LEVEL 5

ANALYSIS, DESIGN AND IMPLEMENTATION

Code from the Lecturer Guide

Modification History

|  |  |  |
| --- | --- | --- |
| **Version** | **Date** | **Revision Description** |
| V1.0 | October 2011 | For release |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

© NCC Education Limited, 2011

All Rights Reserved

The copyright in this document is vested in NCC Education Limited. The document must not be reproduced by any means, in whole or in part, or used for manufacturing purposes, except with the prior written permission of NCC Education Limited and then only on condition that this notice is included in any such reproduction.

Published by: NCC Education Limited, The Towers, Towers Business Park, Wilmslow Road, Didsbury, Manchester M20 2EZ, UK.

Tel: +44 (0) 161 438 6200 Fax: +44 (0) 161 438 6240 Email: info@nccedu.com <http://www.nccedu.com>

**CONTENTS**

[Topic 7: Design Patterns 1 5](#_Toc307814729)

[7.5 Laboratory Sessions 5](#_Toc307814730)

[7.7 Tutorial Notes 7](#_Toc307814731)

[Topic 8: Design Patterns 2 9](#_Toc307814732)

[8.5 Laboratory Sessions 9](#_Toc307814733)

[8.7 Tutorial Notes 11](#_Toc307814734)

[Topic 9: Elements of Good Design 13](#_Toc307814735)

[9.5 Laboratory Sessions 13](#_Toc307814736)

[Topic 10: Redesign and Implementation 17](#_Toc307814737)

[10.5 Laboratory Sessions 17](#_Toc307814738)

Introduction

This document contains all of the code snippets from the Lecturer Guide for Analysis, Design and Implementation. This is presented in Word format to allow you to transfer it to other formats as necessary. Please note that the section numbering in this document is the same that in the Lecturer Guide to make it easier to locate the relevant code.

# Design Patterns 1

## Laboratory Sessions

Activity 2 Suggested Answer:

abstract class Account {

private int balance;

private double interestRate;

public Account(int bal) {

balance = bal;

}

public boolean deposit (int val) {

balance += val;

return true;

}

public boolean withdraw (int val) {

balance -= val;

return true;

}

public int getBalance() {

return balance;

}

public void setBalance (int b) {

balance = b;

}

public void setInterest (double i) {

interestRate = i;

}

public double getInterest() {

return interestRate;

}

public void applyInterest() {

interestRate = getBalance() \* getInterest();

}

abstract public void monthlyUpkeep();

}

public class HighInterest extends Account {

public HighInterest(int bal) {

super(bal);

setInterest (0.1);

}

public boolean withdraw (int val) {

setInterest (0.02);

return super.withdraw (val);

}

public void monthlyUpkeep() {

setInterest (0.1);

}

}

public class CurrentAccount extends Account {

public CurrentAccount(int bal) {

super(bal);

setInterest (0.03);

}

public void monthlyUpkeep() {

}

}

public class InvestmentAccount extends Account {

public InvestmentAccount(int bal) {

super(bal);

setInterest (0.15);

}

public boolean withdraw (int val) {

return false;

}

public void monthlyUpkeep() {

}

}

public class AccountFactory {

public static Account setupAccount(String type, int amount) {

Account acc;

if (type.equals("current")) {

acc = new CurrentAccount(amount);

} else if (type.equals("investment")) {

acc = new InvestmentAccount(amount);

} else {

acc = new HighInterest(amount);

}

acc.setBalance(amount);

return acc;

}

}

## Tutorial Notes

Exercise 1 Suggested Answer:

public class AbstractFactory {

public static AccountFactory getBank (String bank) {

AccountFactory af;

if (bank.equals ("bank 1")) {

af = new BankOne();

}

else {

af = new BankTwo();

}

return af;

}

}

public class AccountFactory extends Bank{

public Account setupAccount(String type, int amount) {

Account acc;

if (type.equals("current")) {

acc = new CurrentAccount(amount);

} else if (type.equals("investment")) {

acc = new InvestmentAccount(amount);

} else {

acc = new HighInterest(amount);

}

acc.setBalance(amount);

acc.setOverdraft(getOverdraft());

return acc;

}

}

public class Bank {

private int defaultOverdraft;

void setOverdraft (int o) {

defaultOverdraft = o;

}

int getOverdraft() {

return defaultOverdraft;

}

}

public class BankOne extends AccountFactory {

public BankOne() {

super();

setOverdraft(1000);

}

}

public class BankTwo extends AccountFactory {

public BankTwo() {

super();

setOverdraft(2000);

}

}

# Design Patterns 2

## Laboratory Sessions

Activity 1 Suggested Answer:

import java.util.\*;

public class FlyweightFactory {

HashMap<String,MenuEntry> items;

public FlyweightFactory() {

items = new HashMap<String,MenuEntry>();

}

public void addItem (String name, String item, double price) {

MenuEntry me;

me = new MenuEntry (item, price);

items.put (name, me);

}

public MenuEntry getItem (String name, String description, double price) {

if (items.get (name) == null) {

addItem (name, description, price);

}

return items.get (name);

}

}

public class Customer {

ArrayList<MenuEntry> orders;

public Customer() {

orders = new ArrayList<MenuEntry>();

}

void addOrder (MenuEntry me) {

orders.add (me);

System.out.println ("Bing!");

}

double getCost() {

double total = 0.0;

for (MenuEntry me : orders) {

total += me.getPrice();

}

return total;

}

ArrayList<String> getOrders() {

ArrayList<String> desc = new ArrayList<String>();

for (MenuEntry me : orders) {

desc.add (me.getDescription());

}

return desc;

}

}

public class MenuEntry {

String description;

double price;

public MenuEntry(String d, double p) {

description = d;

price = p;

}

public double getPrice() {

return price;

}

public String getDescription() {

return description;

}

}

public class MainClass {

public static void main (String args[]) {

FlyweightFactory myFact = new FlyweightFactory();

Customer cust = new Customer();

MenuEntry me;

ArrayList<String> desc;

me = myFact.getItem ("starter 1", "It's a starter", 10.0);

cust.addOrder (me);

me = myFact.getItem ("starter 2", "It's another starter", 10.0);

cust.addOrder (me);

me = myFact.getItem ("starter 1", "It's a third starter", 10.0);

cust.addOrder (me);

desc = cust.getOrders();

for (String str : desc) {

System.out.println (str);

}

System.out.println ("The total price is " + cust.getCost());

}

}

## Tutorial Notes

Exercise 1 Suggested Answer:

public class MenuFacade {

private ArrayList<Customer> myCustomers;

private FlyweightFactory myFact;

public MenuFacade() {

myFact= new FlyweightFactory();

myCustomers = new ArrayList<Customer>();

}

Customer getCustomer (int number) {

teturn myCustomers.get(number);

}

MenuEntry getMenuEntry (String str, String desc, float price) {

MenuEntry me = myFact.getItem (str, desc, price);

Return me;

}

void addOrderToCustomer (Customer cust, MenuEntry me) {

cust.addOrder (me);

}

}

Exercise 2 Suggested Answer:

public class MenuFacade {

private ArrayList<Customer> myCustomers;

private FlyweightFactory myFact;

private MenuFacade menu;

// Implements the singleton.

public static MenuFacade getMenu() {

if (menu == null) {

menu = new MenuFacade();

}

return menu;

}

private MenuFacade() {

myFact= new FlyweightFactory();

myCustomers = new ArrayList<Customer>();

}

Customer getCustomer (int number) {

teturn myCustomers.get(number);

}

MenuEntry getMenuEntry (String str, String desc, float price) {

MenuEntry me = myFact.getItem (str, desc, price);

return me;

}

void addOrderToCustomer (Customer cust, MenuEntry me) {

cust.addOrder (me);

}

}

# Elements of Good Design

## Laboratory Sessions

Activity 1 Suggested Answer:

**Append operation:**

public static void main(String[] args) {

Date time = new Date();

int iterations[] = {100, 1000, 10000, 100000};

long now = time.getTime();

long then;

double total;

String str = "blue";

for (int j = 0; j < iterations.length; j++) {

for (int i = 0; i < iterations[j]; i++) {

str += "blue";

}

time = new Date();

then = time.getTime();

total = then - now;

System.out.println ("Method took " + total + " milliseconds for " + iterations[j] + " iteration.");

}

}

**For Stringbuilder:**

public static void main(String[] args) {

Date time = new Date();

int iterations[] = {100, 1000, 10000, 100000};

long now = time.getTime();

long then;

double total;

StringBuilder str = new StringBuilder ("blue");

for (int j = 0; j < iterations.length; j++) {

for (int i = 0; i < iterations[j]; i++) {

str.append ("blue");

}

time = new Date();

then = time.getTime();

total = then - now;

System.out.println ("Method took " + total + " milliseconds for " + iterations[j] + " iteration.");

}

}

**Factorials**

public static HashMap<Integer, Integer> cache;

public static int fibb(int num) {

if (num <= 2) {

return 1;

} else {

return (fibb(num - 1) + fibb(num - 2));

}

}

public static int fibbCache(int num) {

int c;

if (cache.containsKey(num) == true) {

c = cache.get(num);

return c;

}

if (num <= 2) {

return 1;

} else {

c = (fibb(num - 1) + fibb(num - 2));

cache.put(num, c);

return c;

}

}

public static int fibbLoop(int num) {

int next = 0, current = 1, old = 0;

for (int i = 0; i < num - 1; i++) {

next = current + old;

old = current;

current = next;

}

return current;

}

public static void main(String[] args) {

cache = new HashMap<Integer, Integer>();

Date time = new Date();

int iterations[] = {10, 20, 30, 40, 50};

long now = time.getTime();

int fact;

long then;

double total;

for (int j = 0; j < iterations.length; j++) {

fact = fibbCache(iterations[j]);

time = new Date();

then = time.getTime();

total = then - now;

System.out.println("Method took " + total + " milliseconds for " + iterations[j] + " iteration: " + fact);

}

}

}

Activity 2:

public void doTheThings() {

int[] sizes = {10, 20, 30, 40, 50, 60, 70, 80, 90, 100};

double circumference, half, doub, quart, four;

for (int i = 0; i < sizes.length; i++) {

for (int j = 0; j < 1000; j++) {

circumference = Math.PI \* sizes[i];

doub = Math.PI \* sizes[i] \* 2;

four = Math.PI \* sizes[i] \* 4;

half = (Math.PI \* sizes[i]) / 2;

quart = (Math.PI \* sizes[i]) / 4;

System.out.println ("Stats for Circle " + j + ": "

+ quart + ", " + half + ", " + circumference + ", "

+ doub + ", " + four);

}

}

}

Suggested Answer:

public static void doTheThings() {

int[] sizes = {10, 20, 30, 40, 50, 60, 70, 80, 90, 100};

double circumference, half, doub, quart, four;

StringBuilder str = new StringBuilder();

for (int i = 0; i < sizes.length; i++) {

circumference = Math.PI \* sizes[i];

for (int j = 0; j < 1000; j++) {

doub = circumference \* 2;

four = circumference \* 4;

half = circumference / 2;

quart = circumference / 4;

str.append ("Stats for Circle " + j + ": "

+ quart + ", " + half + ", "

+ circumference + ", " + doub + ", "

+ four);

}

}

System.out.println (str.toString());

1. **Redesign and Implementation**

## Laboratory Sessions

Activity 1 Suggested Answer:

import java.util.\*;

public interface ModelObserver {

public void vehicleAdded (String licence);

public void vehicleRemoved (String licence);

public void customerAdded (String id);

public void customerRemoved (String id);

}

public class ModelFacade {

private static final int EVENT\_VEHICLE\_ADDED = 0;

private static final int EVENT\_VEHICLE\_REMOVED = 1;

private static final int EVENT\_CUSTOMER\_ADDED = 2;

private static final int EVENT\_CUSTOMER\_REMOVED = 3;

ArrayList <ModelObserver> myListeners;

public ModelFacade() {

myListeners = new ArrayList<ModelObserver>();

}

public void addModelListener (ModelObserver m) {

myListeners.add (m);

}

public void removeModelListener (ModelObserver m) {

myListeners.remove (m);

}

public void notifyListeners(int type, String data) {

for (ModelObserver m : myListeners) {

switch (type) {

case EVENT\_VEHICLE\_ADDED:

m.vehicleAdded (data);

break;

case EVENT\_VEHICLE\_REMOVED:

m.vehicleRemoved (data);

break;

case EVENT\_CUSTOMER\_ADDED:

m.customerAdded (data);

break;

case EVENT\_CUSTOMER\_REMOVED:

m.customerRemoved (data);

break;

}

}

}

}

Activity 2 Suggested Answer:

import java.util.\*;

public class Payroll {

ArrayList<StaffMember> myStaff;

public Payroll() {

myStaff = new ArrayList<StaffMember>();

}

public void addStaffMember (String type, String ID, String name) {

StaffMember sm;

if (type.equals ("driver")) {

sm = new Driver();

}

else {

sm = new Owner();

}

sm.setID (ID);

sm.setName (name);

myStaff.add (sm);

}

public StaffMember findStaffMember (String ID) {

for (StaffMember sm : myStaff) {

if (sm.getID().equalsIgnoreCase (ID)) {

return sm;

}

}

return null;

}

}

abstract class StaffMember {

private String ID;

private String name;

public void setID (String s) {

ID = s;

}

public String getID() {

return ID;

}

public void setName (String s) {

name = s;

}

public String getName() {

return name;

}

abstract String getType();

}

public class Owner extends StaffMember {

String getType() {

return "Owner";

}

}

public class Driver extends StaffMember {

String getType() {

return "driver";

}

}