DESIGN PATTERNS

```java
package oops;

public class StrategyPattern {
	public static void main(String[] args) {
		BadBrush shaitan=new BadBrush();
		shaitan.doPaint(1);

		GoodBrush angel=new GoodBrush();
		angel.doPaint(new BluePaint());
	}
}

class BadBrush{
	public void doPaint(int n) {
		if(n==1) {
			System.out.println("red colour...");
		}
		else if(n==2) {
			System.out.println("blue colour...");
		}
		else if(n==3) {
			System.out.println("green colour....");
		}
	}
}

//To eliminate the if-else-if ladder - implement strategy pattern

/*
	3 golden rules
	1. Convert the condition to classes.
	2. Group the classes under a common hierarchy
	3. create a association between the using class and the
hierarchial class
*/
abstract class Paint{

}
class RedPaint extends Paint{

}
class BluePaint extends Paint{

}
```

```java
class GoodBrush{

	public void doPaint(Paint paint) {
		System.out.println(paint);

	}
}
```

```java
package oops;
/*
 * 1. Convert the condition to classes.
 * 2. Create a hierarchial classification of condition using a common
class .
 * 3. Create a association between the  using class and hierarchail
common class.
 */
public class StrategyPattern2 {
	public static void main(String[] args) {
		GoodDog tiger=new GoodDog();

		tiger.play(new Stick());
	}
}
class Dog{
	public void play(String item) {
		if(item.equals("stick")) {
			System.out.println("dog bites......");
		}
		else if(item.equals("stone")) {
			System.out.println("dog barks.....");
		}
		else if(item.equals("biscuit")) {
			System.out.println("dog wags tail....");
		}
	}
}
abstract class Item{
	public abstract void action() ;
}
class Stick extends Item{
	public void action() {
		System.out.println("dog bites....");
	}
}
class Stone extends Item{
	public void action() {
		System.out.println("dog
barks................................");
	}
}
//Closed for modification but open for extension
class GoodDog{
	public void play(Item item) {
		item.action();
	}
}
```

```
package day5;
import java.util.Scanner;

public class InherDemo8 {
	public static void main(String[] args) {
		//BadFan shaitan=new BadFan();
		GoodFan khaitan=new GoodFan();
		Scanner scan=new Scanner(System.in);
		while(true) {
			System.out.println("Please enter for pulling...");
			scan.next();
			khaitan.pull();
		}
	}
}
class BadFan{
	int state=0;
	public void pull() {
		if(state==0) {
			System.out.println("switch on state...");
			state=1;
		}
		else if(state==1) {
			System.out.println("medium speed state...");
			state=2;
		}
		else if(state==2) {
			System.out.println("high speed state...");
			state=3;
		}
		else if(state==3) {
			System.out.println("switch off state...");
			state=0;
		}
	}
```

```java
}
abstract class State{
	public abstract void pull(GoodFan fan);
}
class GoodFan{
	State state=new SwitchOffState();
	public void pull() {
		state.pull(this);
	}
}
class SwitchOffState extends State{
	@Override
	public void pull(GoodFan fan) {
		System.out.println("switch on state.....");
		fan.state=new SwitchOnState();
	}
}
class SwitchOnState extends State{
	@Override
	public void pull(GoodFan fan) {
		System.out.println("medium speed state.....");
		fan.state=new MediumState();
	}
}
class MediumState extends State{
	@Override
	public void pull(GoodFan fan) {
		System.out.println("high speed state.....");
		fan.state=new HighSpeedState();
	}
}
class HighSpeedState extends State{
	@Override
	public void pull(GoodFan fan) {
		System.out.println("switch off state.....");
		fan.state=new SwitchOffState();
	}
}
```

```java
package day4;

public class InherDemo7 {
	public static void main(String[] args) {
		Tv tv=new Tv();
		XBox xbox=new XBox();
		VGame vgame=new VGame();
		SetTopBox box=new SetTopBox();
		Ginie ginie=new Ginie();
		
		NewsChannelCommand ncc=new NewsChannelCommand(tv, xbox, 
vgame, box);
		ginie.setCommand(ncc, 1);
		ginie.executeCommnad(1);
	}
}

class Ginie{
	Command c[]=new Command[5];
	
	public Ginie() {
		for(int i=0;i<5;i++) {
			c[i]=new DummyCommand();
		}
	}
	public void executeCommnad(int slot) {
		c[slot].execute();
	}
	
	public void setCommand(Command command,int slot) {
		c[slot]=command;
	}
}
abstract class Command{
	Tv tv;
	XBox xbox;
	VGame vgame;
	SetTopBox box;
	public Command() {
		
	}
	public Command(Tv tv, XBox xbox, VGame vgame, SetTopBox box) {
		this.tv = tv;
		this.xbox = xbox;
		this.vgame = vgame;
		this.box = box;
	}
	public abstract void execute();
}
class DummyCommand extends Command{

	@Override
	public void execute() {
		System.out.println("I am dummy yet to be operational...");
	}
}
class NewsChannelCommand extends Command{

	public NewsChannelCommand(Tv tv, XBox xbox, VGame vgame,
SetTopBox box) {
		super(tv,xbox,vgame,box);
	}

	@Override
	public void execute() {
		System.out.println("news channel command started ....");
		tv.av1();
		box.newsChannel();
		System.out.println("news channel started..enjoy news...");
	}
}
class SerialChannelCommand extends Command{
	public SerialChannelCommand(Tv tv, XBox xbox, VGame vgame,
SetTopBox box) {
		super(tv,xbox,vgame,box);
	}

	@Override
	public void execute() {
		System.out.println("serial channel command started ....");
		tv.av1();
		box.serialChannel();
		System.out.println("serial channel started..enjoy saas bahu
serial...");
	}
}
class TTGameCommand extends Command{
	public TTGameCommand(Tv tv, XBox xbox, VGame vgame, SetTopBox
box) {
		super(tv,xbox,vgame,box);
	}

@Override
	public void execute() {
		System.out.println("tt game command started ....");
		tv.av2();
		xbox.TTGame();
		System.out.println("tt game started..enjoy playing...");
	}
```

```
}
class SkiiGameCommand extends Command{
	public SkiiGameCommand(Tv tv, XBox xbox, VGame vgame, SetTopBox 
box) {
		super(tv,xbox,vgame,box);
	}
	
	@Override
	public void execute() {
		System.out.println("skii game command started ....");
		tv.av2();
		vgame.skatingGame();
		System.out.println("skii game started..enjoy playing...");
	}
}
class Tv{
	public void av1() {
		System.out.println("av1 started...");
	}
	public void av2() {
		System.out.println("av2 started...");
	}
}
class XBox{
	public void TTGame() {
		System.out.println("start tt game...");
	}
}
class VGame{
	public void skatingGame() {
		System.out.println("start skiing game..");
	}
}
class SetTopBox{
	public void newsChannel() {
		System.out.println("news channel started...");
	}
	public void serialChannel() {
		System.out.println("serial channel started..");
	}
}
```

```
package day4;
public class InherDemo6 {
	public static void main(String[] args) {
		Food food=new Rice(new FishCurry(new Rice(new Kabab())));
		System.out.println("Rice cost..:"+food.cost());
	}
}
abstract class Food{
	public abstract int cost();
}
abstract class VegFood extends Food{

}
abstract class NonVegFood extends Food{

}
class Rice extends VegFood{
	public Rice() {
		// TODO Auto-generated constructor stub
	}
	Food food;
	public Rice(Food food) {
		this.food=food;
	}
	@Override
	public int cost() {
		// TODO Auto-generated method stub
		if(food!=null) {
			return 10+food.cost();
		}
		else {
			return 10;
		}
	}
}

class FishCurry extends NonVegFood{
	public FishCurry() {
		// TODO Auto-generated constructor stub
	}
	Food food;
	public FishCurry(Food food) {
		this.food=food;
	}
	@Override
	public int cost() {
		// TODO Auto-generated method stub
		if(food!=null) {
			return 20+food.cost();
			}
			else {
				return 20;
			}
		}
}
class Kabab extends NonVegFood{
		public Kabab() {
			// TODO Auto-generated constructor stub
		}
		Food food;
		public Kabab(Food food) {
			this.food=food;
		}
		@Override
		public int cost() {
			// TODO Auto-generated method stub
			if(food!=null) {
				return 50+food.cost();
			}
			else {
				return 50;
			}
		}
}
```

```
package day4;
public class InherDemo5 {
		public static void main(String[] args) {
			ShakthiSocket ss=new ShakthiSocket();
			HPPlug hp=new HPPlug();
			IndianAdapter ip=new IndianAdapter();
			ip.setAmericanPlug(hp);
			ss.roundPinHole(ip);
		}
}
class IndianAdapter extends IndianPlug{
		AmericanPlug ap;
		public void setAmericanPlug(AmericanPlug ap) {
			this.ap=ap;
		}
		@Override
		public void action() {
			ap.action();
		}
```

```java
}
abstract class IndianSocket{
	public abstract void roundPinHole(IndianPlug ip);
}
abstract class IndianPlug{
	public abstract void action();
}

abstract class AmericanPlug{
	public abstract void action();
}

class HPPlug extends AmericanPlug{
	@Override
	public void action() {
		System.out.println("american plug working...");
	}
}

class ShakthiSocket extends IndianSocket{
	@Override
	public void roundPinHole(IndianPlug ip) {
		ip.action();
	}
}
```

```java
package designpatterns;
//https://fluvid.com/videos/detail/ykZL6ck9jLiY9w2zo#.YhTNuPmWDig.link
import java.util.Scanner;

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

		Child baby=new Child();
		System.out.println("please input and enter to proceed....");
		Scanner scan=new Scanner(System.in);

		String n=scan.next();

		Dog tiger=new Dog();


		baby.playWithDog(tiger, n);
	}
}
```

```
class Dog{
	public void play(String item)throws DogExceptions {
		if(item.equals("stick")) {
			throw new DogBiteException();
		}
		else if(item.equals("stone")) {
			throw new DogBarkException();
		}
		else if(item.equals("bone")) {
			throw new DogHappyException();
		}
	}
}

abstract class DogExceptions extends Exception{
	public abstract void visit();
}
class DogBiteException extends DogExceptions{
	@Override
	public void visit() {
		new Handler911().handle(this);
	}
}
class DogBarkException extends DogExceptions{
	@Override
	public void visit() {
		new Handler911().handle(this);
	}
}
class DogHappyException extends DogExceptions{
	@Override
	public void visit() {
		new Handler911().handle(this);
	}
}
class Child{
	void playWithDog(Dog dog,String item) {
		try {
			dog.play(item);
		}catch(DogExceptions de){
			de.printStackTrace();
			de.visit();
		}
	}
}

class Handler911{
	public void handle(DogBiteException dbe) {
		System.out.println("dog has bitten, wait for the
ambulance,,,,,,,,,,");
	}
	public void handle(DogBarkException dbr) {
		System.out.println("no worries  just
ignore...................");
	}
	public void handle(DogHappyException dbr) {
		System.out.println("enjoy..................");
	}
}
```

```
package designpatterns;

public class BuilderPattern {
	public static void main(String[] args) {
		//Using builder to get the object in a single line of code
and
				//without any inconsistent state or arguments
management issues
		Computer comp = new Computer.ComputerBuilder("500 GB", "2
GB").setBluetoothEnabled(true)

	.setGraphicsCardEnabled(true).build();
	}
}class Computer {

	//required parameters
	private String HDD;
	private String RAM;

	//optional parameters
	private boolean isGraphicsCardEnabled;
	private boolean isBluetoothEnabled;
		public String getHDD() {
		return HDD;
	}	public String getRAM() {
		return RAM;
	}	public boolean isGraphicsCardEnabled() {
		return isGraphicsCardEnabled;
	}	public boolean isBluetoothEnabled() {
```

```
            return isBluetoothEnabled;
      }

      private Computer(ComputerBuilder builder) {
            this.HDD=builder.HDD;
            this.RAM=builder.RAM;
            this.isGraphicsCardEnabled=builder.isGraphicsCardEnabled;
            this.isBluetoothEnabled=builder.isBluetoothEnabled;
      }

      //Builder Class
      public static class ComputerBuilder{        // required
parameters
            private String HDD;
            private String RAM;         // optional parameters
            private boolean isGraphicsCardEnabled;
            private boolean isBluetoothEnabled;

            public ComputerBuilder(String hdd, String ram){
                  this.HDD=hdd;
                  this.RAM=ram;
            }           public ComputerBuilder
setGraphicsCardEnabled(boolean isGraphicsCardEnabled) {
                  this.isGraphicsCardEnabled = isGraphicsCardEnabled;
                  return this;
            }           public ComputerBuilder
setBluetoothEnabled(boolean isBluetoothEnabled) {
                  this.isBluetoothEnabled = isBluetoothEnabled;
                  return this;
            }

            public Computer build(){
                  return new Computer(this);
            }     }}
```