



МИНОБРНАУКИ РОССИИ

*Федеральное государственное бюджетное образовательное учреждение
высшего образования*

«МИРЭА – Российский технологический университет»

РТУ МИРЭА

**Институт информационных технологий (ИТ)
Кафедра инструментального и прикладного программного обеспечения (ИиППО)**

Дисциплина «Программирование на языке Джава»

**ОТЧЕТ
ПО ПРАКТИЧЕСКОМУ ЗАНЯТИЮ №10**

Выполнил студент группы ИНБО-02-20

Лукияненко Д.В.

Принял

Степанов П.В.

Практическая работа выполнена «__» _____ 2021 г.

«_____»
Отметка о выполнении

«__» _____ 2021 г.

Москва – 2021 г.

СОДЕРЖАНИЕ

Цель работы	3
Задание	3
Репозиторий	4
Выполнение работы	4
Код выполненной работы	7
Код упражнения №1	7
Код упражнения №2	8
Код упражнения №3*	11
Вывод.....	18

Цель работы

Цель данной практической работы – Научиться применять порождающие паттерны при разработке программ на Java.

Задание

Упражнение 1

Реализовать класс Абстрактная фабрика для комплексных чисел

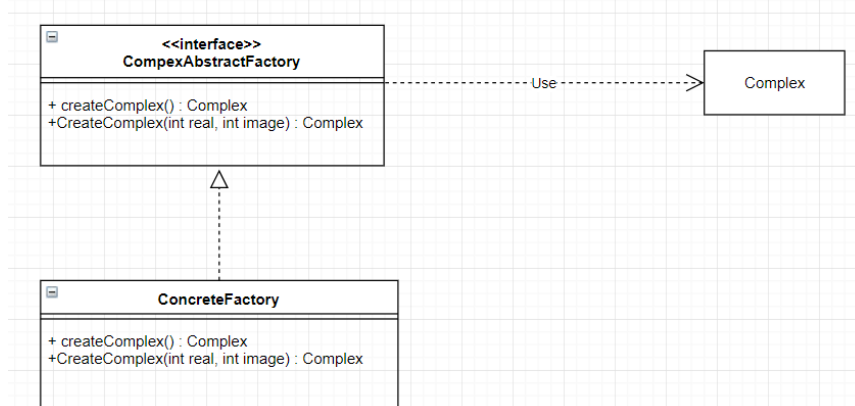


Рисунок 1 – Заданная UML Диаграмма

Упражнение 2

Реализовать класс Абстрактная фабрика для различных типов стульев: Викторианский стул, Многофункциональный стул, Магический стул, а также интерфейс Стул, от которого наследуются все классы стульев, и класс Клиент, который использует интерфейс стул в своем методе Sit (Chair chair).

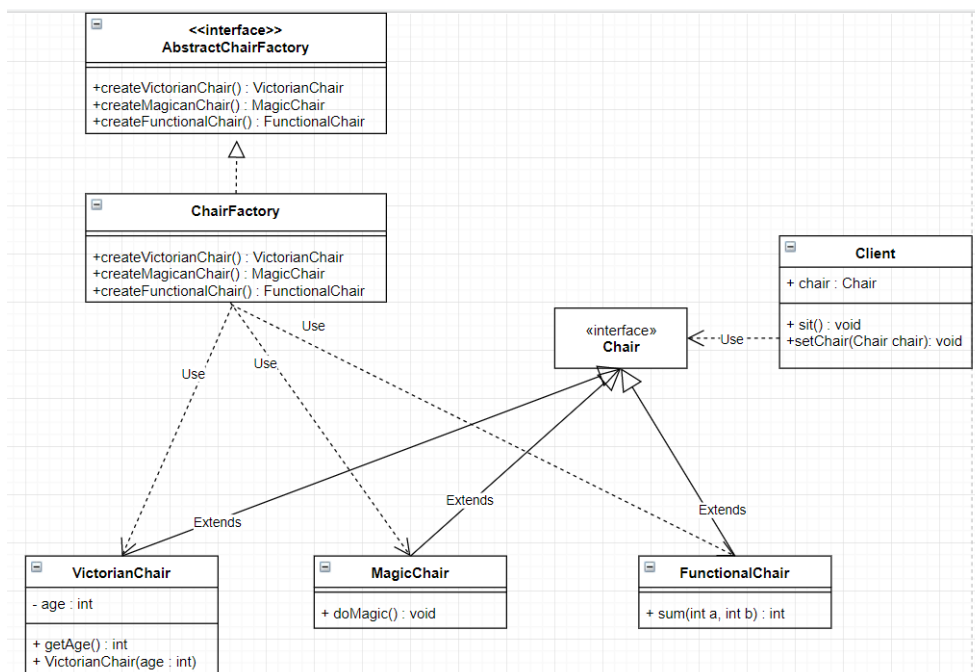


Рисунок 2 – Заданная UML Диаграмма

Упражнение 3*

Вводная. Компании нужно написать редактор текста, редактор изображений и редактор музыки. В трёх приложениях будет очень много общего: главное окно, панель инструментов, команды меню будут весьма схожими. Чтобы не писать повторяющуюся основу трижды, вам поручили разработать основу (каркас) приложения, которую можно использовать во всех трёх редакторах. На совещании в компании была принята следующая архитектура:

Исходные данные. Есть некий базовый интерфейс `IDocument`, представляющий документ неопределённого рода. От него впоследствии будут унаследованы конкретные документы: `TextDocument`, `ImageDocument`, `MusicDocument` и т.п. Интерфейс `IDocument` перечисляет общие свойства и операции для всех документов.

- При нажатии пунктов меню `File -> New` и `File -> Open` требуется создать новый экземпляр документа (конкретного подкласса). Однако каркас не должен быть привязан ни к какому конкретному виду документов.
- Нужно создать фабричный интерфейс `ICreateDocument`. Этот интерфейс содержит два абстрактных фабричных метода: `CreateNew` и `CreateOpen`, оба возвращают экземпляр `IDocument`
- Каркас оперирует одним экземпляром `IDocument` и одним экземпляром `ICreateDocument`. Какие конкретные классы будут подставлены сюда, определяется во время запуска приложения.

Репозиторий

Ссылка: https://github.com/neluckoff/mirea-java-lessons/tree/master/src/ru/luckoff/mirea/practice_10

Выполнение работы

Упражнение 1

В первом задании все просто. Первым делом я создал все классы, указанные в диаграмме, а затем заполнил их методами из тех же диаграмм.

Переопределение методов интерфейса проходят в классе `ConcreteFactory`.

В классе `Complex` мною были созданы два конструктора и метод `toString()`, в котором и выводится комплексное число, в моем случае формула.

Ниже будет представлена UML диаграмма первого упражнения.

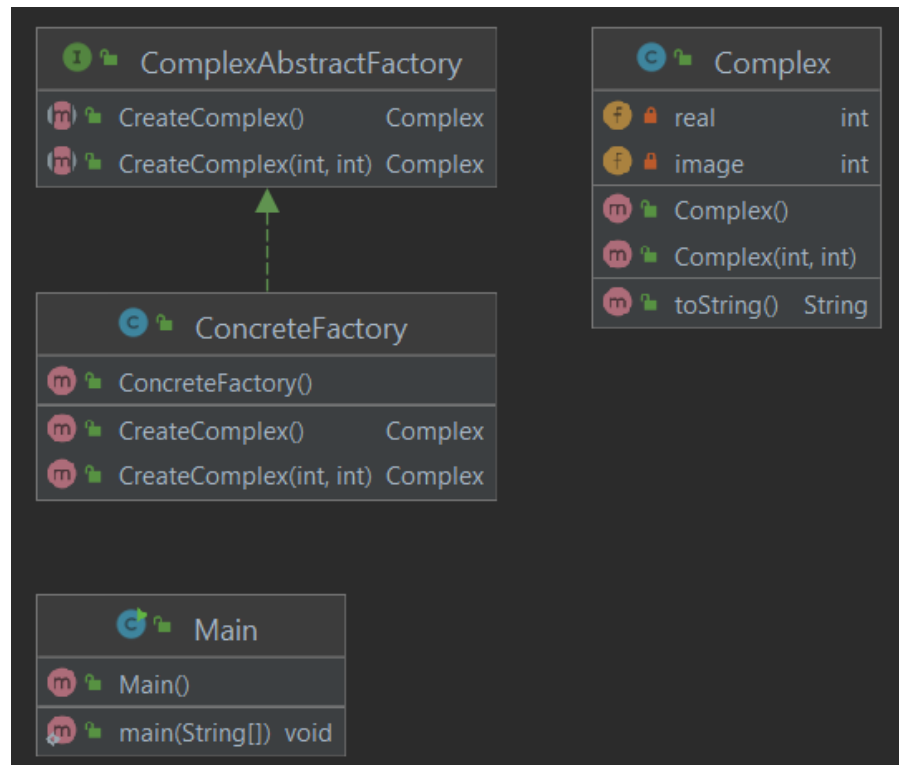


Рисунок 3 – UML Диаграмма для Упражнения 1

Упражнение 2

Во втором задании все также, как и в первом. Я просто переписал все классы из диаграммы второго упражнения и заполнил их методами из этой же диаграммы.

Как я реализовал эти методы можно будет посмотреть ниже, в коде программы.

На рисунке 4 я показал Вам полученную UML Диаграмму в результате выполнения данного упражнения.

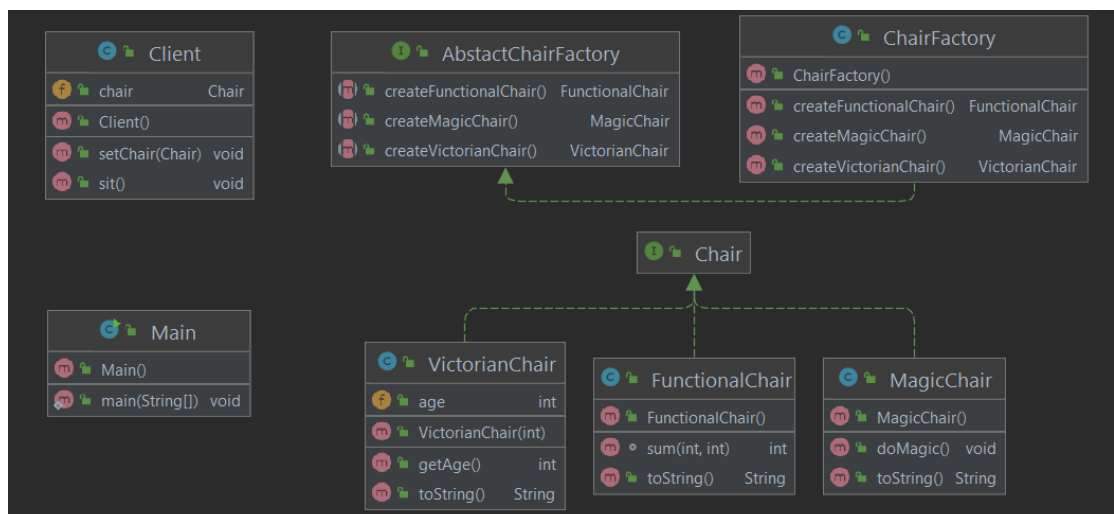


Рисунок 4 – UML Диаграмма для Упражнения 2

Упражнение 3*

В этом задании мною были созданы два интерфейса – IDocument и ICreateDocument. После этого я сделал три класса ImageDocument, TextDocument и MusicDocument, содержащие пустой конструктор и имплементирующие интерфейс IDocument.

А также еще три класса – CreateTextDocument, CreateMusicDocument и CreateImageDocument – имплементирующие интерфейс ICreateDocument.

Затем было самое интересное для меня – создание GUI. Здесь я не вижу смысла что либо описывать, ведь по этой теме у нас было отдельное задание.

Я решил добавить в своей программе вывод в консоль все создание и прочие логи. Я считаю, что в такой задаче, людям после меня будет легче доделывать Редактор с логами в консоле.

Ниже я представил полученную UML Диаграмму (рис. 5) и код программы.

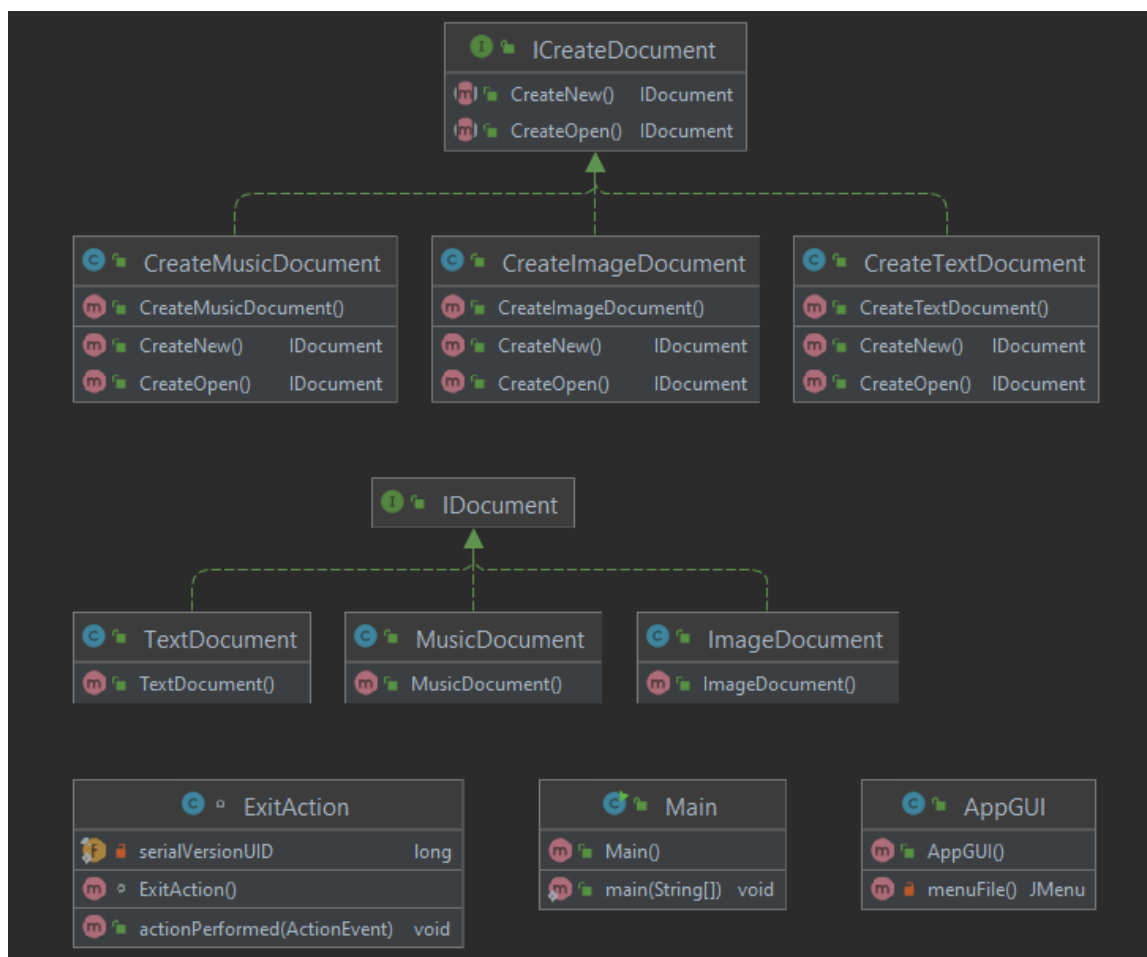


Рисунок 5 – UML Диаграмма для Упражнения 3

Код выполненной работы

Здесь в нескольких скриншотах можно увидеть как выглядит код выполненного задания и результат его работы.

Код упражнения №1

```
public interface ComplexAbstractFactory {  
    Complex CreateComplex();  
    Complex CreateComplex(int real, int image);  
}
```

Рисунок 6 – Класс ComplexAbstractFactory

```
public class ConcreteFactory implements ComplexAbstractFactory{  
  
    @Override  
    public Complex CreateComplex() {  
        return new Complex();  
    }  
  
    @Override  
    public Complex CreateComplex(int real, int image) { return new Complex(real, image); }  
}
```

Рисунок 7 – Класс ConcreteFactory

```
public class Complex {  
    private int real;  
    private int image;  
  
    public Complex(int real, int image) {  
        this.real = real;  
        this.image = image;  
    }  
  
    public Complex() {  
        this.real = 0;  
        this.image = 0;  
    }  
  
    @Override  
    public String toString() {  
        return "z = " + real + " + i*" + image;  
    }  
}
```

Рисунок 8 – Класс Complex

```
z = 0 + i*0

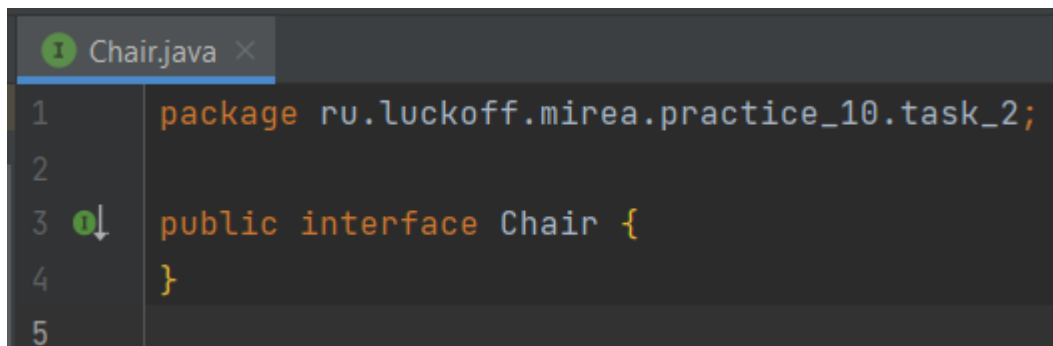
Choose the <real> and <image>:
12 92
z = 12 + i*92
```

Рисунок 9 – Вывод первого упражнения

Код упражнения №2

```
public interface AbstractChairFactory {
    VictorianChair createVictorianChair();
    MagicChair createMagicChair();
    FunctionalChair createFunctionalChair();
}
```

Рисунок 10 – Класс AbstractChairFactory



```
Chair.java x
1 package ru.luckoff.mirea.practice_10.task_2;
2
3 public interface Chair {
4 }
5
```

Рисунок 11 – Класс Chair

```
public class VictorianChair implements Chair {
    public int age;

    public VictorianChair(int age) { this.age = age; }

    public int getAge() { return age; }

    @Override
    public String toString() { return "Victorian Chair has been created. Selected age: " + age; }
}
```

Рисунок 12 – Класс VictorianChair


```

public class MagicChair implements Chair {

    public MagicChair() {

    }

    public void doMagic() { System.out.println("Yooo, magic is done, bro!"); }

    @Override
    public String toString() { return "Magic Chair has been created."; }

}

```

Рисунок 13 – Класс MagicChair

```

public class FunctionalChair implements Chair {
    public FunctionalChair() {

    }

    int sum(int a, int b) { return a + b; }

    @Override
    public String toString() { return "Functional Chair has been created." ; }

}

```

Рисунок 14 – Класс FunctionalChair

```

public class Client {
    public Chair chair;

    public void sit() {
        System.out.println("Sitting is very comfortable.");
    }

    public void setChair(Chair chair) { this.chair = chair; }

}

```

Рисунок 15 – Класс Client

```

public class ChairFactory implements AbstractChairFactory {
    @Override
    public VictorianChair createVictorianChair() {
        int age;
        Scanner in = new Scanner(System.in);
        System.out.println("Choose age: ");
        age = in.nextInt();
        return new VictorianChair(age);
    }

    @Override
    public MagicChair createMagicChair() { return new MagicChair(); }

    @Override
    public FunctionalChair createFunctionalChair() { return new FunctionalChair(); }
}

```

Рисунок 16 – Класс ChairFactory

```

public class Main {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        ChairFactory factory = new ChairFactory();
        Client client = new Client();
        System.out.println("Creating the first chair.");
        FunctionalChair chair1 = factory.createFunctionalChair();
        System.out.println(chair1.toString());
        System.out.println("Creating a second chair.");
        MagicChair chair2 = factory.createMagicChair();
        System.out.println(chair2.toString());
        System.out.println("Creating the third chair.");
        VictorianChair chair3 = factory.createVictorianChair();
        System.out.println(chair3.toString());

        System.out.println("\n");
        chair2.doMagic();
        System.out.println("Age: " + chair3.getAge());
        client.sit();
    }
}

```

Рисунок 17 – Класс Main, запуск программы

```

Creating the firs chair.
Functional Chair has been created.
Creating a second chair.
Magic Chair has been created.
Creating the third chair.
Choose age:
13
Victorian Chair has been created. Selected age: 13

Yooo, magic is done, bro!
Age: 13
Sitting is very comfortable.

Process finished with exit code 0

```

Рисунок 18 – Вывод второго упражнения

Код упражнения №3*

```

public interface ICreateDocument {
    abstract IDocument CreateNew();
    abstract IDocument CreateOpen();
}

```

Рисунок 19 – Интерфейс ICreateDocument

```

IDocument.java x
1 package ru.luckoff.mirea.practice_10.task_3;
2
3 public interface IDocument {
4     |
5 }

```

Рисунок 20 – Интерфейс IDocument

```

public class ImageDocument implements IDocument {
    public ImageDocument() {
    }
}

```

Рисунок 21 – Класс ImageDocument

```
public class MusicDocument implements IDocument {
    public MusicDocument() {
    }
}
```

Рисунок 22 – Класс MusicDocument

```
public class TextDocument implements IDocument {
    public TextDocument() {
    }
}
```

Рисунок 23 – Класс TextDocument

```
public class CreateImageDocument implements ICreateDocument {

    @Override
    public IDocument CreateNew() {
        System.out.println("Image Created");
        return new ImageDocument();
    }

    @Override
    public IDocument CreateOpen() {
        System.out.println("Image Opened");
        return new ImageDocument();
    }
}
```

Рисунок 24 – Класс CreateImageDocument

```
public class CreateMusicDocument implements ICreateDocument {  
    @Override  
    public IDocument CreateNew() {  
        System.out.println("Music Created");  
        return new MusicDocument();  
    }  
  
    @Override  
    public IDocument CreateOpen() {  
        System.out.println("Music Opened");  
        return new MusicDocument();  
    }  
}
```

Рисунок 24 – Класс CreateMusicDocument

```
public class CreateTextDocument implements ICreateDocument {  
    @Override  
    public IDocument CreateNew() {  
        System.out.println("Text Created");  
        return new TextDocument();  
    }  
  
    @Override  
    public IDocument CreateOpen() {  
        System.out.println("Text Opened");  
        return new TextDocument();  
    }  
}
```

Рисунок 25 – Класс CreateTextDocument

```

public class AppGUI extends JFrame {

    public AppGUI() {
        super( title: "Redactor");
        this.setBounds( x: 200, y: 200, width: 500, height: 500);
        Container pane = this.getContentPane();
        pane.setLayout(new GridLayout( rows: 3, cols: 3, hgap: 1, vgap: 1));

        JMenuBar menuBar = new JMenuBar();
        menuBar.add(menuFile());
        setJMenuBar(menuBar);
        setSize( width: 300, height: 200);
        setVisible(true);
    }
}

```

Рисунок 26 – Класс AppGUI и его конструктор

```

class ExitAction extends AbstractAction {
    private static final long serialVersionUID = 1L;
    ExitAction() { putValue(NAME, newValue: "Exit"); }
    public void actionPerformed(ActionEvent e) { System.exit( status: 0); }
}

```

Рисунок 27 – Класс ExitAction

```

public class Main {
    public static void main(String[] args) {
        AppGUI app = new AppGUI();
        app.setVisible(true);
    }
}

```

Рисунок 28 – Класс Main

```

private JMenu menuFile() {
    JMenu file = new JMenu( s: "File");
    JMenu newFile = new JMenu( s: "New");
    JMenuItem textDocumentNew = new JMenuItem( text: "Text Document");
    JMenuItem imageDocumentNew = new JMenuItem( text: "Image Document");
    JMenuItem musicDocumentNew = new JMenuItem( text: "Music Document");
    JMenu open = new JMenu( s: "Open");
    JMenuItem save = new JMenuItem( text: "Save");
    JMenuItem exit = new JMenuItem(new ExitAction());
    JMenuItem textDocumentOpen = new JMenuItem( text: "Text");
    JMenuItem imageDocumentOpen = new JMenuItem( text: "Image");
    JMenuItem musicDocumentOpen = new JMenuItem( text: "Music");

    file.add(newFile);
    newFile.add(textDocumentNew);
    newFile.add(imageDocumentNew);
    newFile.add(musicDocumentNew);
    file.add(open);
    open.add(textDocumentOpen);
    open.add(imageDocumentOpen);
    open.add(musicDocumentOpen);
    file.addSeparator();
    file.add(save);
    file.add(exit);

    ICreateDocument imageFac = new CreateImageDocument();
    ICreateDocument textFac = new CreateTextDocument();
    ICreateDocument musicFac = new CreateMusicDocument();
}

```

Рисунок 29 – Метод menuFile() класса AppGUI

```

imageDocumentNew.addActionListener(arg0 -> {
    imageFac.CreateNew();
});
textDocumentNew.addActionListener(arg0 -> {
    textFac.CreateNew();
});
musicDocumentNew.addActionListener(arg0 -> {
    musicFac.CreateNew();
});

imageDocumentOpen.addActionListener(arg0 -> {
    imageFac.CreateOpen();
});
textDocumentOpen.addActionListener(arg0 -> {
    textFac.CreateOpen();
});
musicDocumentOpen.addActionListener(arg0 -> {
    musicFac.CreateOpen();
});

save.addActionListener(e -> {JOptionPane.showMessageDialog( parentComponent: null,
    message: "Document has been saved.", title: "Saved", JOptionPane.INFORMATION_MESSAGE);
    System.out.println("Document has been saved.");
});

return file;
}

```

Рисунок 30 – Продолжение метода menuFile() класса AppGUI

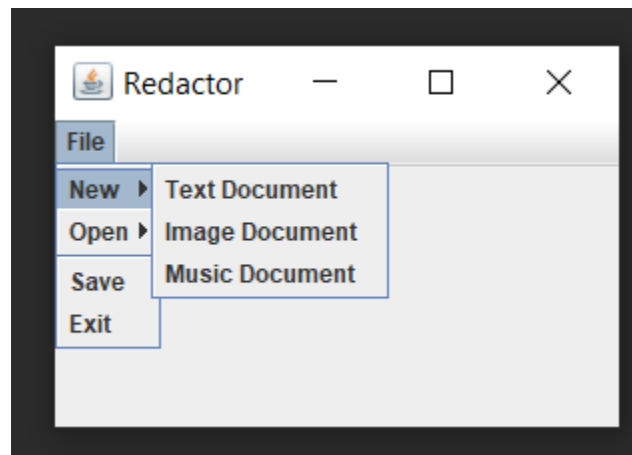


Рисунок 31 – Результат запуска

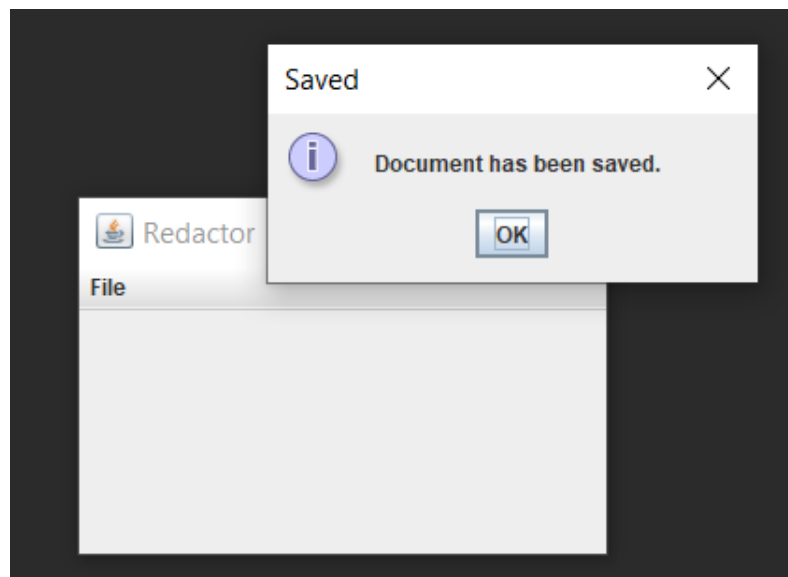


Рисунок 32 – Сохранение файла

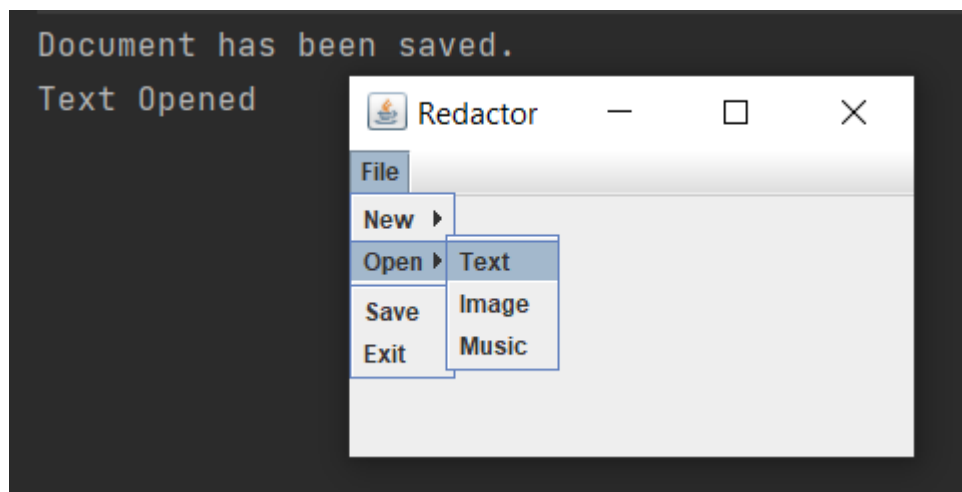


Рисунок 33 – Открытие файла и логи программы

Вывод

В результате выполнения данной практической работы я научился применять порождающие паттерны при разработке программ на Java. Изучил и выяснил такие паттерны, как абстрактная фабрика и фабричный метод.