МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

«БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ» ФАКУЛЬТЕТ ЭЛЕКТРОННО-ИНФОРМАЦИОННЫХ СИСТЕМ

Кафедра интеллектуальных информационных технологий

Отчет по лабораторной работе №6

Специальность ПО-5(о)

Выполнил

Д. С. Бриштен,

студент группы ПО-5

Проверил

А. А. Крощенко

ст. преп. каф. ИИТ

Брест 2021

Цель работы: приобрести навыки применения паттернов проектирования при решении практических задач с использованием языка Java.

В данной лабораторной работе переделаны 1 и 2 задания.

Вариант 6

Задание 1. Музыкальный магазин. Должно обеспечиваться одновременное обслуживание нескольких покупателей. Магазин должен предоставлять широкий выбор товаров различных музыкальных направлений.

Выполнение:

Код программы

/\*

    SINGLETON

    Необходимо, чтобы для класса существовал один экземпляр. То есть паттерн гарантирует существование

    одного объекта для класса, что и демонстрируется в классе-клиенте.

\*/

final class MusikStore {

    private static MusikStore instance;

    public String value;

    private MusikStore(String value) {

        this.value = value;

    }

    public static synchronized MusikStore getInstance(String value) {

        if (instance == null) {

            instance = new MusikStore(value);

        }

        return instance;

    }

}

public class Runner {

    public static void main(String[] args) {

        MusikStore products = MusikStore.getInstance("Folk Tools || Studio || Guitars");

        System.out.println(products.value);

        MusikStore materials = MusikStore.getInstance("Harmony || Drums");

        System.out.println(materials.value);

    }

}

Спецификация вывода:

Используя переопределённый метод Move()

<value> - перечень направлений товара

Пример:



Задание 2. Учетная запись покупателя книжного интернет-магазина. Предусмотреть различные уровни учетки в зависимости от активности покупателя. Дополнительные уровни добавляют функциональные возможности и открывают доступ к уникальным предложениям.

Выполнение:

Код программы

/\*

    DECORATOR

    Когда надо динамически добавлять к объекту новые функциональные возможности.

    В моём случае в зависимости от уровня есть разные возможности.

\*/

import java.util.ArrayList;

public class Accounting2 {

    public static void main(String[] args)

    {

        ArrayList<Human> People = new ArrayList<Human>();

        Human Ivan = new Low();

        Human Petr = new Average();

        Human Vera = new High();

        People.add(Ivan);

        People.add(Petr);

        People.add(Vera);

        System.out.println("|Stock for New Year and Christmas|");

        for (Human buyer: People)

        {

            System.out.print("The " + buyer.getName() + " reader ");

            if(buyer.GetActivity() > 9)

            {

                buyer = new Discount(buyer);

                buyer = new Gift(buyer);

                System.out.println("get " + buyer.getName());

            }

            else if (buyer.GetActivity() < 5) { System.out.println("nothing get"); }

            else

            {

                buyer = new Discount(buyer);

                System.out.println("get " + buyer.getName());

            }

        }

    }

}

abstract class Human

{

    public String name;

    public Human(String n)

    {

        this.name = n;

    }

    public String getName() { return this.name; }

    public abstract int GetActivity();

}

class High extends Human

{

    public High()

    {

        super("high-level");

    }

    @Override

    public int GetActivity()

    {

        return 10;

    }

}

class Average extends Human

{

    public Average()

    {

        super("average-level");

    }

    @Override

    public int GetActivity()

    {

        return 8;

    }

}

class Low extends Human

{

    public Low()

    {

        super("low-level");

    }

    @Override

    public int GetActivity()

    {

        return 4;

    }

}

abstract class HumanDecorator extends Human

{

    protected Human human;

    public HumanDecorator(String name, Human human)

    {

        super(name);

        this.human = human;

    }

}

class Discount extends HumanDecorator

{

    public Discount(Human p)

    {

        super(p.getName() + "at 20% discount ", p);

    }

    @Override

    public int GetActivity()

    {

        return human.GetActivity() - 3;

    }

}

class Gift extends HumanDecorator

{

    public Gift(Human p)

    {

        super(p.getName() + " and a gift", p);

    }

    @Override

    public int GetActivity()

    {

        return human.GetActivity() - 5;

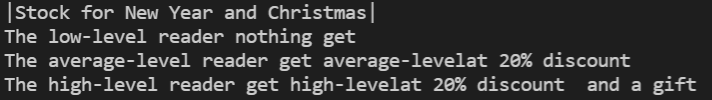
    }

}

Спецификация вывода:

Метод Human.getName(): <name> <текст>

Пример:



Задание 3. Проект Принтер. Предусмотреть выполнение операций (печать, загрузка бумаги, извлечение зажатой бумаги, заправка картриджа), режимы – ожидание, печать документа, зажатие бумаги, отказ – при отсутствии бумаги или краски, атрибуты – модель, количество листов в лотке, % краски в картридже, вероятность зажатия.

Выполнение:

Код программы

import java.util.Scanner;

//  COMMAND

/\*

    Необходимо обеспечить выполнение очереди запросов, а также их возможную отмену.

\*/

public class Printer {

    public static void main(String[] args) {

        Scanner in = new Scanner(System.in);

        System.out.print("Enter number of sheets: ");

        int NS = in.nextInt();

        in.nextLine();

        System.out.print("Enter paint percentage: ");

        int PP = in.nextInt();

        in.nextLine();

        Pult pult = new Pult();

        Print print = new Print("Bosh", NS, PP);

        pult.SetCommand(new PrinterOnCommand(print));

        if (NS == 0 || PP == 0) { pult.PressOff(); in.close(); return; }

        print.infoPrinter();

        pult.PressOn();

        pult.SetAction(new RefillCommand(print));

        pult.Execute();

        print.waiting();

        pult.SetAction(new LoadOnCommand(print));

        pult.Execute();

        print.clampPaper();

        pult.SetAction(new PrintOnCommand(print));

        pult.Execute();

        print.printDocument();

        pult.SetAction(new ExtractionOnCommand(print));

        pult.Execute();

        print.waiting();

        pult.SetCommand(new PrinterOnCommand(print));

        pult.PressOff();

        in.close();

        return;

    }

}

interface Regimes

{

    void waiting();

    void printDocument();

    void clampPaper();

}

interface StartOperation

{

    void Start();

}

interface Command

{

    void Execute();

    void Undo();

}

// Receiver - Получатель

class Print implements Regimes

{

    private String model;

    private int numSheet, paintPercentage, pinchingProbability;

    private int max = 100, min = 0;

    Print(String model, int numSheet, int paintPercentage)

    {

        this.model = model;

        this.numSheet = numSheet;

        this.paintPercentage = paintPercentage;

        this.pinchingProbability = (int)(Math.random()\*((max-min)+1))+min;

    }

    public void On() { System.out.println("Printer On!"); }

    public void Off() { System.out.println("Printer Off!"); }

    public void Printing() { System.out.println("Printing..."); }

    public void loadPaper() { System.out.println("Paper loading..."); }

    public void extractionPaper() { System.out.println("Paper extraction...") ;}

    public void refillCartridge() { System.out.println("Cartridge refill...") ;}

    public void infoPrinter()

    {

        System.out.println("\nModel \'" + model + "\', number of sheets = " + numSheet +

        ", percentage of paint = " + paintPercentage + "% and pinching probability = " +

        pinchingProbability + "%");

    }

    public void waiting()

    {

        try {

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

            Thread.sleep(10000);

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

        } catch (InterruptedException e) {

            e.printStackTrace();

        }

    }

    public void printDocument()

    {

        try {

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

            Thread.sleep(15000);

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

        } catch (InterruptedException e) {

            e.printStackTrace();

        }

    }

    public void clampPaper()

    {

        try {

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

            Thread.sleep(5000);

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

        } catch (InterruptedException e) {

            e.printStackTrace();

        }

    }

}

// ========================================================

class PrinterOnCommand implements Command

{

    Print print;

    public PrinterOnCommand(Print printSet) { print = printSet; }

    public void Execute() { print.On(); }

    public void Undo() { print.Off(); }

}

class PrintOnCommand implements StartOperation

{

    Print print;

    public PrintOnCommand(Print printSet) { print = printSet; }

    public void Start() { print.Printing(); }

}

class LoadOnCommand implements StartOperation

{

    Print print;

    public LoadOnCommand(Print printSet) { print = printSet; }

    public void Start() { print.loadPaper(); }

}

class ExtractionOnCommand implements StartOperation

{

    Print print;

    public ExtractionOnCommand(Print printSet) { print = printSet; }

    public void Start() { print.extractionPaper(); }

}

class RefillCommand implements StartOperation

{

    Print print;

    public RefillCommand(Print printSet) { print = printSet; }

    public void Start() { print.refillCartridge(); }

}

// ==============================================================

class Pult

{

    Command command;

    StartOperation start;

    public Pult() { }

    public void SetCommand(Command com) { command = com; }

    public void SetAction(StartOperation operation) { start = operation; }

    public void PressOn() { command.Execute(); }

    public void PressOff() { command.Undo(); }

    public void Execute() { start.Start(); }

}

Спецификация ввода:

<NS> - количество листов

<PP> - процент оставшейся краски

Спецификация вывода:

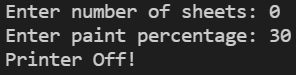
Метод info(): описание принтера

Метод execute(): осуществляет выполнение установленного действия

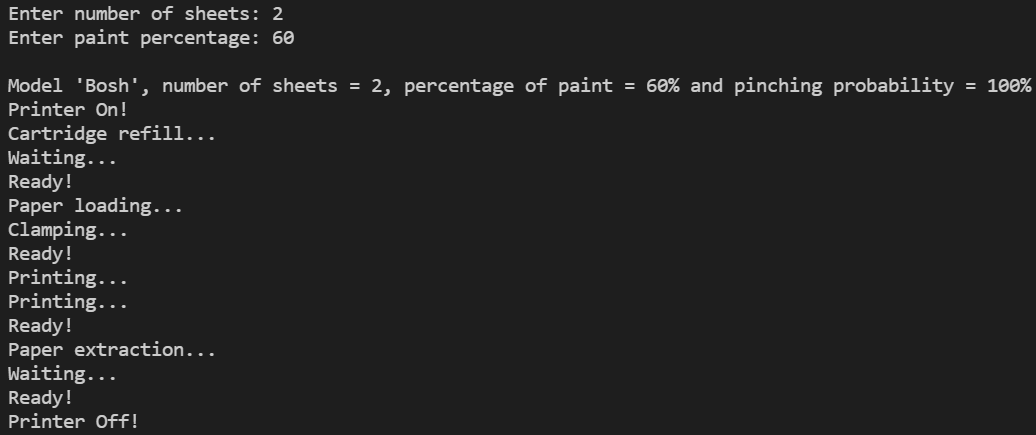
Методы интерфейса Regemes: обеспечивают задержку исполнения кода

Пример:

Случай, когда листов в принтере не оказалось – он выключается



Если всё в порядке, то действия выполнятся следующим образом



Вывод: Ознакомился с паттернами проектирования при решении практических задач в языке программирования Java.