МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
«БРЕСТСКИЙГОСУДАРСТВЕННЫЙТЕХНИЧЕСКИЙУНИВЕРСИТЕТ»

ФАКУЛЬТЕТ ЭЛЕКТРОННО-ИНФОРМАЦИОННЫХ СИСТЕМ

Кафедра интеллектуальных информационных технологий

Отчет по лабораторной работе №6

Специальность ПО3

Выполнила Р.И.
Гаврилюк, студентка
группы ПО3

Проверил А.А.
Крощенко,
ст. преп. кафедры ИИТ,
«——» ———— 2020 г.

Брест 2020

Вариант 6

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

Задание 1.

Магазин по производству смартфонов. Обеспечить создание нескольких различных моделей мобильных телефонов с заранее выбранными характеристиками.

Решение

Паттерн Строитель, который позволяет создавать сложные объекты пошагово.

Код программы

Task_1.java

```
class Task_1{
    public static void main(String[] args) {
        Director director = new Director();

        SmartphoneBuilder builder = new SmartphoneBuilder();

        director.constructSmarthoneDAY(builder);

        Smartphone sm = builder.getResult();
        System.out.println(sm.print() + "\n");

        director.constructSmarthoneNIGHT(builder);

        sm = builder.getResult();
        System.out.println(sm.print() + "\n");

        director.constructSmarthoneSTAR(builder);

        sm = builder.getResult();
        System.out.println(sm.print() + "\n");
    }
}
```

Director.java

```
public class Director {

    public void constructSmarthoneDAY(Builder builder) {
        builder.setSmartphoneType(SmartphoneType.SMARTPHONE_DAY);
        builder.setColor(new Color("white"));
        builder.setName("DAY");
        builder.setNFC(new NFC("4.1"));
    }

    public void constructSmarthoneNIGHT(Builder builder) {
        builder.setSmartphoneType(SmartphoneType.SMARTPHONE_NIGHT);
```

```
        builder.setColor(new Color("black"));
        builder.setName("NIGHT");
        builder.setFingerprintScanner(new FingerprintScanner("8.9"));
    }

    public void constructSmarthoneSTAR(Builder builder) {
        builder.setSmartphoneType(SmartphoneType.SMARTPHONE_STAR);
        builder.setColor(new Color("yellow"));
        builder.setName("STAR");
        builder.setNFC(new NFC("3.2"));
        builder.setFingerprintScanner(new FingerprintScanner("10.0"));
    }
}
```

SmartphoneBuilder.java

```
public class SmartphoneBuilder implements Builder {
    private SmartphoneType type;
    private String name;
    private Color color;
    private NFC nfc;
    private FingerprintScanner FingerprintScanner;

    @Override
    public void setSmartphoneType(SmartphoneType type){
        this.type = type;
    }

    @Override
    public void setName(String name) {
        this.name = name;
    }

    @Override
    public void setColor(Color color) {
        this.color = color;
    }

    @Override
    public void setNFC(NFC nfc) {
        this.nfc = nfc;
    }

    @Override
    public void setFingerprintScanner(FingerprintScanner FingerprintScanner) {
        this.FingerprintScanner = FingerprintScanner;
    }

    public Smartphone getResult() {
        return new Smartphone(this.type, this.name, this.color, this.nfc, this.Fingerprint
Scanner);
    }
}
```

Smartphone.java

```
public class Smartphone {
    private SmartphoneType type;
    private String name;
    private Color color;
    private NFC nfc;
    private FingerprintScanner FingerprintScanner;

    public Smartphone (SmartphoneType type, String name, Color color, NFC nfc,
    FingerprintScanner FingerprintScanner) {
        this.setSmartphoneType(type);
        this.setName(name);
        this.setColor(color);
        this.setNFC(nfc);
        this.setFingerprintScanner(FingerprintScanner);
    }

    public void setSmartphoneType(SmartphoneType type){
        this.type = type;
    }

    public void setName(String name) {
        this.name = name;
    }

    public void setColor(Color color) {
        this.color = color;
    }

    public void setNFC(NFC nfc) {
        this.nfc = nfc;
    }

    public void setFingerprintScanner(FingerprintScanner FingerprintScanner) {
        this.FingerprintScanner = FingerprintScanner;
    }

    public SmartphoneType setSmartphoneType(){
        return this.type;
    }

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

    public Color setColor() {
        return this.color;
    }

    public NFC setNFC() {
        return this.nfc;
    }

    public FingerprintScanner setFingerprintScanner() {
```

```
        return this.FingerprintScanner;
    }

    public String print() {
        String info = "";
        info += "Type of smartphone: " + this.type + "\n";
        info += "Name: " + this.name + "\n";
        info += "Color: " + this.color.getColor() + "\n";
        if (this.nfc != null) {
            info += "NFC: " + this.nfc.getNFCVersion() + "\n";
        } else {
            info += "NFC: -" + "\n";
        }
        if (this.FingerprintScanner != null) {
            info += "Fingureprint Scanner: " + this.FingerprintScanner.getScannerVersion()
 + "\n";
        } else {
            info += "Fingureprint Scanner: -" + "\n";
        }
        return info;
    }
}
```

SmartphoneManualBuilder.java

```
public class SmartphoneManualBuilder implements Builder {
    private SmartphoneType type;
    private String name;
    private Color color;
    private NFC nfc;
    private FingerprintScanner FingerprintScanner;

    @Override
    public void setSmartphoneType(SmartphoneType type){
        this.type = type;
    }

    @Override
    public void setName(String name) {
        this.name = name;
    }

    @Override
    public void setColor(Color color) {
        this.color = color;
    }

    @Override
    public void setNFC(NFC nfc) {
        this.nfc = nfc;
    }

    @Override
    public void setFingerprintScanner(FingerprintScanner FingerprintScanner) {
        this.FingerprintScanner = FingerprintScanner;
```

```
    }

    public Manual getResult() {
        return new Manual(this.type, this.name, this.color, this.nfc, this.FingerprintScan
ner);
    }
}
```

Manual.java

```
public class Manual {
    private SmartphoneType type;
    private String name;
    private Color color;
    private NFC nfc;

    public Smartphone (SmartphoneType type, String name, Color color, NFC nfc,
    FingerprintScanner FingerprintScanner) {
        this.setSmartphoneType(type);
        this.setName(name);
        this.setColor(color);
        this.setNFC(nfc);
    }

    public void setSmartphoneType(SmartphoneType type){
        this.type = type;
    }

    public void setName(String name) {
        this.name = name;
    }

    public void setColor(Color color) {
        this.color = color;
    }

    public void setNFC(NFC nfc) {
        this.nfc = nfc;
    }

    public SmartphoneType setSmartphoneType(){
        return this.type;
    }

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

    public Color setColor() {
        return this.color;
    }

    public NFC setNFC() {
        return this.nfc;
    }
```

```
    public String print() {
        String info = "";
        info += "Type of smartphone: " + this.type + "\n";
        info += "Name: " + this.name + "\n";
        info += "Color: " + this.color.getColor() + "\n";
        if (this.nfc != null) {
            info += "NFC: " + this.nfc.getNFCVersion() + "\n";
        } else {
            info += "NFC: -" + "\n";
        }
        return info;
    }
}
```

SmartphoneType.java

```
public enum SmartphoneType {
    SMARTPHONE_STAR, SMARTPHONE_DAY, SMARTPHONE_NIGHT
}
```

FingerprintScanner.java

```
class FingerprintScanner {
    private String version;

    public FingerprintScanner(String version){
        this.setScannerVersion(version);
    }

    public void setScannerVersion(String version) {
        this.version = version;
    }

    public String getScannerVersion() {
        return this.version;
    }
}
```

NFC.java

```
class NFC {
    private String version;

    public NFC(String version){
        this.setNFCVersion(version);
    }

    public void setNFCVersion(String version) {
        this.version = version;
    }

    public String getNFCVersion() {
        return this.version;
    }
}
```

Color.java

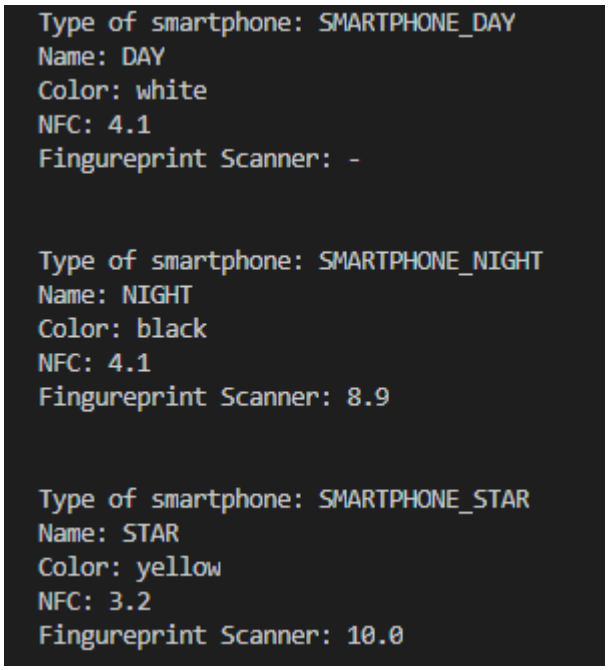
```
class Color {
    private String color;

    public Color(String color){
        this.setColor(color);
    }

    public void setColor(String color) {
        this.color = color;
    }

    public String getColor() {
        return this.color;
    }
}
```

Рисунки с результатами работы программы



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.

Решение

Паттерн Декоратор, позволяет динамически добавлять объектам новую функциональность, оборачивая их в разные обертки.

Код программы

Main.java

```
class Task_2{
    public static void main(String[] args) {
        Client cl = new Client();
        System.out.println(cl.getLevel());
        System.out.println(cl.addToFavourites("book"));
        System.out.println(cl.leaveMark("book", "8"));

        cl.toSecondLevel();
        System.out.println(cl.getLevel());
        System.out.println(cl.addToFavourites("book"));
        System.out.println(cl.leaveMark("book", "8"));

        cl.toThirdLevel();
        System.out.println(cl.getLevel());
        System.out.println(cl.addToFavourites("book"));
        System.out.println(cl.leaveMark("book", "8"));
    }

}
```

Level.java

```
interface Level {
    public String getLevel();
    public String buyBook(String name);
    public String addToFavourites(String name);
    public String leaveMark(String name, String mark);
}
```

FirstLevel.java

```
class FirstLevel implements Level {

    private String mLevelName;

    public FirstLevel() {
        this.mLevelName = "1";
    }
```

```
    @Override
    public String getLevel() {
        return this.mLevelName;
    }

    @Override
    public String buyBook(String name) {
        return name;
    }

    @Override
    public String addToFavourites(String name) {
        return "you cann't add to favourites on level 1.";
    }

    @Override
    public String leaveMark(String name, String mark) {
        return "you cann't leave mark on level 1.";
    }

}
```

Decorator.java

```
class Decorator implements Level {

    protected Level mLevel;
    protected String mLevelName;

    public Decorator(Level level) {
        this.mLevel = level;
    }

    @Override
    public String getLevel() {
        return this.mLevel.getLevel();
    }

    @Override
    public String buyBook(String name) {
        return this.mLevel.buyBook(name);
    }

    @Override
    public String addToFavourites(String name) {
        return this.mLevel.addToFavourites(name);
    }

    @Override
    public String leaveMark(String name, String mark) {
        return this.mLevel.leaveMark(name, mark);
    }

}
```

SecondLevel.java

```
class SecondLevel extends Decorator {

    public SecondLevel(Level level) {
        super(level);
        this.mLevelName = "2->" + super.getLevel();
    }

    @Override
    public String getLevel() {
        return this.mLevelName;
    }

    @Override
    public String buyBook(String name) {
        return this.mLevel.buyBook(name);
    }

    @Override
    public String addToFavourites(String name) {
        return "you have added book " + name + " to favourites.";
    }

    @Override
    public String leaveMark(String name, String mark) {
        return "you cann't leave mark on level 2.";
    }
}
```

ThirdLevel.java

```
class ThirdLevel extends Decorator {

    public ThirdLevel(Level level) {
        super(level);
        this.mLevelName = "3->" + super.getLevel();
    }

    @Override
    public String getLevel() {
        return this.mLevelName;
    }

    @Override
    public String buyBook(String name) {
        return this.mLevel.buyBook(name);
    }

    @Override
    public String addToFavourites(String name) {
        return "you have added book " + name + " to favourites.";
    }

    @Override
    public String leaveMark(String name, String mark) {
```

```
        return "you have left mark" + mark + " on book " + name + ".";
    }

}
```

Client.java

```
class Client {
    private Level mClientLevel;

    Client() {
        this.mClientLevel = new FirstLevel();
    }

    public void toSecondLevel() {
        this.mClientLevel = new SecondLevel(this.mClientLevel);
    }

    public void toThirdLevel() {
        this.mClientLevel = new ThirdLevel(this.mClientLevel);
    }

    public String getLevel() {
        return this.mClientLevel.getLevel();
    }

    public String buyBook(String name) {
        return this.mClientLevel.buyBook(name);
    }

    public String addToFavourites(String name) {
        return this.mClientLevel.addToFavourites(name);
    }

    public String leaveMark(String name, String mark) {
        return this.mClientLevel.leaveMark(name, mark);
    }
}
```

Рисунки с результатами работы программы

```
1
you cann't add to favourites on level 1.
you cann't leave mark on level 1.
2->1
you have added book book to favourites.
you cann't leave mark on level 2.
3->2->1
you have added book book to favourites.
you have left mark8 on book book.
```

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.

Решение

Паттерн Состояние, в зависимости от состояния объект принтер меняет свое поведение.

Код программы

main.java

```
import java.util.Random;

class Task_3{
    public static void main(String[] args) {
        Printer printer = new Printer("model 7000", 30, 100, 0.6);

        int numPaper;
        String result;
        Random r = new Random();
        for(int i = 0; i < 10; ++i) {
            numPaper = 0 + (10 - 0) * r.nextInt();
            printer.print(numPaper);

            numPaper = 0 + (10 - 0) * r.nextInt();
            result = printer.print(numPaper);

            if(result.lastIndexOf("Clamping State.") != -1) {
                printer.extractClampingPaper();
            }

            if(result.lastIndexOf("Not ehough papers") != -1) {
                numPaper = 0 + (10 - 0) * r.nextInt();
                printer.loadPaper(numPaper);
            }

            if(result.lastIndexOf("Not ehough painters") != -1) {
                printer.refilleCartridge();
            }
        }
    }
}
```

State.java

```
abstract class State {
    Printer mPrinter;
```

```
    State(Printer printer) {
        this.mPrinter = printer;
    }

    public abstract String onPrint(int numPaper);
    public abstract String onLoadPaper(int numPaper);
    public abstract String onExtractClampingPaper();
    public abstract String onRefilleCartridge();
}
```

WaitState.java

```
class WaitState extends State {

    WaitState(Printer printer) {
        super(printer);
    }

    @Override
    public String onPrint(int numPaper) {
        this.mPrinter.changeState(new PrintState(this.mPrinter));
        return "WaitState --> PrintState.\n";
    }

    @Override
    public String onLoadPaper(int numPaper) {
        return "Paper was loaded.\nWaitState.\n";
    }

    @Override
    public String onExtractClampingPaper() {
        return "Paper was extracted.\nWaitState.\n";
    }

    @Override
    public String onRefilleCartridge() {
        this.mPrinter.setPaintVolume(100);
        return "Cartridge was refilled.\nWaitState.\n";
    }
}
```

PrintState.java

```
import java.util.Random;

class PrintState extends State {

    PrintState(Printer printer) {
        super(printer);
    }

    @Override
    public String onPrint(int numPaper) {
        Random r = new Random();
        double probality = 0 + (100 - 0) * r.nextDouble();
```

```
        if(probality <= this.mPrinter.getPinchingProbability() + 20 &&
        probality >= this.mPrinter.getPinchingProbability() - 20) {
            this.mPrinter.changeState(new ClampingState(this.mPrinter));
            return "Paper cann't be printed. It's claimed.\nPrintState --
> ClampingState.\n";
        }

        for(int i = 0; i < numPaper; ++i) {
            //5% = 1 page
            int numPapersInPrinter = this.mPrinter.getNumPapers();
            double paintVolume = this.mPrinter.getPaintVolume();
            if(paintVolume - 5 >= 0 && numPapersInPrinter > 0) {
                this.mPrinter.setPaintVolume(paintVolume - 5);
                this.mPrinter.setNumPapers(numPapersInPrinter - 1);
            }
            else {
                this.mPrinter.changeState(new FailureState(this.mPrinter, numPaper - i));
                if(paintVolume - 5 < 0) {
                    return "Other paper cann't be printed. Not enough painter in cartridge
.\nPrintState --> FailureState.\n";
                }
                if(numPapersInPrinter <= 0) {
                    return "Other paper cann't be printed. Not enough papers.\nPrintState
--> FailureState.\n";
                }
            }
        }

        this.mPrinter.changeState(new WaitState(this.mPrinter));
        return "Paper was printed.\nPrintState --> WaitState.\n";
    }

    @Override
    public String onLoadPaper(int numPaper) {
        return "Paper cann't be loaded.\nPrintState.\n";
    }

    @Override
    public String onExtractClampingPaper() {
        return "Paper cann't be clamped\nPrintState.\n";
    }

    @Override
    public String onRefilleCartridge() {
        return "Cartridge cann't be refilled.\nPrintState.\n";
    }
}
```

FailureState.java

```
class FailureState extends State {
    private int mNumUnprintedPapers;

    FailureState(Printer printer, int numUnprintedPapers) {
```

```
        super(printer);
        this.setNumUnprintedPapers(numUnprintedPapers);
    }

    public void setNumUnprintedPapers(int numUnprintedPapers) {
        this.mNumUnprintedPapers = numUnprintedPapers;
    }

    public int getNumUnprintedPapers() {
        return this.mNumUnprintedPapers;
    }

    @Override
    public String onPrint(int numPaper) {
        if(this.mPrinter.getPaintVolume() - 5 >= 0
        && this.mPrinter.getNumPapers() > 0) {
            this.mPrinter.changeState(new PrintState(this.mPrinter));
            this.mPrinter.print(this.mNumUnprintedPapers + numPaper);
            return "Waiting State --> PrintState\n";
        }
        else {
            if(this.mPrinter.getPaintVolume() - 5 < 0) {
                return "Other paper cann't be printed. Not enough painter in cartridge.\nF
ailureState.\n";
            }
            if(this.mPrinter.getNumPapers() <= 0) {
                return "Other paper cann't be printed. Not enough papers.\nFailureState.\n
";
            }
        }
        return "FailureState\n";
    }

    @Override
    public String onLoadPaper(int numPaper) {
        if(this.mPrinter.getPaintVolume() - 5 >= 0) {
            this.mPrinter.changeState(new WaitState(this.mPrinter));
            return "Paper was loaded. Enough painters in cartridge.\nFailureState --
> WaitState.\n";
        }
        else {
            return "Paper was loaded. Not ehough painters in cartridge.\nFailureState.\n";
        }
    }

    @Override
    public String onExtractClampingPaper() {
        this.mPrinter.changeState(new WaitState(this.mPrinter));
        return "Paper was extracted.\nFailureState.\n";
    }

    @Override
    public String onRefilleCartridge() {
        this.mPrinter.setPaintVolume(100);
        if(this.mPrinter.getNumPapers() > 0) {
```

```
            this.mPrinter.changeState(new WaitState(this.mPrinter));
            return "Cartridge was refilled. Enough papers.\nFailureState --
> WaitState.\n";
        }
        else {
            return "Cartridge was refilled. Not ehough papers.\nFailureState.\n";
        }
    }
}
```

ClaimingState.java

```
class ClampingState extends State {

    ClampingState(Printer printer) {
        super(printer);
    }

    @Override
    public String onPrint(int numPaper) {
        return "Paper cann't be printed.\nClamping State.\n";
    }

    @Override
    public String onLoadPaper(int numPaper) {
        return "Paper cann't be loaded.\nClampingState.\n";
    }

    @Override
    public String onExtractClampingPaper() {
        this.mPrinter.changeState(new WaitState(this.mPrinter));
        return "Paper was extracted.\nClampingState --> WaitState.\n";
    }

    @Override
    public String onRefilleCartridge() {
        this.mPrinter.setPaintVolume(100);
        return "Cartridge was refilled.\nClampingState.\n";
    }
}
```

IPrinter.java

```
interface IPrinter{
    String print(int numPaper);
    void loadPaper(int numPaper);
    void extractClampingPaper();
    void refilleCartridge();
    void changeState(State state);
}
```

Printer.java

```
class Printer implements IPrinter{
    private String mModel;
```

```
    private int mNumPapers;
    private double mPaintVolume;
    private double mPinchingProbability;
    private State mState;

    public Printer(String model, int numPapers, double paintVolume, double pinchingProbabi
lity) {
        this.setModel(model);
        this.setNumPapers(numPapers);
        this.setPaintVolume(paintVolume);
        this.setPinchingProbability(pinchingProbability);
        this.changeState(new WaitState(this));
    }

    @Override
    public void changeState(State state) {
        this.mState = state;
    }

    public State getSatate() {
        return this.mState;
    }

    public void setModel(String model) {
        this.mModel = model;
    }

    public String getModel() {
        return this.mModel;
    }

    public void setNumPapers(int numPapers) {
        this.mNumPapers = numPapers;
    }

    public int getNumPapers() {
        return this.mNumPapers;
    }

    public void setPaintVolume(double paintVolume) {
        this.mPaintVolume = paintVolume;
    }

    public double getPaintVolume() {
        return this.mPaintVolume;
    }

    public void setPinchingProbability(double pinchingProbability) {
        this.mPinchingProbability = pinchingProbability;
    }

    public double getPinchingProbability() {
        return this.mPinchingProbability;
    }
```

```java
    @Override
    public String print(int numPaper) {
        String resultOfPrint = this.mState.onPrint(numPaper);
        System.out.println(resultOfPrint);
        return resultOfPrint;
    }

    @Override
    public void loadPaper(int numPaper) {
        System.out.println(this.mState.loadPaper(numPaper));
    }

    @Override
    public void extractClampingPaper() {
        System.out.println(this.mState.onExtractClampingPaper());
    }

    @Override
    public void refilleCartridge() {
        System.out.println(this.mState.onRefilleCartridge());
    }

}
```

Рисунки с результатами работы программы

```
WaitState --> PrintState.

Paper cann't be printed. It's claimed.
PrintState --> ClampingState.

Paper cann't be printed.
Clamping State.

Paper cann't be printed.
Clamping State.

Paper was extracted.
ClampingState --> WaitState.

WaitState --> PrintState.

Paper was printed.
PrintState --> WaitState.

WaitState --> PrintState.

Paper cann't be printed. It's claimed.
PrintState --> ClampingState.

Paper cann't be printed.
Clamping State.

Paper cann't be printed.
Clamping State.
```

```
Paper was extracted.
ClampingState --> WaitState.

WaitState --> PrintState.

Paper was printed.
PrintState --> WaitState.

WaitState --> PrintState.

Paper was printed.
PrintState --> WaitState.

WaitState --> PrintState.

Paper was printed.
PrintState --> WaitState.

WaitState --> PrintState.

Other paper cann't be printed. Not enough painter in cartridge.
PrintState --> FailureState.

Other paper cann't be printed. Not enough painter in cartridge.
FailureState.

Other paper cann't be printed. Not enough painter in cartridge.
FailureState.
```

Вывод: приобрела навыки применения паттернов проектирования при решении практических задач с использованием языка Java.