1. **BRIDGE PATTERN**

package bridgepatternn;

public class BridgePatternn {

public static void main(String[] args) {

ThreadScheduler Thread1 = new PreemptiveThreadSchedular(new Windows(), new Unix(), new JVM());

Thread1.Schedular();

ThreadScheduler Thread2 = new TimeSlicedThreadSchedular(new Windows(), new Unix(), new JVM());

Thread2.Schedular();

}

}

package bridgepatternn;

/\*\*

\*

\* @author Administrator

\*/

public interface Platforms {

abstract public void plat();

}

class Windows implements Platforms {

@Override

public void plat()

{

System.out.print("Windows");

}

}

class Unix implements Platforms {

@Override

public void plat()

{

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

}

}

class JVM implements Platforms {

@Override

public void plat()

{

System.out.print(" And");

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

}

}

package bridgepatternn;

public abstract class ThreadScheduler

{

protected Platforms platform1;

protected Platforms platform2;

protected Platforms platform3;

protected ThreadScheduler(Platforms platform1, Platforms platform2, Platforms platform3)

{

this.platform1 = platform1;

this.platform2 = platform2;

this.platform3 = platform3;

}

abstract public void Schedular();

}

class PreemptiveThreadSchedular extends ThreadScheduler {

public PreemptiveThreadSchedular(Platforms platform1, Platforms platform2, Platforms platform3)

{

super(platform1, platform2, platform3);

}

@Override

public void Schedular()

{

System.out.print("PreemptiveThreadSchedular ");

platform1.plat();

platform2.plat();

platform3.plat();

}

}

// Refine abstraction 2 in bridge pattern

class TimeSlicedThreadSchedular extends ThreadScheduler {

public TimeSlicedThreadSchedular(Platforms platform1, Platforms platform2, Platforms platform3)

{

super(platform1, platform2, platform3);

}

@Override

public void Schedular()

{

System.out.print("TimeSlicedThreadSchedular ");

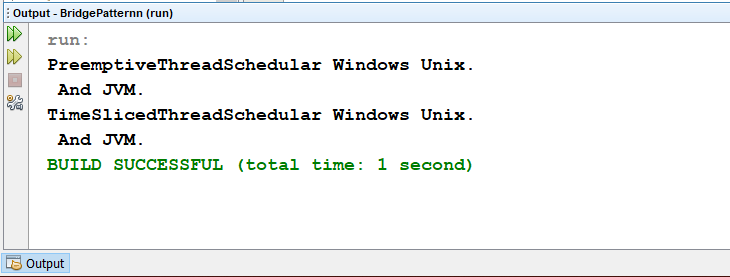
platform1.plat();

platform2.plat();

platform3.plat();

}

}



1. **CHAIN OF RESPONSIBILITY PATTERN**

package chain.design.pattern;

/\*\*

\*

\* @author hp

\*/

public abstract class AbstractLogger {

public static int INFO = 1;

public static int DEBUG = 2;

public static int ERROR = 3;

protected int level;

//next element in chain or responsibility

protected AbstractLogger nextLogger;

public void setNextLogger(AbstractLogger nextLogger){

this.nextLogger = nextLogger;

}

public void logMessage(int level, String message){

if(this.level <= level){

write(message);

}

if(nextLogger !=null){

nextLogger.logMessage(level, message);

}

}

abstract protected void write(String message);

}

package chain.design.pattern;

/\*\*

\*

\* @author hp

\*/

public class ChainDesignPattern {

private static AbstractLogger getChainOfLoggers(){

AbstractLogger errorLogger = new ErrorLogger(AbstractLogger.ERROR);

AbstractLogger fileLogger = new FileLogger(AbstractLogger.DEBUG);

AbstractLogger consoleLogger = new ConsoleLogger(AbstractLogger.INFO);

errorLogger.setNextLogger(fileLogger);

fileLogger.setNextLogger(consoleLogger);

return errorLogger;

}

public static void main(String[] args) {

AbstractLogger loggerChain = getChainOfLoggers();

loggerChain.logMessage(AbstractLogger.INFO,

"This is an information.");

loggerChain.logMessage(AbstractLogger.DEBUG,

"This is an debug level information.");

loggerChain.logMessage(AbstractLogger.ERROR,

"This is an error information.");

}

}

package chain.design.pattern;

/\*\*

\*

\* @author hp

\*/

public class ConsoleLogger extends AbstractLogger {

public ConsoleLogger(int level){

this.level = level;

}

@Override

protected void write(String message) {

System.out.println("Standard Console::Logger: " + message);

}

}

package chain.design.pattern;

/\*\*

\*

\* @author hp

\*/

public class ErrorLogger extends AbstractLogger {

public ErrorLogger(int level){

this.level = level;

}

@Override

protected void write(String message) {

System.out.println("Error Console::Logger: " + message);

}

}

package chain.design.pattern;

/\*\*

\*

\* @author hp

\*/

public class FileLogger extends AbstractLogger {

public FileLogger(int level){

this.level = level;

}

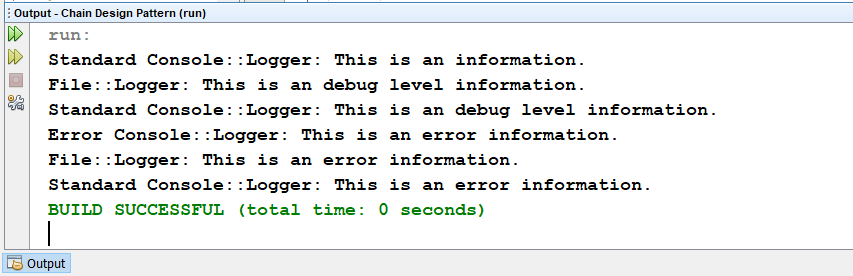
@Override

protected void write(String message) {

System.out.println("File::Logger: " + message);

}

}



1. **COMMAND PATTERN**

package command.design.pattern;

import java.util.ArrayList;

import java.util.List;

public class Broker {

private List<Order> orderList = new ArrayList<Order>();

public void takeOrder(Order order){

orderList.add(order);

}

public void placeOrders(){

for (Order order : orderList) {

order.execute();

}

orderList.clear();

}

}

package command.design.pattern;

/\*\*

\*

\* @author hp

\*/

public class BuyStock implements Order {

private Stock abcStock;

public BuyStock(Stock abcStock){

this.abcStock = abcStock;

}

public void execute() {

abcStock.buy();

}

}

package command.design.pattern;

/\*\*

\*

\* @author hp

\*/

public class CommandDesignPattern {

public static void main(String[] args) {

Stock abcStock = new Stock();

BuyStock buyStockOrder = new BuyStock(abcStock);

SellStock sellStockOrder = new SellStock(abcStock);

Broker broker = new Broker();

broker.takeOrder(buyStockOrder);

broker.takeOrder(sellStockOrder);

broker.placeOrders();

}

}

package command.design.pattern;

/\*\*

\*

\* @author hp

\*/

public interface Order {

void execute();

}

package command.design.pattern;

/\*\*

\*

\* @author hp

\*/

public class SellStock implements Order {

private Stock abcStock;

public SellStock(Stock abcStock){

this.abcStock = abcStock;

}

public void execute() {

abcStock.sell();

}

}

package command.design.pattern;

/\*\*

\*

\* @author hp

\*/

public class Stock {

private String name = "ABC";

private int quantity = 10;

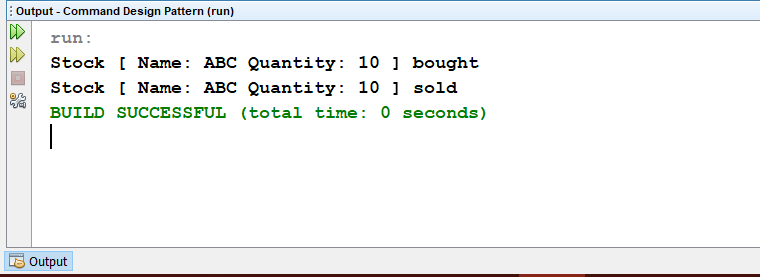
public void buy(){

System.out.println("Stock [ Name: "+name+" Quantity: " + quantity +" ] bought");

}

public void sell(){

System.out.println("Stock [ Name: "+name+" Quantity: " + quantity +" ] sold");}



1. **TEMPLATE PATTERN**

package templatepattern;

/\*\*

\*

\* @author hp

\*/

public class GlassHouse extends HouseTemplate

{

GlassHouse()

{

}

@Override

public void build\_windows() {

System.out.println("Glass windows has been created.\n");

}

@Override

public void build\_walls() {

System.out.println("Glass Walls has been created.");

}

@Override

public void build\_foundations() {

System.out.println("Glass foundations has been created.");

}

@Override

public void build\_pillars() {

System.out.println("Glass pillars has been created.");

}

package templatepattern;

/\*\*

\*

\* @author hp

\*/

public abstract class HouseTemplate {

//Constructor//

HouseTemplate()

{

}

public void build\_house()

{

build\_foundations();

build\_pillars();

build\_walls();

build\_windows();

}

public abstract void build\_windows();

public abstract void build\_walls();

public abstract void build\_foundations();

public abstract void build\_pillars();

}

package templatepattern;

public class TemplatePattern {

public static void main(String[] args) {

HouseTemplate object1 = new GlassHouse();

HouseTemplate object2 = new WoodenHouse();

object1.build\_house();

object2.build\_house();

}

}

package templatepattern;

public class WoodenHouse extends HouseTemplate {

WoodenHouse()

{

}

@Override

public void build\_windows() {

System.out.println("Wooden windows has been created.\n");

}

@Override

public void build\_walls() {

System.out.println("Wooden walls has been created.");

}

@Override

public void build\_foundations() {

System.out.println("Wooden foundations has been created.");

}

@Override

public void build\_pillars() {

System.out.println("Wooden pillars has been created.");

}}

