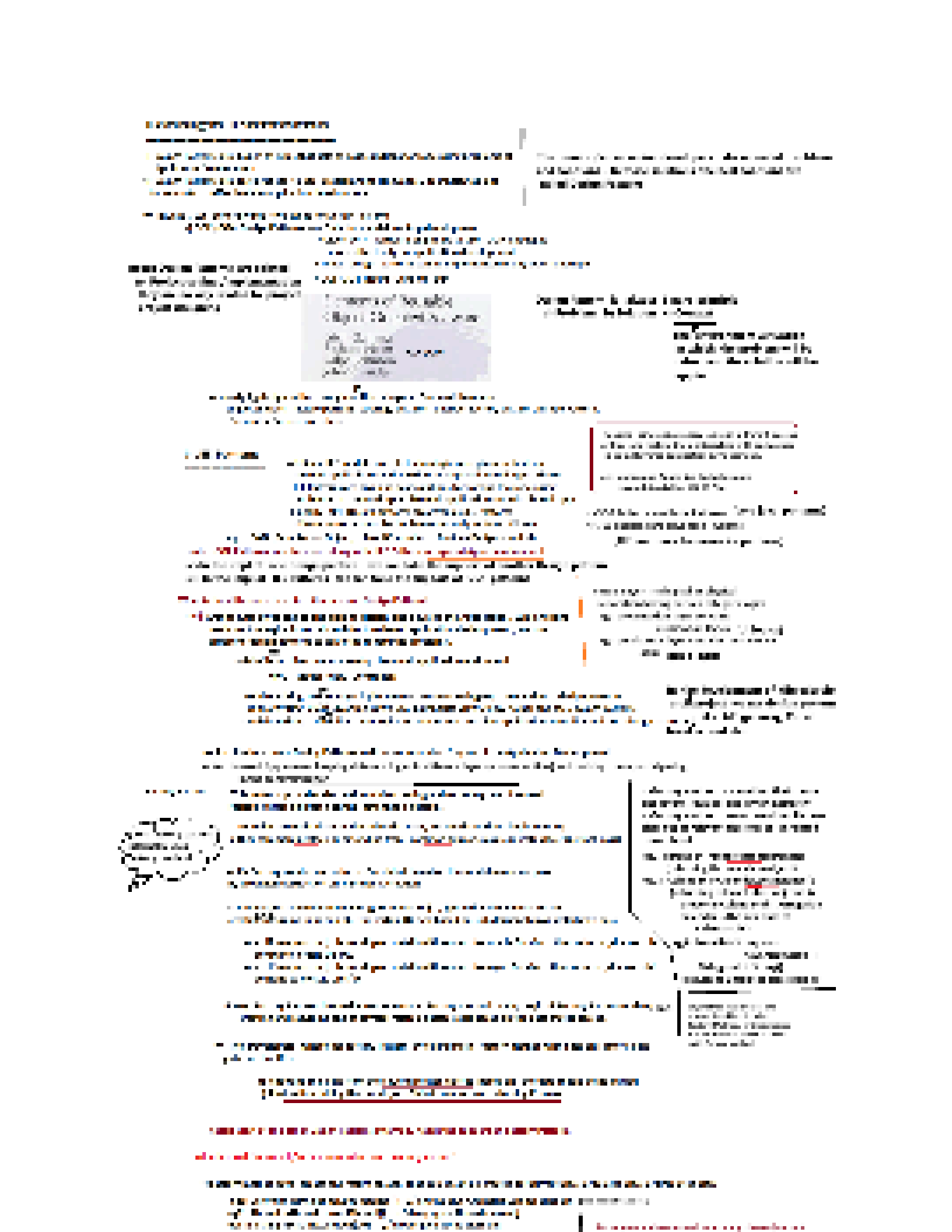

Design Patterns

=> Design Patterns are bunch of rules that acts as best solutions for reoccuring problems of Applicaton Development

=> Design Patterns are best pratices to use programming languages, technologies and frameworks effectively in application development.

=> In Java, we generally see two types of Design Patterns a) GOF /GO4 Design Patterns

note:: Design Patterns are related to Project coding /implementation they are no way related to project project designing

---> Only for standalone App development

The senior/super senior developers documented problems and solutions .. In those solutions the best solutions are called Design Patterns

---> Given by 4 software engineers to use OOP languages more effectively in application development

--> These design patterns can be implemented in any OOP language --> GOF/GO4 Means Gang Of Four

Elements of Reusable Object-Oriented Software

Erich Gamma Richard Helm

Ralph Johnson

John Vlissides

in

GOF/GO4

=> nearly 23 design pattern are given this category.. Some of them are

Design Pattern is always 3 part principle

a) Problem b) Solution c) Context

singleton class, factory pattern, strategy patterm, abstract factory pattern, Birdge Pattern, Decorator Pattern and etc...

b) JEE Pattterns

==========

==> Since JEE module and its technologies are given to devleop the web applicaitons and distributed apps as layered apps.. these JEE Patterns are given as best solutions for solving the reoccuring Problems of Layered Apps like web applications or Distributed Apps => nearly 20+ Design patterns are given as JEE Patterns

ग्र

The environment/situation in which the problem will be raised and the solution will be applied

Keeping different category logics in different classes or files and making them interacting with each other is called Layered application development..

eg:: standalone App is like Bachelor room Layered App is like 4BHK Flat

=>GOF Patterns are Basic Patterns (core java patterns) => JEE patterns are advanced Patterns

=> These patterns must be implemented only in Java,JEE env... eg:: DAO (Data Access Object), FrontController, Busines Deligate and etc... note:: GOF Patterns can be used along with JEE Pattern in Layered Apps development =>In the impl of one design pattern, we can take the support of another Design

pattern => In the impl of JEE Patterns, we can take the support of GOF patterns

What is the difference b/w Architecture and Design Pattern?

Ans) Architecutre gives end to end plan to design and develop the Application.. While writing each secton logics in certain architecture based application devleopment, we use different design patterns to solve the reoccuring problems,,

Factory Pattern

an

=>MVC is architecture to develop Java web application end to end

MVC (Model -View -Controller)

(JEE and Java frameworks patterns)

note:: Layer is a logical or physical separation having certain category logics eg:: presentation layer contains presetation logics (Ul logics) eg:: persistence layer contains persistence logics (jdbc logics)

=>While writing Model Layer logics we use Business Delegate, DAO and etc.. design patterns =>While writing ViewLayer logics we use Composite view, view helper and etc.. design patterns =>While writing Controller Layer logics we use Front Controller, ApplicationController and etc.. design patterns

In the development of MicroService arch project we use design patterns like API gateway, Circuit breaker and etc..

=> Architecture uses Desing Patterns in different modules /Layers of the Application Devleopment.. note:: Layered App means keeping different logics in different layers (classes or files) and making them participating in the communication

=========

note:: factory pattern internally uses

(factory method

: This pattern provides abstraction while creating and returning one of several realted classes obj based on the data that is passed...

=> To make serveral calsses as the related classes, we need see them implementing either common inteface or extending from Common supper class (concrete class/abstract class)

=> CarFactory creates and returns Car object based on the model name we pass by providign abstraction on Car object creation

=> Connection con=DriverManager.getConnection(-,-,-) provides the abstraction on creating returning jdbc con obj for certain Db s/w based on url, dbuser,db pwd details we pass.

=>Factory method is a method that create and return Java obj directly or indirectly =>Factory method can return either its own class obj or relevant class obj or unrelated class object

eg1: Thread t=Thread.currentThread(); (returning its own class object) eg2:: Calendar cl-Calendar.getInstance(); (returning relevant class obj that is GregorianCalendar obj. Gregorian Calendar is the sub class of Calendar(AC)

note:: if we pass url, dbuser,dbpwd details with respect to oracle Db s/w .. then returns jdbc con obj poiting to oracle Db s/w

note:: if we pass url, dbuser,dbpwd details with respect to mysql Db s/w .. then returns jdbc con obj pointing to mySQL DB s/w

=>Every Factory Pattern internally uses one static Factory method having logic of Factory Pattern(nothing but creating and returning one of several related classses obj) based on the data that is passed.

=> The Developers whose use factory Pattern by calling static factory method with data on Factory class gets two benifits..

a) choosing the class from the several related classes based on the data that is being passed b) Abstraction (hiding the details) on Object creation and returning Process

eg3: ArrayList<String> al=

new ArrayList<>(); String s=al.toString(); (returning unrelated class object)

any method that creates and returns the object is called Factory Method.. if that method is static method then it is called static factory method

note:: Most of the other Design Patterns internally needs the support of Factory Pattern.

What is the difference b/w factory method and Factory Pattern?

=>Factory method says i create and return object.. that object can be created for current class or related class or unrelated class eg1: Calendar cal=Calendar.getInstance(); // always give GregorianCalenar class obj (sub class of Calendar) eg2:: Thread t=Thread.currentThread(); //always give Thread class obj eg3 :: String s1=String.valueOf(20); //always give String class obj

=>Factory Pattern internally uses the factory method and creates, returns one of several related classes obj based on the data that is being passed.

eg1:

Factory method creates and returns any class obj(current class or related class or unrelated class)

Factory pattern internally uses factory method but returns one of the several related classes obj based on the data that is passed

Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","manager"); Gives jdbc con object poiting to oracle Db s/w

Connection con=DriverManager.getConnection("jdbc:mysql:///ntaj616db",

root"root" ');

Gives jdbc con object pointing to mysql Db s/w

of

The classes both these jdbc con objs are related classes becoz they implement the common interface java.sql.Connection(I). is

eg2:: Spring's IOC container given based on factory Pattern

WishMessageGenerator wmg=ctx.getBean("wmg");

Factory method of Factory Pattern (IOC container)

note:: factory method of facctory pattern mostly be static method .. (but not mandatory) i.e it can be taken as static or non-static method.

Most of statements in JDBC are Factory Pattern statement becoz they create and return objects by providing abstraction on object creation process

eg1:: Statement st= con.createStatement() eg2:: ResultSet rs=st.executeQuery(--);

having

Example App on Factory Pattern --Core Java

CoreProj2 |--------->src

(Core Java App)

|---->com.nt.comps

=>Factory Pattern is a class that is static /non-static factory method having logics to create and return one of the several related classes obj based on the data that is being passed

In jdbc programming most method calls that are giving JDBC objs are factory Patterns

Connection con= DriverManager.getConnection(url,dbuser,dbpwd); Factory Pattern with static factory method

|---->Bike.java (common interface)

|---->SportsBike.java

|----->EletricBike.java

Implementation classes of

|----->BulletBike.java

Bike(1) .. So we can say these

Statement st=con.createStatement();

|-----> StandardBike.java are related classes

|---->com.nt.factory

|----->BikeFactory.java (Factory Pattern)

|--->com.nt.test

|--------> FactoryPatternTest.java (Client App)

CoreJavaProj03-FactoryPattern

> JRE System Library [JavaSE-17]

#src

com.nt.comps

> Bike.java

> BulletBike.java

> ElectricBike.java

>SportsBike.java

> StandardBike.java

com.nt.factory

> BikeFactory.java

com.nt.test

> FactoryPattern Test.java

//Bike.java (Common Interface)

```
package com.nt.comps;
public interface Bike {
}
public void drive();
```

//ElectricBike.java

```
package com.nt.comps;
public class ElectricBike implements Bike {
```

Factory Patterin with non-static factory method ResultSet rs=st.executeQuery("SELECT * FROM EMP");
Factory Pattern with non-static factory method

=>Any reusable java class is called Component =>Spring beans are also components

=>we can make classes as the related classes by making them implementing common interface and extending from common super class

```
@Override
}
}
public void drive() {
System.out.println("Electrice Bike.drive():: driving ElectricBike (Ather bike)");
//BulletBike.java
package com.nt.comps;
public class BulletBike implements Bike {
@Override
```

public void drive() {

=>To make several classes as related classes, we need to make those classes implementing common interface or extending from common super class

=> The classes of different jdbc con objs are related classes becoz they are implementing common interface called java.sql.Connection(I) =>The classes of different jdbc statement objs are related classes becoz they are implementing common interface called java.sql.Statement(1)

```
System.out.println("BulletBike.drive():: driving BulletBike (Royal Enfield- Bullet)");
}
//SportsBike.java
package com.nt.comps;
public class Sports Bike implements Bike {
@Override
```

public void drive() {

```
}
}
System.out.println("SportsBike.drive():: driving SporttsBike (kawasaki- Ninja)");
```

//StandardBike.java

package com.nt.comps;

public class Standard Bike implements Bike {

@Override

public void drive() {

System.out.println("StandardBike.drive():: driving Standard bike (bajaj discover)");

}

}

//BikeFactory.java (Factory Pattern)

package com.nt.factory;

import com.nt.comps.Bike; import com.nt.comps. BulletBike; import com.nt.comps. Electric Bike;

import com.nt.comps.SportsBike; import com.nt.comps.StandardBike;

public class BikeFactory {

//static factory method having factory pattern logic public static Bike orderBike(String type) { Bike bike=null;

if(type.equalsIgnoreCase("standard"))

bike=new StandardBike();

else if(type.equalsIgnoreCase("sports")) bike=new SportsBike();

else if(type.equalsIgnoreCase("electric"))

bike=new ElectricBike(); else if(type.equalsIgnoreCase("bullet")) bike=new BulletBike();

else

=>This method is taken as public becoz it shoud be invoked outside of the factory class in different pkgs

=> this method is taken as static becoz it should be invoked on class name (with out object) from the Client Apps becoz Client Apps hot intrested in Factory class obj.. they are intrested Factory Supplied Resultant Bike object

=> The retuX Bike(1) becoz that is common for all the related Bike classes.. So method can create and return any Bike(1) impl class obj based on the data that is being provided.

throw new IllegalArgumentException("Invalid Bike Type");

return bike;

}

}

(Coient App)

//ractiyrallerest.java package com.nt.test;

import com.nt.comps.Bike; import com.nt.factory.BikeFactory;

public class FactoryPatternTest {

public static void main(String[] args) {

3 important statements on method return types

=> if the java method return type is an interface then method returns one

of its implementation class obj as the return value

=> if the java method return type is an abstract class then method returns one of its sub class obj as the return value

=> if the java method return type is an concrete class then method can return

either that concreate class obj or one its sub class obj as the return value

Bike bike-BikeFactory.orderBike("standard"); bike.drive();

System.out.println(":

}

}

");

Bike bike1=BikeFactory.orderBike("sports"); bike1.drive();

System.out.println(": =======

===");

Bike bike2=BikeFactory.orderBike("electric");

bike2.drive();

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

Bike bike3=BikeFactory.orderBike("bullet"); bike3.drive();

Strategy DesignPattern

=> This is GOF Pattern .. can be implemented in any OOP language

=> This Pattern is very popular in spring programming becoz it gives set of principles towards designing classes

of Dependency Management and we use dependency Management a lot in spring programming

DP

=> Strategy says ....

1

set¶ules/principles

While designing the classes of dependecny Managenement follow which allows to design the classes as loosely coupled interchangable parts.

(we can change one dependent with another dependent with out touching the source code of target class)

The target and dependent classes of Dependency Management can be designed by using the following principles of strategy DP

=====

Rule/Principle1 :: Favour Compsition (HAS-A Relation) over Inheritence (IS -A Relation) Rule/Principle2 :: Always code to interfaces/abstract classes i.e do not code to concreate classes

to achieve the loose coupling in the implementation Dynamic Polymorphism

Rule Principle3 :: Our Code must be open for extension and must be closed for Modification..

Rule/Principle1 :: Favour Compsition (HAS-A Relation) over Inheritence (IS -A Relation)

Inheritence (Is-A relationship)

class A{

class B extends A{

"B is the sub class of A" (IS-A relationship)

"B is inherited from A" (IS-a Relationship)

=>if any object name is pronounced with interface name then it is the object of class that implements an interface. eg: Serializable obj =>if any object name is pronounced with abstract class name then it is the object of a class that extends from the abstract class eg: Calendar obj

=>if any object name is pronounced with class name then that can be object of java class or the obj of one its

sub class eg: Stack obj

Composition (HAS-A relation)

```
class A{
class B { HAS-A property private A a=new A();
}
}
```

=>HEre the object of B class has

the object of A class (HAS-A relationship)

B class obj

`A a`

A class obj

ЭТО

The object of "B" class

Where should i use Inheritence and

where should i use Composition?

Ans) if class1 wants to use entire properties and methods

of class2 then prefer using Inheritence

eg: class extending from the Thread

class extending from the HttpServlet

has the object of "A" class.

Limitations of Inheritence

if class1 wants to use only few properties and few methods of class2 then prefer using Composition

a) Some Programming Languages including Java.. does not support Multiple Inheritence using classes

b) With Inheritence the code becomes fragile (Easily breakable)

of

c) The Testing code becomes complex .. if classes are there in the Inheritence

eg: spring first App

a) Some Programming Languages including Java .. does not support Multiple Inheritence using classes

problem:

class A{

class B{

class C extends A,B{

Not Supported in Java

..

}

}

...

=>Java does not support multiple inheritence_through_classes..

note: Do not look at interfaces in the angel of inheritence.. becoz

static

=> interface methods are public methods i.e they are visible every where

=> interface properties are public static final properties .. i.e they are access able

in all the places..irrespective wheather we implement interface or not f

=> static methods of interface (java 8) can be access every where with interace name..

note:: Always look at interfaces in the angel of polymorphism to achive looselycoupling..

interface 11{

}

public void m1();

public class A1 implements I1{

public void m1(){

public class B1 implements 11{ public void m1(){

class A implements X,Y{

"

గు

...

//logic1

version1/

Version2/form2

}

form1

//logic2

Looks like multiple inheritance... but taken

}

}}

for polymorphism

...

}.

Polymorphism means 1 method --- multiple forms (implementations) (m1() -> 11 interface)

Solution using composition

class A{

class B{

class C{

...

}

}

}

|---> m1() definitation in class A1

|---> m1() definitation in class B1

private A a1=new A(); private B b1=new B();

--> Now C can use both

cass A and class B properties and methods.

(the effect of multiple inheritence

has come here)

b) With Inheritence the code becomes fragile (Easily breakable)

problem:

class A{

float

public int m1(){

return 100;

return 1000;

class B extends A{

public int m1(){

class C extends B{

public int m1(){ return 1000;

}

}

}

}

class E extends B{

public int m1(){

return

10;

class D extends B{

}

public int m1(){

}

C

extends

extends

E

return 100;

}

}

(return type)

if u modify the the signature of m1() method of class A (Top class of the inheritence hierarchy) (return type is changed from

int to float) then all the clsses of that inheritence will be collapsed.. (becomes incompitable) .. This indicates

inheritence the code is very much fragile.

Solution using Composition

class A{

float

public int m1(){

return 100;

}

}

class B{ HAS-A property(Composition) private A a1=new A();

public int m1(){

int x1-a1.m1();

int x1=Math.round(a1.m1());

class C extends B{

}

(inheritance)

class D extends B{

with

}

}

return x+100;

(inheritance)

}

Here if we modify the return type of m1() method in class A .. that will effect only one line in class B

where that m1() method is called.. By just modifying that line code.. we can restrict the problem to that level itself..

from

i.e the other classes inheritering Class B will not be effected.

old code

int x=a1.m1();

new code

int x=Math.round(a1.m1());

When should i go for composition and when should i go for Inheritence?

Ans) if class1 wants use all the properties and methods of class2 then keep them in Inheritence relationship

class c2{

...

}

class c1 extends c2{

"

while making our classees as

Threads, Servlet comps.. we fully

dependent on Inheritence

if class1 wants use only few properties and methods of class2 then keep them in composition relationship

class c2{

class c1 {

private C2 c2=new C2();

Use composition while designing classes

for Dependency Management (Spring Programming

}

c) The Testing code becomes complex .. if classes are there in the Inheritence

uses this a lot)

both

are

=>Testing is all about matching expected results with actual results .. if results matched then we can say test results are positive

if not matched test result are -ve

=> lots of testings will be taken care by testers.. But Unit testing must be taken care by the developers..

=> Progammer's testing on his own piece of code is called Unit Testing...

=> To test the code in all permitations and combintations we write the test cases/Test plans

we generally write these test plans on 1 per each varitey of input

Problem

class A{

public int m1(){

return 100;

}

```
class B extends A{ (inheritance is there)
public int m3(){
return 200;
}
public int m2(){
public int m4(){
return 1000;
return 2000;
}
}
}
}
```

While testing class "B" we need

to write Testcases for "B" class

methods (m3(),m4())

and A class

methods (m1(),m2()) becoz

using the object of class B we can

invoke total 4 methods

m1(),m2(),m3(),m4()

note:: In Inheritence, we need to perform unit testing on both current class methods and the inherited methods belonging to all the classes of inheritence hierarchy (Very Very complex)

solution using composition

```
class A{
class B{
public int m1(){
private A a=new A(); //composition
public int m3(){
int x=a.m1();
return 100;
return x+100;
}
}
public int m2(){
return 1000;
}
}
```

}

}

public int m4(){

return 2000;

=>While doing unit Testing on

class "B" methods we need to

write test cases only m3(),m4()

methods i.e no need of writing

testcases for class A methods (m1(),m2())..

=> m1() method class A is indirectly

unit

tested while performing unit

testing on class B m3() method.