interface?

- Interface : Contract bw two parties
- Interface method
  - Only declaration
  - No method definition
- Interface variable
  - Public static and final constant
  - Its how java support global constant Break the hierarchy
- Solve diamond problem
- Callback in Java* Some Example ….



rgupta.mtech@gmail.com

Interface?

- Rules
  - All interface methods are always public and abstract, whether we say it or not.
  - Variable declared inside interface are always public static and final Interface method can’t be static or final
  - Interface cant have constructor
  - An interface can extends other interface Can be used polymorphically
  - An class implementing an interface must implement all of its method otherwise it need to declare itself as an abstract class…

Implementing an interface...

What we declare

```
interface Bouncable{
     int i=9;
    void bounce();
    void setBounceFactor();
}
```

What compiler think...

```
interface Bouncable{
    public static final int i=9;
    public abstract void bounce();
    public abstract void setBounceFactor();
}
```

All interface method must be implemented....

```
class Tyre implements Bouncable{

    public void bounce(){
        Sysout(i);
        Sysout(i++);
    }
    public void setBounceFactor(){}
}
```

rgupta.mtech@gmail.com

Note

Following interface constant declaration are identical

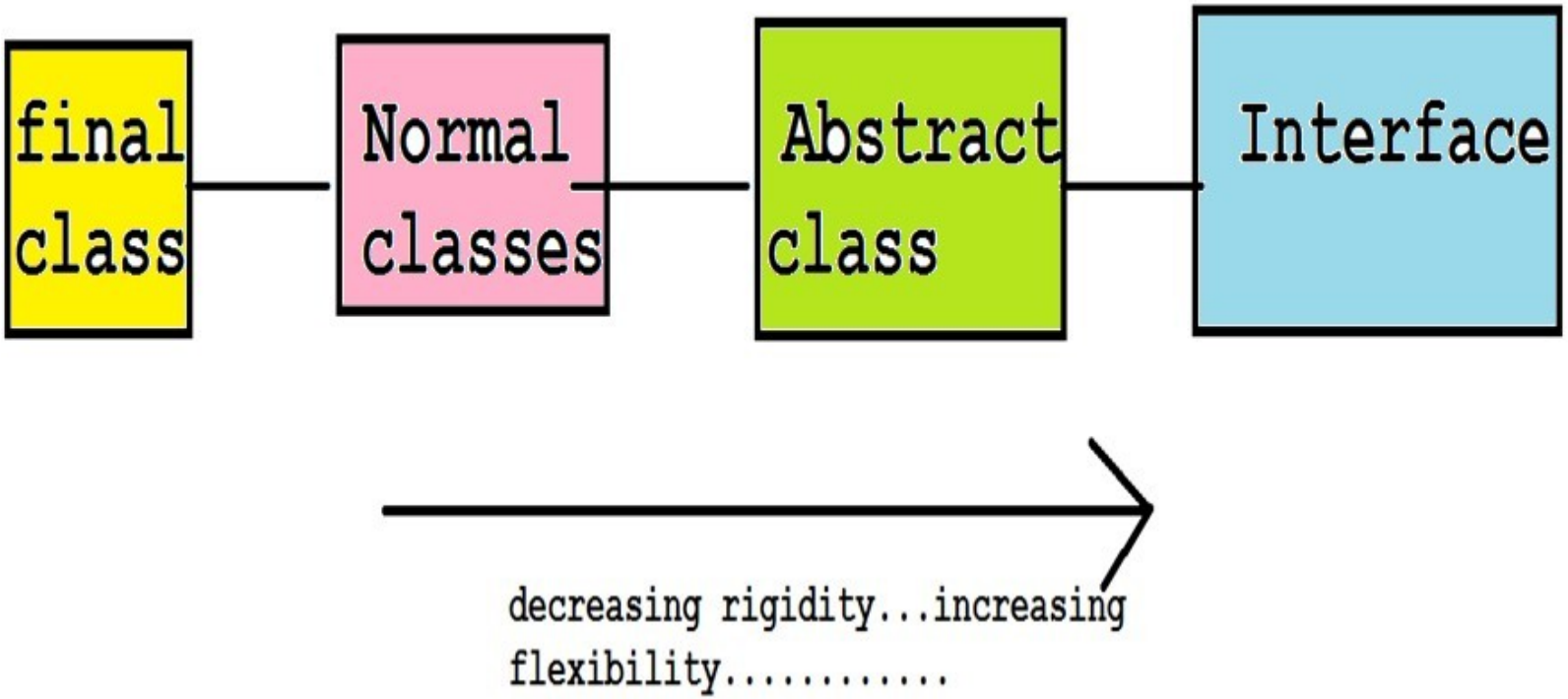
- int i=90;
- public static int i=90;
- public int i=90;
- public static int i=90;
- public static final int i=90;

Following interface method declaration don’t compile

- final void bounce();
- static void bounce();
- private void bounce();
- protected void bounce();

rgupta.mtech@gmail.com

Decreasing Rigidity..increasing Flexibility



rgupta.mtech@gmail.com

Type of relationship bw objects

- USE-A
- HAS-A
- IS-A (Most costly ?)

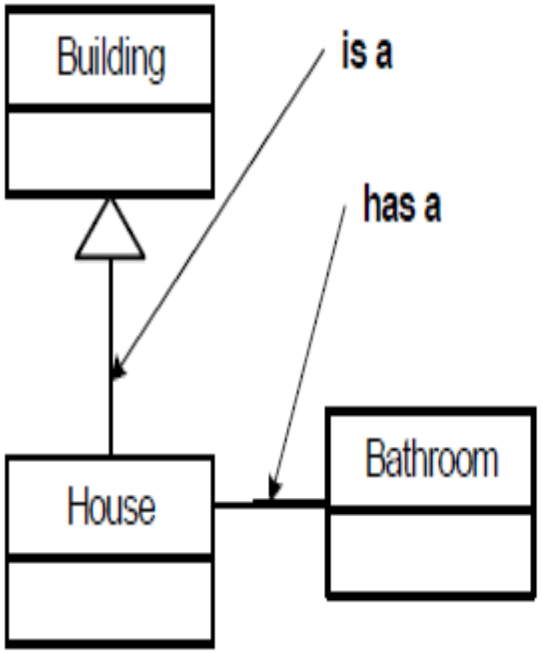
ALWAYS GO FOR LOOSE COUPLING AND HIGH COHESION…

But HOW?

rgupta.mtech@gmail.com

IS-A VS HAS-A

Inheritance [is a] Vs Composition [has a]



is a [House is a Building]

```
class Building{
 .......
}

class House extends Building{
 .........
}
```

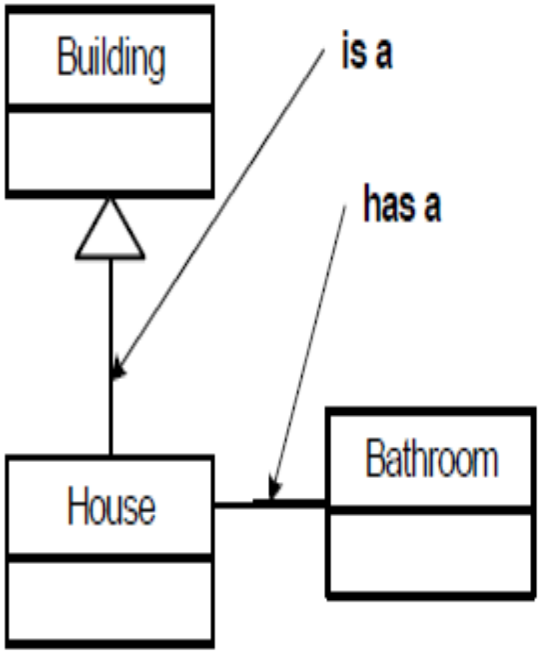
has a [House has a Bathroom]

```
class House {
    Bathroom room = new Bathroom() ;
    ....
   public void getTotMirrors(){
       room.getNoMirrors();
       ....
   }
}
```

rgupta.mtech@gmail.com

IS-A VS HAS-A

Inheritance [is a] Vs Composition [has a]



is a [House is a Building]

```
class Building{
 .......
}

class House extends Building{
 .........
}
```

has a [House has a Bathroom]

```
class House {
    Bathroom room = new Bathroom() ;
    ....
   public void getTotMirrors(){
       room.getNoMirrors();
       ....
   }
}
```

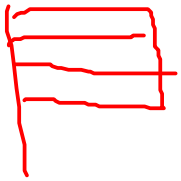
rgupta.mtech@gmail.com

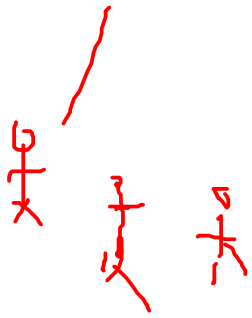
marker interface Types of inner classes,
Static and non-static inner classes,
local and anonymous inner classes

What are marker interface?

ie a interface that is empty

```
interface Foo{
}
```





Annotation : flag, meta data

```
@Entity
class Emp{

}
```

---

final keyword:
- **final data**
- **final class**
- **final method**
- **final method arg**

final class ⟷ **interface**

```
final class VeryImpLogic{
        //
}
```

all wrapper class , String, Class...

Nested classes

static nested classes

No static (inner classes)

- **Top level inner class**
- **annonymous inner class**
- **method local inner classes**

```
class LinkedList{
    class Node{
    }

}
```

there is mother and child relationship bw outer and inner class

without object of outer class object of inner can not be created

```
class Outer {
        class Inner {
                public void doInner() {
                        System.out.println("doInner code...");
                }
        }
        public static void getInner() {
                Inner inner=new Inner();
                inner.doInner();
        }
}
```

```
//LinkedList
class LinkList {

    private class Node {
        int data;
        Node next;

        public Node(int data) {
            this.data = data;
            this.next = null;
        }
    }

    Node head, tail;

    public LinkList() {
        head = tail = null;
    }

    public void addNode(int data) {
        Node newNode = new Node(data);
        if (head == null) {
            head = newNode;
            tail = newNode;
        } else {
            tail.next = newNode;
            tail = newNode;
        }
    }

    public void printLL() {
        Node currNode = head;
        if (currNode == null) {
            System.out.println("no element to print...");
        } else {
            while (currNode != null) {
                System.out.print(currNode.data + " --> ");
                currNode = currNode.next;
            }
            System.out.println();
        }
    }
}

public class DemoLL {

    public static void main(String[] args) {

        // Collection : aka readymade ds in java!, Iterator (top level inner class)

        LinkList linkList=new LinkList();
        linkList.addNode(33);
        linkList.addNode(3);
        linkList.addNode(303);
        linkList.addNode(993);
        linkList.addNode(83);

        linkList.addNode(30);


        linkList.printLL();
    }

}
```

Example
of Inner class

inner class