



МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА - Российский технологический университет»

РТУ МИРЭА

Институт Информационных Технологий
Кафедра Вычислительной Техники (ВТ)

ОТЧЁТ ПО ПРАКТИЧЕСКИМ РАБОТАМ

по дисциплине

«Программирование на Java»

Выполнил студент группы
ИВБО-11-23

Туктаров Т.А

Принял преподаватель кафедры ВТ

Волков М.Ю

Лабораторная работа выполнена

« __ » _____ 2024 г.

«Зачтено»

« __ » _____ 2024 г.

Москва 2024

СОДЕРЖАНИЕ

ПРАКТИЧЕСКАЯ РАБОТА 1	7
Теоретическое введение	7
Постановка задачи.....	8
Ход выполнения работы	8
Программный код.....	8
Листинг 1.1 — реализация конвертора с определением правильного окончания.	8
Результат работы программы.....	9
Вывод.....	9
ПРАКТИЧЕСКАЯ РАБОТА 2	10
Теоретическое введение	10
Постановка задачи.....	11
Ход выполнения работы	12
Программный код.....	12
Результат работы программы.....	13
Вывод.....	13
ПРАКТИЧЕСКАЯ РАБОТА 3	14
Теоретическое введение	14
Постановка задачи.....	15
Ход выполнения работы	15
Программный код.....	15
Результат работы программы.....	17
Вывод.....	17
ПРАКТИЧЕСКАЯ РАБОТА 4	18

Теоретическое введение	18
Постановка задачи.....	19
Ход выполнения работы	20
Программный код.....	20
Результат работы программы.....	22
Вывод.....	22
ПРАКТИЧЕСКАЯ РАБОТА 5	23
Теоретическое введение	23
Постановка задачи.....	24
Ход выполнения работы	24
Программный код.....	25
Результат работы программы.....	27
Вывод.....	27
ПРАКТИЧЕСКАЯ РАБОТА 6	28
Теоретическое введение	28
Постановка задачи.....	29
Ход выполнения работы	30
Программный код.....	30
Результат работы программы.....	33
Вывод.....	34
ПРАКТИЧЕСКАЯ РАБОТА 7	35
Теоретическое введение	35
Постановка задачи.....	36
Результат работы программы.....	45
Вывод.....	46

ПРАКТИЧЕСКАЯ РАБОТА 8	47
Теоретическое введение	47
Постановка задачи.....	47
Ход выполнения работы	48
Программный код.....	48
Результат работы программы.....	51
Вывод.....	51
ПРАКТИЧЕСКАЯ РАБОТА 9	52
Теоретическое введение	52
Постановка задачи.....	52
Ход выполнения работы	53
Программный код.....	53
Результат работы программы.....	56
Вывод.....	56
ПРАКТИЧЕСКАЯ РАБОТА 10	57
Теоретическое введение	57
Постановка задачи.....	58
Ход выполнения работы	58
Программный код.....	58
Результат работы программы.....	60
Вывод.....	60
ПРАКТИЧЕСКАЯ РАБОТА 11	61
Теоретическое введение	61
Постановка задачи.....	61
Ход выполнения работы	62

Программный код.....	62
Результат работы программы.....	63
Вывод.....	63
ПРАКТИЧЕСКАЯ РАБОТА 12	64
Теоретическое введение	64
Постановка задачи.....	64
Ход выполнения работы	65
Программный код.....	65
Результат работы программы.....	66
Вывод.....	66
ПРАКТИЧЕСКАЯ РАБОТА 13	67
Теоретическое введение	67
Постановка задачи.....	68
Ход выполнения работы	68
Программный код.....	68
Результат работы программы.....	71
Вывод.....	71
ПРАКТИЧЕСКАЯ РАБОТА 14	72
Теоретическое введение	72
Постановка задачи.....	73
Ход выполнения работы	73
Программный код.....	73
Вывод.....	74
ПРАКТИЧЕСКАЯ РАБОТА 15	75
Теоретическое введение	75

Постановка задачи.....	75
Ход выполнения работы	76
Программный код.....	76
Результат работы программы.....	77
Вывод.....	77
ПРАКТИЧЕСКАЯ РАБОТА 16	78
Теоретическое введение	78
Постановка задачи.....	79
Ход выполнения работы	79
Программный код.....	79
Результат работы программы.....	82
Вывод.....	83
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	84

ПРАКТИЧЕСКАЯ РАБОТА 1

Теоретическое введение

Каждый тип данных имеет диапазон значений. Компилятор выделяет место в памяти для каждой переменной или константы в соответствии с ее типом данных. Java предоставляет восемь примитивных типов данных для числовых значений, символов и логических значений. В этом подразделе курса описываются числовые типы данных и операторы.

Для чисел с плавающей точкой в Java используется два типа данных: `float` и `double`. Диапазон значений типа данных `double` в два раза больше, чем `float`, поэтому значения типа `double` еще называются числами двойной точности, а `float` — одинарной точности.

К операторам числовых типов данных относятся стандартные арифметические операторы: сложения (+), вычитания (-), умножения (*), деления (/) и остатка от деления (%), перечисленные в следующей таблице. Операнды — это значения, обрабатываемые операторами.

Арифметические операторы +, -, *, / и % можно объединять с оператором присваивания для формирования составных операторов. Очень часто текущее значение переменной используется, потом изменяется, а затем опять присваивается этой же переменной. Например, следующее предложение увеличивает значение переменной `count` на 1: `count = count + 1`.

Числа с плавающей точкой можно преобразовать в целые числа с помощью явного приведения типа. Можно ли выполнять бинарные операции с двумя операндами разных типов? Да. Если целое число и число с плавающей точкой участвуют в бинарной операции, то в Java целое число автоматически преобразуется в число с плавающей точкой, поэтому `3 * 4.5` эквивалентно `3.0 * 4.5`.

Предложение внутри предложения `if` или `if-else` может быть любым допустимым Java-предложением, включая другое предложение `if` или `if-else`.

Внутреннее предложение if, как говорят, вложено во внешнее предложение if. В свою очередь, внутреннее предложение if может содержать другое предложение if. На самом деле нет предела глубины вложенности.

Постановка задачи

Задача 1: Напишите программу, которая конвертирует сумму денег из китайских юаней в российские рубли по курсу покупки 11.91.

Задача 2: Перепишите программу, которая конвертирует сумму денег из китайских юаней в российские рубли по курсу покупки 11.91, добавив структуру выбора для принятия решений об окончаниях входной валюты в зависимости от ее значения.

Ход выполнения работы

Напишем вначале код, который будет конвертировать из юаней в рубли по заданному курсу, а далее перепишем данный код так, чтобы он реализовывал определение правильного окончания для слов.

Программный код

Листинг 1.1 — реализация конвертора с определением правильного окончания.

```
import java.util.Scanner;
class CurrencyConverter {
    public static void main(String[] args) {
        final double ROUBLES_PER_YUAN = 11.91; // курс покупки
        int yuan; // сумма денег в юанях
        double roubles; // сумма денег в рублях
        String s = " ";
        Scanner input = new Scanner(System.in); // Получение суммы в
юанях
        System.out.print("Введите сумму денег в юанях: ");
        yuan = input.nextInt();
        if (yuan <= 0) {
            System.out.print("Некорректная сумма в юанях");
        }
        // Вычисление суммы в рублях
```

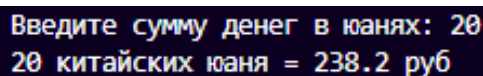


```

    roubles = ROUBLES_PER_YUAN * yuan;           // Отображение суммы в
рублях
    //System.out.println("Сумма в рублях: " + roubles);
    System.out.print(yuan);
    int digit = yuan % 10;
    int digit1 = yuan / 10 % 10;
    if (digit==1){
        s = " китайский юань";          }
    else if (digit > 4 || digit1==1){
        s = " китайских юаней";          }
    else{
        s = " китайских юаня";          }
    System.out.print(s + " = ");
    System.out.print(roubles);
    System.out.print(" руб");
    input.close();
}
}

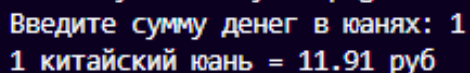
```

Результат работы программы



Введите сумму денег в юанях: 20
20 китайских юаня = 238.2 руб

Рисунок 1.1 Первый возможный результат выполнения.



Введите сумму денег в юанях: 1
1 китайский юань = 11.91 руб

Рисунок 1.2 Второй возможный результат выполнения.

Вывод

В результате выполнения данной практической работы был реализован обыкновенный конвертор валют, а далее был добавлен автоматический определитель правильного окончания.

ПРАКТИЧЕСКАЯ РАБОТА 2

Теоретическое введение

Объектно-ориентированное программирование по существу является технологией разработки многократно используемого программного обеспечения. Изучив материалы курсов «Основы Java-программирования», вы уже можете решить многие задачи программирования с помощью структур выбора, циклов, методов и массивов. Однако этих возможностей Java недостаточно для разработки крупномасштабных программных систем. Класс определяет свойства и поведение объектов.

Объектно-ориентированное программирование (ООП) включает в себя программирование с помощью объектов. Объект представляет сущность реального мира, которая может быть четко идентифицирована. Например, слушатель, стол, круг, кнопка и даже кредит могут рассматриваться как объекты. Объект имеет уникальный идентификатор, состояние и поведение.

Состояние объекта (также известное как его свойства или атрибуты) представлено полями данных с их текущими значениями. У объекта круга, например, есть поле данных `radius`, которое является свойством, характеризующим круг. У объекта прямоугольника, например, есть поля данных **`width`** и **`height`**, которые являются свойствами, характеризующими прямоугольник.

Поведение объекта (также известное как его действия) определяется методами. Чтобы вызвать метод объекта, необходимо попросить объект выполнить действие. Например, можно определить методы **`getArea()`** и **`getPerimeter()`** для объектов круга. У объекта круга можно вызвать метод **`getArea()`**, чтобы вернуть его площадь, и **`getPerimeter()`**, чтобы вернуть его периметр. Можно также определить метод **`setRadius(radius)`**. У объекта круга можно вызвать этот метод, чтобы изменить его радиус.

Конструктор — это особый вид метода. У него есть три особенности:

Конструктор должен иметь то же имя, что и класс.

Конструктор не имеет типа возвращаемого значения — даже типа `void`.

Конструктор вызывается при создании объекта с помощью оператора `new`, таким образом он играет роль инициализатора объектов.

Конструктор имеет точно такое же имя, как и определяющий его класс. Как и обычные методы, конструкторы могут быть перегружены (т.е. несколько конструкторов могут иметь одно и то же имя, но разные сигнатуры), что упрощает создание объектов с разными начальными значениями полей данных.

Класс можно определить и без конструкторов. В таком случае в классе неявно определяется общедоступный конструктор без аргументов с пустым телом. Этот конструктор, называемый заданным по умолчанию конструктором, предоставляется автоматически, только если в классе нет явно определенных конструкторов.

Постановка задачи

Задача 1: Напишите программу, в которой создается класс `Car`. В данном классе должны быть обозначены следующие поля: `String model`, `String license`, `String color`, `int year` – модель автомобиля, номер автомобиля, цвет автомобиля и год выпуска соответственно. Класс должен содержать три конструктора, один конструктор, который включает в себя все поля класса, один конструктор по умолчанию, один включает поля по выбору студента.

Задача 2: В отдельном классе `Main` создайте экземпляры классов (объекты), используя различные конструкторы, реализованные в задаче #1. Создайте в классе метод `To_String()`, который будет выводить значения полей экземпляров класса. Проверьте работу созданного метода, вызвав его у объекта. Дополните класс методами для получения и установки значений для всех полей (геттерами и сеттерами). Создайте метод класса, который будет возвращать возраст автомобиля, вычисляющийся от текущего года, значение текущего года допускается сделаться константным.

Ход выполнения работы

Для начала создадим car.java и определим там класс Car с заданными параметрами. Далее в отдельном классе Main создадим экземпляры класса Car, используя его конструкторы.

Программный код

Листинг 2.1 — Реализация класса Car в car.java.

```
public class Car{
    String model;
    String license;
    String color;
    int year;
    public Car(String _model, String _license, String _color, int _year){
        model = _model;
        license = _license;
        color = _color;
        year = _year;
    }
    public Car(){
        model = "";
        license = "";
        color = "";
        year = 0;
    }
    public Car(String model, int year){
        this.model = model;
        this.year = year;
        color = "grey";
        license = "default";
    }

    @Override
    public String toString(){
        return("model: " + model + "; year: " + year + "; color: " + color + ";
license: " + license);
    }

    public String getModel() {
        return model;
    }

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

    public String getLicense() {
        return license;
    }

    public void setLicense(String license) {
        this.license = license;
    }
}
```

```

    public String getColor() {
        return color;
    }

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

    public int getYear() {
        return year;
    }

    public void setYear(int year) {
        this.year = year;
    }

    public int getCarAge(){
        return 2024 - this.year;
    }
}

```

Листинг 2.2 — Реализация класса Main с экземплярами класса Car.

```

class Main{
    public class Main {
        public static void main(String[] args) {
            Car fiat = new Car("Fiat", "License", "yellow", 1970);
            Car ford = new Car("Fiat", 1984);
            Car def = new Car();
            System.out.println(fiat.toString());
        }
    }
}

```

Результат работы программы

```

model: Fiat; year: 1970; color: yellow; license: License

Process finished with exit code 0

```

Рисунок 2.1 Результат работы.

Вывод

Был реализован простой класс Car с различными конструкторами и геттерами с сеттарами. Также был реализован класс Main, который создавал экземпляры класса Car.

ПРАКТИЧЕСКАЯ РАБОТА 3

Теоретическое введение

Доступ к классу и его элементам можно задавать с помощью модификаторов доступа.

Чтобы ограничить доступ к классам, методам и полям данных, по умолчанию используется модификатор доступа `private`. Для доступа к методам обычно создаются геттеры и сеттеры, а при необходимости данные дублируются в новые переменные. В случае наследования для полей используются модификаторы `protected` или `default`, что обеспечивает доступ потомкам в пределах пакета. Модификатор `public` применяется реже и дает доступ к элементам класса из любых других классов.

В современной разработке принято следовать правилу: один каталог соответствует одному пакету, в котором может быть любое количество классов. Это упрощает организацию кода и делает проект более структурированным.

В дополнение к модификатору доступа `public` и заданному по умолчанию, Java предоставляет модификаторы `private` и `protected` для элементов класса.

Помимо модификаторов доступа `public`, `private` и `protected`, в Java существует четвертый модификатор — `default`, он применяется для элементов класса, к которым требуется доступ только внутри одного пакета. Этот модификатор важен для понимания более углубленных аспектов работы с доступом в Java.

Модификатор `private` делает методы и поля данных доступными только внутри класса, чьими элементами они являются. В следующей таблице показано, можно ли получить доступ к полю данных или методу класса `C1` с модификаторами `public`, `private` и заданному по умолчанию из класса `C2` этого же пакета и из класса `C3` другого пакета.

Модификатор доступа определяет, можно ли получить доступ к полям данных и методам класса за пределами этого класса. Не существует ограничений на доступ к полям данных и методам внутри класса.

Постановка задачи

Задача 1: Создайте пакет `vehicles`, который будет содержать классы `Car` и `ElectricCar` и пакет `app`, в котором будет находиться основной класс с методом `main`. Добавьте в класс `Car` приватные поля (`private`) `ownerName` и `insuranceNumber`. Создайте методы доступа (геттеры и сеттеры) для полей `ownerName` и `insuranceNumber`. Добавьте поле `engineType` с модификатором доступа `protected` и создайте методы доступа к этому полю.

Задача 2: Создайте новый класс `ElectricCar`, который наследует класс `Car`, и добавьте в него поле `batteryCapacity`. В классе `ElectricCar` используйте поле `engineType`, чтобы задать тип двигателя как `"Electric"`. Проверьте работу инкапсуляции и наследования, создав объекты классов `Car` и `ElectricCar` и продемонстрируйте доступ к полям с разными модификаторами.

Ход выполнения работы

Для выполнения этой задачи, мы создадим два пакета: ``vehicles`` и ``app``. В пакете ``vehicles`` будут классы ``Car`` и ``ElectricCar``, а в пакете ``app`` будет находиться основной класс с методом ``main``.

Программный код

Листинг 3.1 — Реализация класса `Main` в `app/Main.java`.

```
package app;

import vehicles.*;

public class Main {
    public static void main(String[] args) {
        Car c = new Car();
        ElectricCar e = new ElectricCar();
        c.setColor("Green");
    }
}
```

```

        e.setYear(2020);
        System.out.println(c.toString());
        System.out.println(e.toString());
    }
}

```

Листинг 3.2 — Реализация класса Car в vehicle/Car.java.

```

package vehicels;

public class Car extends Vehicle {
    public Car(){
        this.ownerName = "Tim";
        this.insuranceNumber = 88;
        this.engineType = "diesel";
    }

    public Car(String ownerName, int insuranceNumber, String engineType) {
        super(ownerName, insuranceNumber, engineType);
    }

    public String getownerName() {
        return ownerName;
    }

    public void setownerName(String ownerName) {
        ownerName = ownerName;
    }

    @Override
    public String vehicleType() {
        return "Car";
    }

    @Override
    public String toString(){
        return("model: " + getModel() + "; year: " + getYear() + "; color: " +
        getColor() + "; license: " + getLicense());
    }

    public String getEngineType() {
        return engineType;
    }

    public void setEngineType(String engineType) {
        this.engineType = engineType;
    }

    public int getInsuranceNumber() {
        return insuranceNumber;
    }

    public void setInsuranceNumber(int insuranceNumber) {
        this.insuranceNumber = insuranceNumber;
    }
}

```

Листинг 3.3 — Реализация класса ElectricCar в vehicle/ ElectricCar.java.

```

package vehicels;

public class ElectricCar extends Car{
    double batteryCapacity;
    public ElectricCar(){
        engineType = "Electric";
    }
}

```



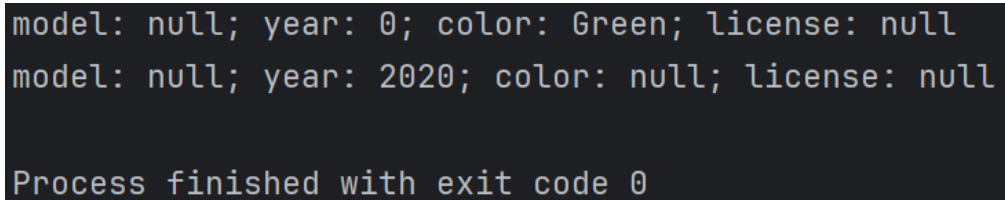
```
}

public ElectricCar(double batteryCapacity) {
    this.batteryCapacity = batteryCapacity;
}

public ElectricCar(String ownerName, int insuranceNumber, String engineType,
double batteryCapacity) {
    super(ownerName, insuranceNumber, engineType);
    this.batteryCapacity = batteryCapacity;
}

@Override
public String vehicleType() {
    return "electric Car";
}
}
```

Результат работы программы



```
model: null; year: 0; color: Green; license: null
model: null; year: 2020; color: null; license: null

Process finished with exit code 0
```

Рисунок 3.1 Результат работы.

Вывод

Был реализован класс Car и ElectricCar в пакете vehicles, а также основной класс Main в пакете app.

ПРАКТИЧЕСКАЯ РАБОТА 4

Теоретическое введение

Доступ к классу и его элементам можно задавать с помощью модификаторов доступа.

Чтобы ограничить доступ к классам, методам и полям данных, по умолчанию используется модификатор доступа `private`. Для доступа к методам обычно создаются геттеры и сеттеры, а при необходимости данные дублируются в новые переменные. В случае наследования для полей используются модификаторы `protected` или `default`, что обеспечивает доступ потомкам в пределах пакета. Модификатор `public` применяется реже и дает доступ к элементам класса из любых других классов.

В современной разработке принято следовать правилу: один каталог соответствует одному пакету, в котором может быть любое количество классов. Это упрощает организацию кода и делает проект более структурированным.

В дополнение к модификатору доступа `public` и заданному по умолчанию, Java предоставляет модификаторы `private` и `protected` для элементов класса.

Помимо модификаторов доступа `public`, `private` и `protected`, в Java существует четвертый модификатор — `default`, он применяется для элементов класса, к которым требуется доступ только внутри одного пакета. Этот модификатор важен для понимания более углубленных аспектов работы с доступом в Java.

Модификатор `private` делает методы и поля данных доступными только внутри класса, чьими элементами они являются. В следующей таблице показано, можно ли получить доступ к полю данных или методу класса `C1` с модификаторами `public`, `private` и заданному по умолчанию из класса `C2` этого же пакета и из класса `C3` другого пакета.

Модификатор доступа определяет, можно ли получить доступ к полям данных и методам класса за пределами этого класса. Не существует ограничений на доступ к полям данных и методам внутри класса.

Постановка задачи

Задача: Используя программу, выполненную во 2 и 3 практических работах, внести следующие изменения:

Добавить абстрактный класс `Vehicle`, который будет представлять общие свойства всех транспортных средств. В этот класс включите следующие общие поля для транспортных средств: `model` (модель); `license` (номерной знак); `color` (цвет); `year` (год выпуска); `ownerName` (имя владельца); `insuranceNumber` (страховой номер); `engineType` (тип двигателя, поле должно быть защищённым для наследования). Определите абстрактный метод `vehicleType()`, который будет возвращать тип транспортного средства. Добавьте методы для получения и изменения значений полей (геттеры и сеттеры).

Изменить класс `Car`, чтобы он наследовал `Vehicle`. Реализуйте абстрактный метод `vehicleType()`, чтобы он возвращал `"Car"`. В конструкторе класса `Car` используйте поля и методы родительского класса.

Изменить класс `ElectricCar`, чтобы он наследовал `Car`. Добавьте в класс поле `batteryCapacity` (ёмкость аккумулятора) и методы для работы с ним. Реализуйте метод `vehicleType()`, который будет возвращать `"Electric Car"`. Используйте `protected`-поле `engineType` для установки значения `"Electric"` в классе `ElectricCar`.

Использовать полиморфизм в тестовом классе для работы с объектами `Car` и `ElectricCar` через ссылки на родительские классы. Создайте объекты `Car` и `ElectricCar`, измените их свойства с помощью сеттеров, и выведите информацию на экран с помощью метода `toString()`.

Включить инкапсуляцию: убедитесь, что поля каждого класса имеют доступ через методы (геттеры и сеттеры), а не напрямую.

Ход выполнения работы

Для реализации данной задачи, мы создадим два пакета: `vehicles` для классов `Vehicle`, `Car` и `ElectricCar`, и `app` для тестового класса `TestCar`.

Программный код

Листинг 4.1 — Реализация класса TestCar в app/TestCar.java.

```
package app;
import vehicles.Car;
import vehicles.ElectricCar;
import vehicles.Vehicle;
public class TestCar {
    public static void main(String[] args) {
        Vehicle car1 = new Car("Toyota", "A123BC", "Red", 2015, "Максим Ш",
"20840XD", "Diesel");
        car1.setInsuranceNumber("1039XD");
        car1.setColor("Blue");
        System.out.println("Машина 1: [" + car1 + "]");

        Vehicle car2 = new ElectricCar("Tesla", "B456CD", "Green", 2020,
"Дмитрий Н", "94C0003", 75);
        car2.setLicense("Ab0ba");
        car2.setOwnerName("Денис Г.");
        System.out.println("Машина 2: [" + car2 + "]");
    }
}
```

Листинг 4.2 — Реализация класса Car в vehicle/Car.java.

```
package vehicles;

public class Car extends Vehicle{
    public Car(String model, String license, String color, int year, String
ownerName, String insuranceNumber, String engineType){
        setModel(model);
        setColor(color);
        setLicense(license);
        setYear(year);
        setOwnerName(ownerName);
        setInsuranceNumber(insuranceNumber);
        this.engineType = engineType;
    }
    //
    public String toString(){
        return "Model: " + getModel() + "; license: " + getLicense() + "; Color:
" + getColor() + "; year: "+getYear()+" ; ownerName: "+getOwnerName()+" ;
insuranceNumber: "+getInsuranceNumber()+" ; engineType: "+getEngineType();
    }
    @Override
    public String vehicleType(){
        return "Car";
    }
}
```

Листинг 4.3 — Реализация класса `ElectricCar` в `vehicle/ ElectricCar.java`.

```
package vehicles;
public class ElectricCar extends Car{
    private int batteryCapacity;
    public ElectricCar(String model, String license, String color, int year,
String ownerName, String insuranceNumber, int batteryCapacity) {
        super(model, license, color, year, ownerName,
insuranceNumber, "Electric");
        this.batteryCapacity = batteryCapacity;;
    }
    public int BCGet() {
        return batteryCapacity;
    }

    public void BCSet(int batteryCapacity) {
        this.batteryCapacity = batteryCapacity;
    }
    public String toString() {
        return super.toString() + "; Battery Capacity: " + batteryCapacity;
    }
    @Override
    public String vehicleType(){
        return "Electric Car";
    }
}
```

Листинг 4.4 — Реализация класса `Vehicle` в `vehicle/ Vehicle.java`.

```
package vehicles;

public abstract class Vehicle {
    private String model;
    private String license;
    private String color;
    private int year;

    private String ownerName;
    private String insuranceNumber;

    protected String engineType;

    public abstract String vehicleType();

    public String getModel() {
        return model;
    }

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

    public String getLicense() {
        return license;
    }

    public void setLicense(String license) {
        this.license = license;
    }

    public String getColor() {
```

```

        return color;
    }

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

    public int getYear() {
        return year;
    }

    public void setYear(int year) {
        this.year = year;
    }

    public String getOwnerName() {
        return ownerName;
    }

    public void setOwnerName(String ownerName) {
        this.ownerName = ownerName;
    }

    public String getInsuranceNumber() {
        return insuranceNumber;
    }

    public void setInsuranceNumber(String insuranceNumber) {
        this.insuranceNumber = insuranceNumber;
    }

    public String getEngineType() {
        return engineType;
    }
}

```

Результат работы программы

```

Машина 1: [Model: Toyota; license: A123BC; Color: Blue; year: 2015; ownerName: Максим Ш; insuranceNumber: 1039XD; engineType: Diesel]
Машина 2: [Model: Tesla; license: Ab0ba; Color: Green; year: 2020; ownerName: Денис Г.; insuranceNumber: 94C0003; engineType: Electric; Battery Capacity: 75]

```

Рисунок 4.1 Результат работы.

Вывод

Был реализован класс Car, ElectricCar и Vehicle в пакете vehicles, а также основной класс TestCar в пакете app.

ПРАКТИЧЕСКАЯ РАБОТА 5

Теоретическое введение

Простой пример интерфейса из повседневной жизни — пульт от телевизора. Он связывает два объекта, человека и телевизор, и выполняет разные задачи: прибавить или убавить звук, переключить каналы, включить или выключить телевизор. Одной стороне (человеку) нужно обратиться к интерфейсу (нажать на кнопку пульта), чтобы вторая сторона выполнила действие. Например, чтобы телевизор переключил канал на следующий. При этом пользователю не обязательно знать устройство телевизора и то, как внутри него реализован процесс смены канала.

Все, к чему пользователь имеет доступ — это *интерфейс*. Главная задача — получить нужный результат. Какое это имеет отношение к программированию и Java? Прямое :) Создание интерфейса очень похоже на создание обычного класса, только вместо слова `class` мы указываем слово `interface`.

Хорошим примером интерфейса, является рулевое управление автомобиля - у автомобиля есть руль, педали и коробка передач. У абсолютного большинства автомобилей, эти элементы следуют одному соглашению о поведении. Например, если повернуть рулевое колесо против часовой стрелки, автомобиль совершит поворот влево, а не увеличит скорость, независимо от своей марки. Если вы умеете использовать эти элементы управления, вы без особых усилий сможете справиться с любым автомобилем, независимо от его модели, года выпуска, марки и типа двигателя. Более того, можно представить ситуацию, в которой совершенно другой вид транспорта (к примеру, космический корабль) имеет тот же интерфейс управления, что и автомобили. Если вы, умея водить автомобиль, попадёте в кресло пилота такого корабля, то сможете не потеряться и в этой ситуации.

Интерфейс - это такое соглашение о наборе методов и их поведении, которое могут обязаться соблюдать совершенно не связанные классы, позволяя ссылаться на любой из них с помощью единой ссылки

Постановка задачи

Задача: Создайте абстрактный класс `Vehicle`, который будет представлять общие характеристики любого транспортного средства. Включите следующие поля: `model`, `license`, `color`, `year`, `ownerName`, `insuranceNumber`, `engineType`. Определите методы: геттеры и сеттеры для каждого поля, а также метод `toString()`, который возвращает строку с описанием транспортного средства. Добавьте абстрактный метод `vehicleType()`, который будет возвращать тип транспортного средства (например, `"Car"`, `"Electric Car"`). Класс `Car` должен наследовать абстрактный класс `Vehicle`. В конструкторе задавайте тип двигателя как `"Combustion"`. Реализуйте метод `vehicleType()`, который возвращает `"Car"`. Определите интерфейс `ElectricVehicle`, который будет описывать специфические методы для электромобилей. В интерфейсе должны быть следующие методы: `getBatteryCapacity()` и `setBatteryCapacity()`. Класс `ElectricCar` должен наследовать класс `Car` и реализовывать интерфейс `ElectricVehicle`. Реализуйте методы интерфейса и добавьте поле `batteryCapacity` для хранения информации о емкости батареи. В конструкторе задайте тип двигателя как `"Electric"`

Ход выполнения работы

Для реализации данной задачи мы создадим два пакета: ``vehicles`` для классов ``Vehicle``, ``Car``, ``ElectricCar`` и интерфейса ``ElectricVehicle``, и ``app`` для тестового класса ``TestCar``.

Программный код

Листинг 5.1 — тестирование работы программы.

```
package app;

import vehicles.*;
public class Main {
    public static void main(String[] args) {
        Car c = new Car();
        ElectricCar e = new ElectricCar();
        c.setYear(1980);
        e.setInsuranceNumber(101);
        e.setBatteryCapacity(1000);
        System.out.println(e.getBatteryCapacity());
        System.out.println(c);
        System.out.println(e);
    }
}
```

Листинг 5.2 — Реализация класса Car в vehicle/Car.java.

```
package vehicles;

public class Car extends Vehicle {
    public Car(){
        this.ownerName = "Tim";
        this.insuranceNumber = 88;
        this.engineType = "Combustion";
    }

    public Car(String ownerName, int insuranceNumber, String engineType) {
        super(ownerName, insuranceNumber, engineType);
    }

    @Override
    public String vehicleType() {
        return "Car";
    }
}
```

Листинг 5.3 — Реализация класса ElectricCar в vehicle/ ElectricCar.java.

```
package vehicles;

public class ElectricCar extends Car implements ElectricVehicle{
    private int batteryCapacity;

    public ElectricCar(){
        this.ownerName = "Tim";
        this.insuranceNumber = 57;
        this.engineType = "Electric";
    }

    @Override
    public int getBatteryCapacity() {
        return batteryCapacity;
    }

    @Override
    public void setBatteryCapacity(int capacity) {
        batteryCapacity = capacity;
    }
}
```

```
}
```

Листинг 5.4 — Реализация класса `Vehicle` в `vehicle/ Vehicle.java`.

```
package vehicles;

public abstract class Vehicle {
    private String model;
    private String license;
    private String color;
    private int year;
    protected String ownerName;
    protected int insuranceNumber;
    protected String engineType;

    public Vehicle() {}

    public Vehicle(String ownerName, int insuranceNumber, String engineType) {
        this.insuranceNumber = insuranceNumber;
        this.ownerName = ownerName;
        this.engineType = engineType;
    }

    public abstract String vehicleType();

    @Override
    public String toString() {
        return "Vehicle: " +
            "model='" + model + '\'' +
            ", license='" + license + '\'' +
            ", color='" + color + '\'' +
            ", year=" + year +
            ", ownerName='" + ownerName + '\'' +
            ", insuranceNumber=" + insuranceNumber +
            ", engineType='" + engineType + '\'' +
            '|';
    }

    public String getModel() {
        return model;
    }

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

    public String getEngineType() {
        return engineType;
    }

    public void setEngineType(String engineType) {
        this.engineType = engineType;
    }

    public int getInsuranceNumber() {
        return insuranceNumber;
    }

    public void setInsuranceNumber(int insuranceNumber) {
        this.insuranceNumber = insuranceNumber;
    }

    public String getOwnerName() {
        return ownerName;
    }
}
```

```

    }

    public void setOwnerName(String ownerName) {
        this.ownerName = ownerName;
    }

    public int getYear() {
        return year;
    }

    public void setYear(int year) {
        this.year = year;
    }

    public String getColor() {
        return color;
    }

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

    public String getLicense() {
        return license;
    }

    public void setLicense(String license) {
        this.license = license;
    }
}

```

Листинг 5.5 — Реализация интерфейса `ElectricVehicle` `vehicle/ ElectricVehicle.java`.

```

public interface ElectricVehicle {
    public int getBatteryCapacity();
    public void setBatteryCapacity(int capacity);
}

```

Результат работы программы

```

1000
Vehicle: model='null', license='null', color='null', year=1980, ownerName='Tim', insuranceNumber=88, engineType='Combustion'|
Vehicle: model='null', license='null', color='null', year=0, ownerName='Tim', insuranceNumber=101, engineType='Electric'|

```

Рисунок 5.1 Результат работы.

Вывод

Был реализован класс `Car`, `ElectricCar` и `Vehicle` в пакете `vehicles`, а также основной класс `TestCar` в пакете `app`. Также был реализован интерфейс `ElectricVehicle`.

ПРАКТИЧЕСКАЯ РАБОТА 6

Теоретическое введение

Простой пример интерфейса из повседневной жизни — пульт от телевизора. Он связывает два объекта, человека и телевизор, и выполняет разные задачи: прибавить или убавить звук, переключить каналы, включить или выключить телевизор. Одной стороне (человеку) нужно обратиться к интерфейсу (нажать на кнопку пульта), чтобы вторая сторона выполнила действие. Например, чтобы телевизор переключил канал на следующий. При этом пользователю не обязательно знать устройство телевизора и то, как внутри него реализован процесс смены канала.

Все, к чему пользователь имеет доступ — это *интерфейс*. Главная задача — получить нужный результат. Какое это имеет отношение к программированию и Java? Прямое :) Создание интерфейса очень похоже на создание обычного класса, только вместо слова `class` мы указываем слово `interface`.

Хорошим примером интерфейса, является рулевое управление автомобиля - у автомобиля есть руль, педали и коробка передач. У абсолютного большинства автомобилей, эти элементы следуют одному соглашению о поведении. Например, если повернуть рулевое колесо против часовой стрелки, автомобиль совершит поворот влево, а не увеличит скорость, независимо от своей марки. Если вы умеете использовать эти элементы управления, вы без особых усилий сможете справиться с любым автомобилем, независимо от его модели, года выпуска, марки и типа двигателя. Более того, можно представить ситуацию, в которой совершенно другой вид транспорта (к примеру, космический корабль) имеет тот же интерфейс управления, что и автомобили. Если вы, умея водить автомобиль, попадёте в кресло пилота такого корабля, то сможете не потеряться и в этой ситуации.

Интерфейс - это такое соглашение о наборе методов и их поведении, которое могут обязаться соблюдать совершенно не связанные классы, позволяя ссылаться на любой из них с помощью единой ссылки

Постановка задачи

Задача: 1. Создайте интерфейс в проекте велосипеда, который устанавливает название компании-производителя велосипедов как неизменяемое значение. Он также определяет методы, которые должны быть реализованы любым классом, использующим интерфейс.

2. Создайте интерфейс с именем MountainParts, который имеет константу с именем TERRAIN, которая будет хранить строковое значение «off_road». Интерфейс определит два метода, которые принимают строковое аргументное имя newValue, и два метода, которые будут возвращать текущее значение поля экземпляра. Методы должны быть названы: getSuspension, setSuspension, getType, setType.

3. Создайте интерфейс RoadParts, который имеет константу с именем terrain, которая будет хранить строковое значение «track_racing». Интерфейс определит два метода, которые принимают строковое аргументное имя newValue, и два метода, которые будут возвращать текущее значение поля экземпляра. Методы должны быть названы: getTyreWidth, setTyreWidth, getPostHeight, setPostHeight.

4. Используйте интерфейс BikeParts с классом Bike, добавляя любые необходимые нереализованные методы. Добавьте требуемый внутренний код для каждого из добавленных методов.

5. Используйте интерфейс MountainParts с классом MountainBike, добавив все необходимые нереализованные методы. Добавьте требуемый внутренний код для каждого из добавленных методов.

6. Используйте интерфейс `RoadParts` с классом `RoadBike`, добавив все необходимые нереализованные методы. Добавьте требуемый внутренний код для каждого из добавленных методов.

7. Запустите и протестируйте свою программу, она должна работать точно так же, как и раньше.

8. В нижней части класса драйвера обновите высоту столба для `bike1` до 20 вместо 22.

9. Выведите значения `bike1` на экран, чтобы подтвердить изменение.

10. Запустите и протестируйте свою программу.

Ход выполнения работы

Для выполнения этой задачи мы создадим серию интерфейсов и классов, которые будут моделировать велосипеды с различными характеристиками.

Программный код

Листинг 6.1 — Реализация класса `Bike` в `Bike.java`.

```
package bikeproject;

public class Bike implements BikeParts{

    private String handleBars, frame, tyres, seatType;
    private int NumGears;
    private final String make;

    public Bike(){
        this.make = "Oracle Cycles";
    } //end constructor

    public Bike(String handleBars, String frame, String tyres, String
seatType, int numGears) {
        this.handleBars = handleBars;
        this.frame = frame;
        this.tyres = tyres;
        this.seatType = seatType;
        NumGears = numGears;
        this.make = "Oracle Cycles";
    } //end constructor
    @Override
    public String getMake() {
        return this.make;
    }
}
```

```

        public void printDescription()
        {
            System.out.println("\n" + this.make + "\n"
                               + "This bike has " + this.handleBars + "
handlebars on a "
                               + this.frame + " frame with " + this.NumGears
+ " gears."
                               + "\nIt has a " + this.seatType + " seat with
" + this.tyres + " tyres.");
        } //end method printDescription
    } //end class Bike

```

Листинг 6.2 — Реализация класса `BikeDriver` в `BikeDriver.java`.

```

package bikeproject;

public class BikeDriver {

    public static void main(String[] args) {

        RoadBike bike1 = new RoadBike();
        RoadBike bike2 = new RoadBike("drop", "tourer", "semi-grip", "comfort",
14, 25, 18);
        MountainBike bike3 = new MountainBike();
        Bike bike4 = new Bike();

        bike1.printDescription();
        bike2.printDescription();
        bike3.printDescription();
        bike4.printDescription();

        bike1.setPostHeight(20);
        bike1.printDescription();
    } //end method main
} //end class BikeDriver

```

Листинг 6.3 — Реализация интерфейса `BikeParts` в `BikeParts.java`.

```

package bikeproject;

public interface BikeParts {
    String COMPANY = "Oracle Cycles";
    String getMake();
}

```

Листинг 6.4 — Реализация класса `MountainBike` в `MountainBike.java`.

```

package bikeproject;

public class MountainBike extends Bike implements MountainParts{

    private String suspension, type;
    private int frameSize;

    public MountainBike()
    {
        this("Bull Horn", "Hardtail", "Maxxis", "dropper", 27, "RockShox XC32",
"Pro", 19);
    } //end constructor

    public MountainBike(String handleBars, String frame, String tyres, String
seatType, int numGears,
                        String suspension, String type, int frameSize) {

```

```

        super(handleBars, frame, tyres, seatType, numGears);
        this.suspension = suspension;
        this.type = type;
        this.frameSize = frameSize;
    } //end constructor
    @Override
    public String getSuspension() {
        return suspension;
    }

    @Override
    public void setSuspension(String newValue) {
        this.suspension = newValue;
    }

    @Override
    public String getType() {
        return type;
    }

    @Override
    public void setType(String newValue) {
        this.type = newValue;
    }
    @Override
    public void printDescription()
    {
        super.printDescription();
        System.out.println("This mountain bike is a " + this.type + " bike and
has a " + this.suspension + " suspension and a frame size of " + this.frameSize
+ "inches.");
    }
    } //end method printDescription
} //end class MountainBike

```

Листинг 6.5 — Реализация класса RoadBike в RoadBike.java.

```

package bikeproject;

public class RoadBike extends Bike implements RoadParts{

    private int tyreWidth, postHeight;

    public RoadBike()
    {
        this("drop", "racing", "tread less", "razor", 19, 20, 22);
    } //end constructor

    public RoadBike(int postHeight)
    {
        this("drop", "racing", "tread less", "razor", 19, 20, postHeight);
    } //end constructor

    public RoadBike(String handleBars, String frame, String tyres, String
seatType, int numGears,
        int tyreWidth, int postHeight) {
        super(handleBars, frame, tyres, seatType, numGears);
        this.tyreWidth = tyreWidth;
        this.postHeight = postHeight;
    } //end constructor
    @Override
    public int getTyreWidth() {

```



```

        return tyreWidth;
    }

    @Override
    public void setTyreWidth(int newValue) {
        this.tyreWidth = newValue;
    }

    @Override
    public int getPostHeight() {
        return postHeight;
    }

    @Override
    public void setPostHeight(int newValue) {
        this.postHeight = newValue;
    }

    @Override
    public void printDescription()
    {
        super.printDescription();
        System.out.println("This Roadbike bike has " + this.tyreWidth + "mm
        tyres and a post height of " + this.postHeight + ".");
    } //end method printDescription
} //end class Road

```

Листинг 6.6 — Реализация интерфейса RoadParts в RoadParts.java.

```

package bikeproject;
public interface RoadParts {
    String terrain = "track_racing"; // константа для типа местности
    int getTyreWidth();
    void setTyreWidth(int newValue);

    int getPostHeight();
    void setPostHeight(int newValue);
}

```

Результат работы программы

```

Oracle Cycles
This bike has drop handlebars on a racing frame with 19 gears.
It has a razor seat with tread less tyres.
This Roadbike bike has 20mm tyres and a post height of 22.

Oracle Cycles
This bike has drop handlebars on a tourer frame with 14 gears.
It has a comfort seat with semi-grip tyres.
This Roadbike bike has 25mm tyres and a post height of 18.

Oracle Cycles
This bike has Bull Horn handlebars on a Hardtail frame with 27 gears.
It has a dropper seat with Maxxis tyres.
This mountain bike is a Pro bike and has a RockShox XC32 suspension and a frame size of 19inches.

Oracle Cycles
This bike has null handlebars on a null frame with 0 gears.
It has a null seat with null tyres.

Oracle Cycles
This bike has drop handlebars on a racing frame with 19 gears.
It has a razor seat with tread less tyres.
This Roadbike bike has 20mm tyres and a post height of 20.
PS C:\MyProfile\универ\github\Java-Univesity\практика 6>

```

Рисунок 6.1 Результат работы.

Вывод

Был модифицирован исходных код по заданию. Были добавлены интерфейсы BikeParts и MountainParts.

ПРАКТИЧЕСКАЯ РАБОТА 7

Теоретическое введение

Рассмотрим геометрические фигуры. Нам требуется создать классы для моделирования геометрических фигур, таких как круги и прямоугольники. У геометрических фигур есть много общих свойств и вариантов поведения. Они могут быть определенного цвета, закрашенными или незакрашенными. Таким образом, общий класс `GeometricObject` можно использовать для моделирования всех геометрических фигур. Этот класс содержит свойства `color` и `filled`, а также соответствующие им `getter`- и `setter`-методы. Этот класс также содержит свойство `dateCreated` и методы `getDateCreated()` и `toString()`. Метод `toString()` возвращает строковое представление объекта. Поскольку круг является конкретным типом геометрической фигуры, он имеет общие свойства и методы с другими геометрическими фигурами. Таким образом, имеет смысл определить класс `Circle`, который будет порожден от класса `GeometricObject`. Аналогично, класс `Rectangle` также можно определить как конкретный тип класса `GeometricObject`.

Ключевое слово `extends` сообщает компилятору, что класс `Circle` порождается от класса `GeometricObject`, наследуя таким образом методы `getColor()`, `setColor()`, `isFilled()`, `setFilled()` и `toString()`. Перегруженный конструктор `Circle` реализуется путем вызова методов `setColor()` и `setFilled()` для присваивания свойств `color` и `filled`. `Public`-методы, определенные в суперклассе `GeometricObject`, наследуются подклассом `Circle`, поэтому их можно использовать в `Circle`.

Обработка исключений в Java — один из мощных механизмов обработки ошибок времени выполнения, позволяющий поддерживать нормальный поток работы приложения. В Java исключение — это событие, которое нарушает нормальный ход программы. Это объект, который выбрасывается во время выполнения.

Java предоставляет пять ключевых слов, которые используются для обработки исключений.

Постановка задачи

Задача 1: Создайте класс `Triangle` для представления треугольников, который порождается от класса `GeometricObject`. Напишите клиент этих классов — программу, которая запрашивает у пользователя ввести три стороны треугольника, цвет и логическое значение для указания закрашен ли треугольник. Программа должна создавать объект типа `Triangle` с указанными сторонами и присваивать значения свойствам цвет `color` и заливка `isFilled` с помощью этих входных данных. Программа должна отображать площадь `area`, периметр `perimeter`, цвет, а также `true` или `false` для указания, закрашен треугольник или нет.

Задача 2: В треугольнике сумма длин любых двух сторон больше длины третьей стороны. Класс `Triangle` должен удовлетворять этому правилу. Создайте класс `IllegalTriangleException` и измените конструктор класса `Triangle`, чтобы выбросить объект типа `IllegalTriangleException`, если треугольник создан со сторонами, нарушающими это правило.

Задача 3: Спроектируйте новый класс `Triangle`, который наследуется от абстрактного класса `GeometricObject`.

- 1. Напишите тестовую программу, которая запрашивает у пользователя ввод трёх сторон треугольника, цвета и логического значения для указания заливки.
- 2. Программа должна создать объект типа `Triangle` с этими сторонами и задать свойства `color` и `filled`, используя введенные пользователем данные.
- 3. Программа должна отображать площадь, периметр, цвет и значение `true` или `false` для указания заливки.

- 4. Реализуйте в классе `GeometricObject` интерфейс `Comparable` и определите статический метод `max()` в классе `GeometricObject` для поиска наибольшего из двух объектов типа `GeometricObject`.
- 5. Проверьте тестовой программой работу метода `max()` для поиска наибольшего из двух кругов и наибольшего из двух прямоугольников.
- 6. Вернитесь к классу с именем `ComparableCircle`, который наследуется от `Circle` и реализует `Comparable`. Напишите тестовую программу, чтобы найти наибольший из двух экземпляров класса `ComparableCircle` и наибольший между кругом и прямоугольником, используя метод `compareTo()`.

Задача 4:

- 1. Создайте интерфейс с именем `Colorable` с помощью метода `howToColor()` типа `void`. Каждый класс раскрашиваемого объекта должен реализовывать интерфейс `Colorable`.
- 2. Спроектируйте класс `Square`, который наследуется от `GeometricObject` и реализует `Colorable`. Реализуйте метод `howToColor()` для отображения сообщения: Раскрасьте все четыре стороны.
- 3. Класс `Square` содержит поле данных `side` с `getter`- и `setter`-методами, а также конструктор для создания `Square` с указанной стороной. У класса `Square` есть скрытое поле данных типа `double` с именем `side` и `getter`- и `setter`-методами. У него есть безаргументный конструктор, который создает объект типа `Square` со стороной, равной 0, и еще один конструктор, который создает объект типа `Square` с указанной стороной.
- 4. Напишите тестовую программу, которая создает массив из пяти объектов типа `GeometricObjects`. Для каждого объекта в массиве отобразите его площадь и вызовите метод `howToColor()`, если его можно раскрасить.

Ход выполнения работы

Создание геометрических объектов.

Программный код

Листинг 7.1 — Реализация класса Circle в Circle.java.

```
public class Circle extends GeometricObject implements Colorable{
    private double radius;

    /** Создает по умолчанию заданный круг */
    public Circle() {
    }

    /** Создает круг с указанным радиусом */
    public Circle(double radius) {
        this.radius = radius;
    }

    /** Создает круг с указанным радиусом, цветом и заливкой */
    public Circle(double radius, String color, boolean filled) {
        this.radius = radius;
        setColor(color);
        setFilled(filled);
    }

    /** Возвращает радиус */
    public double getRadius() {
        return radius;
    }

    /** Присваивает новый радиус */
    public void setRadius(double radius) {
        this.radius = radius;
    }

    /** Возвращает площадь */
    public double getArea() {
        return radius * radius * Math.PI;
    }

    /** Возвращает диаметр */
    public double getDiameter() {
        return 2 * radius;
    }

    /** Возвращает периметр */
    public double getPerimeter() {
        return 2 * radius * Math.PI;
    }

    /** Отображает информацию о круге */
    public void printCircle() {
        System.out.println("Круг создан " + getDateCreated() +
            " и радиус равен " + radius);
    }

    @Override
```

```

    public void howToColor(){
        System.out.println("Раскрасьте круг");
    }
}

```

Листинг 7.2 — Реализация класса ComparableCircle в ComparableCircle.java.

```

public class ComparableCircle extends Circle implements
Comparable<GeometricObject> {
    public ComparableCircle(double radius) {
        super(radius);
    }

    @Override
    public int compareTo(GeometricObject o) {
        return Double.compare(this.getArea(), o.getArea());
    }
}

```

Листинг 7.3 — Реализация класса GeometricObject в GeometricObject.java.

```

public abstract class GeometricObject implements Comparable<GeometricObject>{
    private String color = "белый";
    private boolean filled;
    private java.util.Date dateCreated;

    /** Создает по умолчанию заданный геометрический объект */
    public GeometricObject() {
        dateCreated = new java.util.Date();
    }

    /** Создает геометрический объект с указанным цветом и заливкой */
    public GeometricObject(String color, boolean filled) {
        dateCreated = new java.util.Date();
        this.color = color;
        this.filled = filled;
    }

    /** Возвращает цвет */
    public String getColor() {
        return color;
    }

    /** Присваивает новый цвет */
    public void setColor(String color) {
        this.color = color;
    }

    /** Возвращает заливку. Поскольку filled типа boolean,
     *  * getter-метод называется isFilled */
    public boolean isFilled() {
        return filled;
    }

    /** Присваивает новую заливку */
    public void setFilled(boolean filled) {
        this.filled = filled;
    }

    /** Получает dateCreated */
    public java.util.Date getDateCreated() {
        return dateCreated;
    }
}

```

```

public abstract double getArea();

@Override
public int compareTo(GeometricObject o) {
    return Double.compare(this.getArea(), o.getArea());
}

public static GeometricObject max(GeometricObject o1, GeometricObject o2) {
    return o1.compareTo(o2) > 0 ? o1 : o2;
}

/** Возвращает строковое представление этого объекта */
public String toString() {
    return "создан " + dateCreated + ",\nцвет: " + color +
        "\n, заливка: " + filled;
}
}

```

Листинг 7.4 — Реализация класса `IllegalTriangleException` в `IllegalTriangleException.java`.

```

public class IllegalTriangleException extends Exception {
    public IllegalTriangleException(String message) {
        super(message);
    }
}

```

Листинг 7.5 — Реализация класса `Rectangle` в `Rectangle.java`.

```

public class Rectangle extends GeometricObject {
    private double width;
    private double height;

    /** Создает по умолчанию заданный прямоугольник */
    public Rectangle() {
    }

    /** Создает прямоугольник с указанной шириной и высотой */
    public Rectangle(double width, double height) {
        this.width = width;
        this.height = height;
    }

    /** Создает прямоугольник с указанной шириной, высотой, цветом и заливкой */
    public Rectangle(double width, double height, String color, boolean filled)
    {
        this.width = width;
        this.height = height;
        setColor(color);
        setFilled(filled);
    }

    /** Возвращает ширину */
    public double getWidth() {
        return width;
    }

    /** Присваивает новую ширину */
    public void setWidth(double width) {
        this.width = width;
    }

    /** Возвращает высоту */
    public double getHeight() {
        return height;
    }
}

```



```

    }

    /** Присваивает новую высоту */
    public void setHeight(double height) {
        this.height = height;
    }

    /** Возвращает площадь */
    public double getArea() {
        return width * height;
    }

    /** Возвращает периметр */
    public double getPerimeter() {
        return 2 * (width + height);
    }
}

```

Листинг 7.6 — Реализация класса Square в Square.java.

```

public class Square extends GeometricObject implements Colorable {
    private double side;

    public Square() {
        this.side = 0;
    }

    public Square(double side) {
        this.side = side;
    }

    public double getSide() {
        return side;
    }

    public void setSide(double side) {
        this.side = side;
    }

    @Override
    public double getArea() {
        return side * side;
    }

    public double getPerimeter() {
        return 4 * side;
    }

    @Override
    public void howToColor() {
        System.out.println("Раскрасьте все четыре стороны.");
    }
}

```

Листинг 7.6 — Реализация класса Triangle в Triangle.java.

```

import java.awt.*;

public class Triangle extends GeometricObject implements Colorable {
    private double side1 = 1.0;
    private double side2 = 1.0;
    private double side3 = 1.0;

    public Triangle() {

```

```

    }

    public Triangle(double side1, double side2, double side3) throws
    IllegalArgumentException {
        this.side1 = side1;
        this.side2 = side2;
        this.side3 = side3;
        if (side1 + side2 <= side3 || side1 + side3 <= side2 || side2 + side3 <=
side1) {
            throw new IllegalArgumentException("Стороны " + side1 + ", " + side2
+ ", и " + side3 + " не образуют треугольник.");
        }
    }

    public double getSide1() {
        return side1;
    }
    public double getSide2() {
        return side2;
    }
    public double getSide3() {
        return side3;
    }
    @Override
    public double getArea() {
        double s = getPerimeter() / 2;
        return Math.sqrt(s * (s - side1) * (s - side2) * (s - side3));
    }
    public double getPerimeter() {
        return side1 + side2 + side3;
    }

    @Override
    public String toString() {
        return "Треугольник: сторона1 = " + side1 + ", сторона2 = " + side2 + ",
сторона3 = " + side3;
    }

    @Override
    public void howToColor(){
        System.out.println("Раскрасьте 3 стороны");
    }
}

```

Листинг 7.7 — Реализация класса TestCircleRectangle в TestCircleRectangle.java.

```

import java.util.Scanner;

public class TestCircleRectangle {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        Circle circle = new Circle(1);
        System.out.println("Круг " + circle.toString());
        System.out.println("Радиус равен " + circle.getRadius());
        System.out.println("Площадь равна " + circle.getArea());
        System.out.println("Диаметр равен " + circle.getDiameter());

        Rectangle rectangle = new Rectangle(2, 4);
        System.out.println("\nПрямоугольник " + rectangle.toString());
        System.out.println("Площадь равна " + rectangle.getArea());
        System.out.println("Периметр равен " +
rectangle.getPerimeter());

        try{

```

```

        System.out.print("Введите первую сторону треугольника: ");
        double side1 = input.nextDouble();
        System.out.print("Введите вторую сторону треугольника: ");
        double side2 = input.nextDouble();
        System.out.print("Введите третью сторону треугольника: ");
        double side3 = input.nextDouble();
        System.out.print("Введите цвет треугольника: ");
        String color = input.next();
        System.out.print("Заполнен ли треугольник? (true / false): ");
        boolean filled = input.nextBoolean();

        Triangle triangle = new Triangle(side1, side2, side3);
        triangle.setColor(color);
        triangle.setFilled(filled);
        System.out.println("\n" + triangle.toString());
        System.out.println("Площадь равна " + triangle.getArea());
        System.out.println("Периметр равен " + triangle.getPerimeter());
        System.out.println("Цвет: " + triangle.getColor());
        System.out.println("Заполнен: " + triangle.isFilled());
    }
    catch (IllegalTriangleException e) {
        System.out.println(e.getMessage());
    }
}
}

```

Листинг 7.8 — Реализация класса `TestCircleRectangle1` в `TestCircleRectangle1.java`.

```

import java.util.Scanner;

public class TestCircleRectangle1 {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        try {
            System.out.print("Введите первую сторону треугольника: ");
            double side1 = input.nextDouble();
            System.out.print("Введите вторую сторону треугольника: ");
            double side2 = input.nextDouble();
            System.out.print("Введите третью сторону треугольника: ");
            double side3 = input.nextDouble();
            System.out.print("Введите цвет треугольника: ");
            String color = input.next();
            System.out.print("Заполнен ли треугольник? (true/false): ");
            boolean filled = input.nextBoolean();

            Triangle triangle = new Triangle(side1, side2, side3);
            triangle.setColor(color);
            triangle.setFilled(filled);
            System.out.println("\n" + triangle.toString());
            System.out.println("Площадь: " + triangle.getArea());
            System.out.println("Периметр: " + triangle.getPerimeter());
            System.out.println("Цвет: " + triangle.getColor());
            System.out.println("Заполнен: " + triangle.isFilled());

            Circle circle1 = new Circle(3);
            circle1.setColor("red");
            Circle circle2 = new Circle(4);
            circle2.setColor("blue");
            System.out.println("\nБОЛЬШИЙ из двух кругов: " +
                GeometricObject.max(circle1, circle2).toString());

            Rectangle rectangle1 = new Rectangle(5, 14);
            rectangle1.setColor("red");
            Rectangle rectangle2 = new Rectangle(4, 15);

```

```

        rectangle2.setColor("blue");
        System.out.println("Больший из двух прямоугольников: " +
GeometricObject.max(rectangle1, rectangle2).toString());

        ComparableCircle compCircle1 = new ComparableCircle(6);
        ComparableCircle compCircle2 = new ComparableCircle(7);
        System.out.println("Сравнение двух кругов:");
        if (circle1.compareTo(compCircle1) > 0) {
            System.out.println("circle1 больше circle2");
        } else if (circle1.compareTo(compCircle2) < 0) {
            System.out.println("circle2 больше circle1");
        } else {
            System.out.println("Оба круга равны по площади");
        }

        // Сравниваем круг и прямоугольник, используя метод max для
GeometricObject
        System.out.println("\nСравнение круга и прямоугольника:");
        if (circle1.compareTo(rectangle1) > 0) {
            System.out.println("Круг больше прямоугольника");
        } else {
            System.out.println("Прямоугольник больше круга");
        }
    } catch (IllegalArgumentException e) {
        System.out.println("Ошибка: " + e.getMessage());
    }
}
}

```

Листинг 7.9 — Реализация класса TestCircleRectangle2 в TestCircleRectangle2.java.

```

import java.util.Scanner;

public class TestCircleRectangle2 {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        GeometricObject[] objects = new GeometricObject[5];
        objects[0] = new Square(4);
        objects[1] = new Circle(3);
        objects[2] = new Rectangle(2, 5);
        objects[3] = new Square(7);
        objects[4] = new Circle(9);
        int i = 1;
        for (GeometricObject obj : objects) {
            System.out.println("Объект номер: " + i);
            System.out.println("Площадь объекта: " + obj.getArea());

            if (obj instanceof Colorable) {
                ((Colorable) obj).howToColor();
            }
            System.out.println();
            i++;
        }
    }
}

```

Листинг 7.10 — Реализация интерфейса Colorable в Colorable.java.

```

public interface Colorable {
    void howToColor();
}

```

Результат работы программы

```
Круг создан Wed Dec 18 22:31:49 MSK 2024,  
цвет: белый, заливка: false  
Радиус равен 1.0  
Площадь равна 3.141592653589793  
Диаметр равен 2.0  
  
Прямоугольник создан Wed Dec 18 22:31:49 MSK 2024,  
цвет: белый, заливка: false  
Площадь равна 8.0  
Периметр равен 12.0  
Введите первую сторону треугольника: 3  
Введите вторую сторону треугольника: 4  
Введите третью сторону треугольника: 5  
Введите цвет треугольника: red  
Заполнен ли треугольник? (true / false): true  
  
Треугольник: сторона1 = 3.0, сторона2 = 4.0, сторона3 = 5.0  
Площадь равна 6.0  
Периметр равен 12.0  
Цвет: red  
Заполнен: true  
PS C:\MyProfile\univer\github\Java-University\практика 7>
```

Рисунок 7.1 Результат работы №1.

```
Введите первую сторону треугольника: 3  
Введите вторую сторону треугольника: 4  
Введите третью сторону треугольника: 5  
Введите цвет треугольника: red  
Заполнен ли треугольник? (true/false): true  
  
Треугольник: сторона1 = 3.0, сторона2 = 4.0, сторона3 = 5.0  
Площадь: 6.0  
Периметр: 12.0  
Цвет: red  
Заполнен: true  
  
Больший из двух кругов: создан Wed Dec 18 22:32:34 MSK 2024,  
цвет: blue, заливка: false  
Больший из двух прямоугольников: создан Wed Dec 18 22:32:34 MSK 2024,  
цвет: red, заливка: false  
Сравнение двух кругов:  
circle2 больше circle1  
  
Сравнение круга и прямоугольника:  
Прямоугольник больше круга  
PS C:\MyProfile\univer\github\Java-University\практика 7>
```

Рисунок 7.2 Результат работы №2.

```
C:\storage\5636297b45edf0c931988f0ce88c3088\rednat.java  
Объект номер: 1  
Площадь объекта: 16.0  
Раскрасьте все четыре стороны.  
  
Объект номер: 2  
Площадь объекта: 28.274333882308138  
Раскрасьте круг  
  
Объект номер: 3  
Площадь объекта: 10.0  
  
Объект номер: 4  
Площадь объекта: 49.0  
Раскрасьте все четыре стороны.  
  
Объект номер: 5  
Площадь объекта: 254.46900494077323  
Раскрасьте круг  
PS C:\MyProfile\univer\github\Java-University\практика 7>
```

Рисунок 7.3 Результат работы №3.

Вывод

Был реализован треугольник, прямоугольник, круг. Также была реализована проверка на то, закрашена ли фигура и метод сравнения разных фигур по одному признаку.

ПРАКТИЧЕСКАЯ РАБОТА 8

Теоретическое введение

В Java API существует множество predefined классов исключений. Исключения – это объекты, а объекты определяются с помощью классов. Корневым классом для исключений является `java.lang.Throwable`. Классы исключений можно разделить на три основных типа: системные ошибки, исключения и ошибки во время выполнения. В Java можно определить пользовательский класс исключений, породив его от класса `java.lang.Exception`.

Системные ошибки представлены в классе `Error`, их выбрасывает JVM. Класс `Error` описывает внутренние ошибки системы, хотя такие ошибки и происходят очень редко. Если все-таки они происходят, то мало что можно сделать, кроме уведомления пользователя и попытки завершить программу.

Исключения представлены в классе `Exception`, который описывает ошибки, вызванные самой программой и внешними обстоятельствами. Эти ошибки можно обнаружить и обработать с помощью программы.

Ошибки во время выполнения представлены в классе `RuntimeException`, который описывает ошибки программирования, такие как неправильное приведение типов, доступ к массиву вне его границ и числовые ошибки.

`RuntimeException`, `Error` и их подклассы называются непроверяемыми (unchecked) исключениями. Все остальные исключения называются проверяемыми (checked) исключениями, то есть компилятор заставляет программиста проверять и обрабатывать их в блоке `try-catch` или объявлять в заголовке метода.

Постановка задачи

Задача 1: С помощью двух массивов напишите программу, которая предложит пользователю ввести целое число от 1 до 12, а затем отобразит месяц

и количество дней, соответствующие этому целому числу. Если пользователь вводит недопустимое число, то программа должна отображать Недопустимое число с помощью перехвата `ArrayIndexOutOfBoundsException`. Программа должна быть реализована таким образом, чтобы предотвратить ввод пользователем любого числа, кроме целого.

Задача 2: Измените в программе `TestLoanClass` класс `Loan` таким образом, чтобы выбросить `IllegalArgumentException`, если годовая процентная ставка, срок или сумма кредита меньше или равны нулю.

Задача 3: Дополните код 1 задания данной практической работы таким образом, чтобы при выборе февраля была возможность записать год. Добавьте в программу метод для расчета является ли год високосным и измените вывод для количества дней в феврале.

Ход выполнения работы

Реализуем программу-календарь, а также изменим программу так, чтобы при вводе неправильных значений нам выдавалась ошибка.

Программный код

Листинг 8.1 — Реализация программа-календаря в классе `Main1` в `Main1.java`.

```
import java.util.InputMismatchException;
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        String[] months = {"январь", "февраль", "март", "апрель", "май",
                           "июнь", "июль", "август", "сентябрь", "октябрь", "ноябрь",
                           "декабрь"};

        int[] dom = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
        Scanner scan = new Scanner(System.in);
        System.out.println("Введите число от 1 до 12");
        int month_number = 1;
        /*
        while(true) {
            boolean f = true;

            if(f)break;
        }*/
        try {
            month_number = scan.nextInt();
        } catch (InputMismatchException e) {
```



```

        System.out.println("Введен не int, значение месяца задан 1");
        //f = false;
    }

    int pl = 0;
    if (month_number == 2){
        System.out.println("Введите год:");
        int year = scan.nextInt();
        if((year % 4 == 0 && year % 100 != 0) || year % 400 == 0)pl = 1;
    }
    try {
        System.out.println("Месяц: " + months[month_number - 1] + "
Количество дней в нем: " + (dom[month_number - 1] + pl));
    } catch (ArrayIndexOutOfBoundsException e){
        System.out.println("Недопустимое");
    }
}
}
}

```

Листинг 8.2 —Изменения в классе *Loan* в *Loan.java*.

```

public class Loan {
    private double annualInterestRate;
    private int numberOfYears;
    private double loanAmount;
    private java.util.Date loanDate;

    public Loan() {
        this(2.5, 1, 1000);
    }

    /** Создает кредит с указанными: годовой процентной ставкой,
     *  количеством лет и суммой кредита
     */
    public Loan(double annualInterestRate, int numberOfYears, double loanAmount)
    throws IllegalArgumentException{
        if(annualInterestRate <= 0 || numberOfYears <= 0 || loanAmount <= 0)
            throw new IllegalArgumentException("Параметры не могу равняться 0");
        this.annualInterestRate = annualInterestRate;
        this.numberOfYears = numberOfYears;
        this.loanAmount = loanAmount;
        loanDate = new java.util.Date();
    }

    /** Возвращает годовую процентную ставку */
    public double getAnnualInterestRate() {
        return annualInterestRate;
    }

    /** Присваивает новую годовую процентную ставку */
    public void setAnnualInterestRate(double annualInterestRate) {
        this.annualInterestRate = annualInterestRate;
    }

    /** Возвращает количество лет */
    public int getNumberOfYears() {
        return numberOfYears;
    }

    /** Присваивает новое количество лет */
    public void setNumberOfYears(int numberOfYears) {
        this.numberOfYears = numberOfYears;
    }
}

```

```

    /** Возвращает сумму кредита */
    public double getLoanAmount() {
        return loanAmount;
    }

    /** Присваивает новую сумму кредита */
    public void setLoanAmount(double loanAmount) {
        this.loanAmount = loanAmount;
    }

    /** Вычисляет и возвращает ежемесячный платеж по кредиту */
    public double getMonthlyPayment() {
        double monthlyInterestRate = annualInterestRate / 1200;
        double monthlyPayment = loanAmount * monthlyInterestRate /
            (1 - (1 / Math.pow(1 + monthlyInterestRate, numberOfYears *
12)));
        return monthlyPayment;
    }

    /** Вычисляет и возвращает общую стоимость кредита */
    public double getTotalPayment() {
        double totalPayment = getMonthlyPayment() * numberOfYears * 12;
        return totalPayment;
    }

    /** Возвращает дату взятия кредита */
    public java.util.Date getLoanDate() {
        return loanDate;
    }
}

```

Листинг 8.3 —Изменение программы-календаря в Main в Main.java.

```

import java.util.InputMismatchException;
import java.util.Scanner;

public class Main {

    static boolean isLeapYear(int year) {
        return (year % 4 == 0 && year % 100 != 0) || (year % 400 == 0);
    }

    public static void main(String[] args) {
        String[] months = {"январь", "февраль", "март", "апрель", "май", "июнь",
"июль", "август", "сентябрь", "октябрь", "ноябрь", "декабрь"};
        int[] dom = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
        Scanner scanner = new Scanner(System.in);
        try {
            System.out.print("Введите целое число от 1 до 12: ");
            int month = scanner.nextInt();
            if (month == 2) {
                System.out.print("Введите год: ");
                int year = scanner.nextInt();
                if (isLeapYear(year)) {
                    dom[1] = 29;
                }
            }
            System.out.println("Месяц: " + months[month - 1]);
            System.out.println("Количество дней: " + dom[month - 1]);
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Недопустимое число. Введите число от 1 до 12.");
        } catch (InputMismatchException e) {
            System.out.println("Ошибка ввода. Введите целое число.");
        }
    }
}

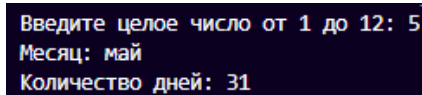
```

```

    } finally {
        scanner.close();
    }
}

```

Результат работы программы

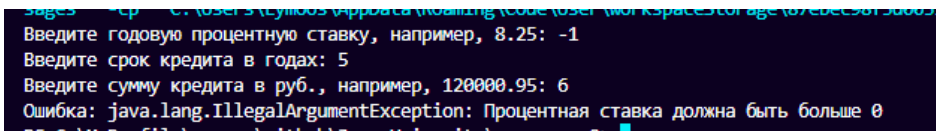


```

Введите целое число от 1 до 12: 5
Месяц: май
Количество дней: 31

```

Рисунок 8.1 Результат работы №1.

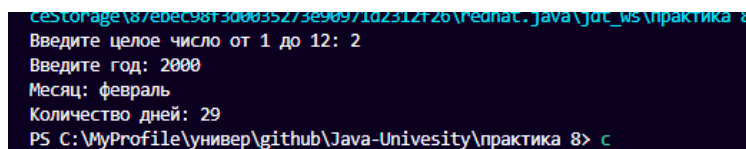


```

Введите годовую процентную ставку, например, 8.25: -1
Введите срок кредита в годах: 5
Введите сумму кредита в руб., например, 120000.95: 6
Ошибка: java.lang.IllegalArgumentException: Процентная ставка должна быть больше 0

```

Рисунок 8.2 Результат работы №2.



```

Введите целое число от 1 до 12: 2
Введите год: 2000
Месяц: февраль
Количество дней: 29

```

Рисунок 8.3 Результат работы №3.

Вывод

Был реализован календарь, определяющий, високосный ли год сейчас, и выводящий количество дней в заданном месяце. Также был изменён класс Loan, для того чтобы он реализовывал проверку входных данных.

ПРАКТИЧЕСКАЯ РАБОТА 9

Теоретическое введение

Дженерики (generics – в пер. с англ. «обобщения») позволяют обнаруживать ошибки уже во время компиляции программы, а не во время ее выполнения. Дженерики позволяют параметризовать типы, т.е. сделать типы зависимыми от параметров. С помощью этой возможности можно определить класс или метод с обобщенными типами, которые компилятор может заменить на конкретные. Например, Java определяет обобщенный класс ArrayList для хранения элементов обобщенного типа. Из этого обобщенного класса можно создать объект типа ArrayList для хранения строк и объект типа ArrayList для хранения чисел. Здесь строки и числа являются конкретными типами, на которые заменяется обобщенный. Java позволяет определять обобщенные классы, интерфейсы и методы, начиная с JDK 1.5. Несколько интерфейсов и классов в Java API были изменены с помощью дженериков. Например, до JDK 1.5 интерфейс java.lang.Comparable был определен, как показано слева, а начиная с JDK 1.5 он был изменен, как показано справа..

Постановка задачи

Задача 1: Напишите метод, которому передается коллекция объектов типа ArrayList, а возвращается коллекция ArrayList, но уже без дубликатов. Необходимо использовать метод contains() интерфейса List. 2. Реализуйте алгоритм линейного поиска элемента в массиве. При нахождении элемента необходимо вернуть его позицию в массиве. Если элемент не найден, то вернуть -1. 3. Реализуйте поиск наибольшего элемента в массиве с помощью метода compareTo() интерфейса Comparable. Определите класс Circle с полем radius и найдите наибольший элемент в массиве экземпляров этого класса. 4. Реализуйте

поиск наибольшего элемента в двумерном массиве с помощью метода `compareTo()` интерфейса `Comparable`.

Задача 2: Измените класс `GenericStack` таким образом, чтобы реализовать его с помощью массива, а не `ArrayList`. Перед добавлением нового элемента в стек необходимо проверить размер массива. Если массив заполнен, создайте новый массив, который удвоит текущий размер массива и скопирует элементы из текущего массива в новый. 2. Класс `GenericStack` из предыдущего описания реализован с помощью отношения композиции. Определите новый класс стека, который наследуется от `ArrayList`. Нарисуйте UML-диаграмму этих классов, а затем реализуйте новый класс `GenericStack`. Напишите тестовую программу, которая запросит у пользователя пять строк, а отобразит их в обратном порядке.

Ход выполнения работы

Реализуем сначала Стек с помощью `ArrayList`, а далее с помощью массива.

Программный код

Листинг 9.1 — Реализация класса `GenericStack`

```
import java.util.Arrays;

public class GenericStack<E> {
    private E[] List;
    int last = -1;
    public int getSize() {
        return List.length;
    }
    public E peek() {
        return List[last];
    }
    public void push(E o) {
        last++;
        if(last == List.length){
            List = Arrays.copyOf(List, List.length * 2);
        }
        List[last] = o;
    }
    public E pop() {
        E o = List[last];
        E[] temp = (E[]) new Object[List.length - 1];
        for (int i = 0; i < last; i++) {
            temp[i] = List[i];
        }
        List = temp;
    }
}
```

```

        last--;
        return o;
    }
    public boolean isEmpty() {
        return List.length == 0;
    }
    @Override
    public String toString() {
        String str = "";
        for (int i = 0; i < List.length; i++) {
            str += List[i] + " ";
        }
        return "стек: " + str;
    }

    public GenericStack() {
        List = (E[]) new Object[1];
    }
}

```

Листинг 9.2 — Реализация класса Circle.

```

public class Circle implements Comparable<Circle>{
    private double radius;

    public Circle(double radius) {
        this.radius = radius;
    }

    public double getRadius() {
        return radius;
    }

    public void setRadius(double radius) {
        this.radius = radius;
    }

    @Override
    public int compareTo(Circle o) {
        if(this.radius > o.getRadius())return 1;
        else if(this.radius < o.getRadius()) return -1;
        else return 0;
    }

    @Override
    public String toString() {
        return "Circle{" +
            "radius=" + radius +
            '}';
    }
}

```

Листинг 9.3 — Реализация класса ArrayStack

```

import java.util.ArrayList;

public class ArrayStack<E> extends ArrayList<E> {
    private java.util.ArrayList<E> list = new java.util.ArrayList<>();
    public int getSize() {
        return list.size();
    }
    public E peek() {
        return list.get(getSize() - 1);
    }
    public void push(E o) {

```

```

        list.add(o);
    }
    public E pop() {
        E o = list.get(getSize() - 1);
        list.remove(getSize() - 1);
        return o;
    }
}

```

Листинг 9.4 — Тестирование программы

```

import java.util.ArrayList;

public class Main {
    static<T extends Comparable<T>> T find_max(T[]a){
        T mx = a[0];
        for (int i = 0; i < a.length; i++) {
            if(a[i].compareTo(mx) > 0)mx = a[i];
        }
        return mx;
    }

    public static void main(String[] args) {
        Circle[] arr = {new Circle(5.5), new Circle(5.0), new Circle(8.0)};
        System.out.println(find_max(arr).toString());
        GenericStack<String> stack = new GenericStack<String>();
        stack.push("1");
        stack.push("2");
        stack.push("3");
        System.out.println(stack.toString());
        System.out.println(stack.peek());
        System.out.println(stack.pop());
        System.out.println(stack.pop());
        System.out.println(stack.peek());
        System.out.println(stack.toString());
        ArrayStack<String>st = new ArrayStack<String>();
        st.push("1");
        st.push("2");
        st.push("3");
        st.push("4");
        st.push("5");
        for (int i = 0; i < 5; i++) {
            System.out.println(st.pop());
        }
    }
}

```

Результат работы программы

```
Circle{radius=8.0}  
стек: 1 2 3 null  
3  
3  
2  
1  
стек: 1 null  
5  
4  
3  
2  
1
```

Рисунок 9.1 Результат работы программы.

Вывод

Был реализован стек при помощи двух контейнеров: ArrayList и массив.

ПРАКТИЧЕСКАЯ РАБОТА 10

Теоретическое введение

Напомним, что стек — это структура данных, которая содержит данные по принципу «последним пришел — первым обслужен». У стека есть множество применений. Например, компилятор использует стек для обработки вызовов методов. При вызове метода его параметры и локальные переменные помещаются в стек (операция «push»). Когда этот метод вызывает другой метод, параметры и локальные переменные уже нового метода также помещаются в стек. Когда новый метод завершает свою работу и возвращается к вызвавшей его стороне, связанное с ним пространство освобождается из стека (операция «pop»). Для моделирования стеков можно определить класс. Предположим, что стек в качестве данных содержит объекты. Для реализации стека объектов можно использовать класс `ArrayList`.

Чтобы определить пользовательский класс, который реализует интерфейс `Cloneable`, этот класс должен переопределить метод `clone()` в классе `Object`. В следующем коде определяется класс `House`, который реализует интерфейсы `Cloneable` и `Comparable`.

Ключевое слово `native` указывает, что этот метод написан не на языке Java, а реализован в JVM для собственной платформы. Ключевое слово `protected` ограничивает доступ к методу в том же пакете или подклассе. По этой причине класс `House` должен переопределить этот метод и изменить модификатор доступа на `public`, чтобы этот метод можно было использовать в любом пакете. Поскольку метод `clone()`, реализованный для собственной платформы в классе `Object`, выполняет задачу клонирования объектов, метод `clone()` в классе `House` просто вызывает `super.clone()`. Метод `clone()`, определенный в классе `Object`, выбрасывает исключение `CloneNotSupportedException`, если объект не типа `Cloneable`. Поскольку мы перехватываем это исключение внутри метода, нет необходимости объявлять его в заголовке метода `clone()`.

Java предоставляет собственный метод, который делает поверхностную копию для клонирования объекта. Поскольку метод в интерфейсе является абстрактным, этот собственный метод не может быть реализован в интерфейсе. Поэтому для языка Java было решено определить и реализовать собственный метод `clone()` в классе `Object`.

Постановка задачи

Задача 1: Определить класс `MyStack`. Убедитесь что класс `MyStack` реализован с помощью композиции. Определите новый класс `MyStack`, который наследуется от класса `ArrayList`. Нарисуйте UML-диаграмму этих двух классов, а затем реализуйте класс `MyStack`. Напишите клиент класса `MyStack` — программу, которая запрашивает у пользователя пять строк и отображает их в обратном порядке. 3. Перепишите класс `MyStack` для выполнения глубокой копии поля списка.

Ход выполнения работы

Реализуем сначала Стек и его глубокое копирование.

Программный код

Листинг 10.1 — Реализация класса `MyStack`

```
public class MyStack<E> implements Cloneable {
    private java.util.ArrayList<E> list = new java.util.ArrayList<>();
    public int getSize() {
        return list.size();
    }
    public E peek() {
        return list.get(getSize() - 1);
    }
    public void push(E o) {
        list.add(o);
    }
    public E pop() {
        E o = list.get(getSize() - 1);
        list.remove(getSize() - 1);
        return o;
    }
    public boolean isEmpty() {
```

```

        return list.isEmpty();
    }
    @Override
    public String toString() {
        return "стек: " + list.toString();
    }
    public Object clone() throws CloneNotSupportedException{
        MyStack<E> stck = (MyStack<E>)super.clone();
        stck.list = (java.util.ArrayList<E>)(list.clone());
        return stck;
    }
}

```

Листинг 10.2 — Реализация класса MyStack2

```

import java.util.ArrayList;

public class MyStack2<E> extends ArrayList<E> {
    private java.util.ArrayList<E> list = new java.util.ArrayList<>();
    public int getSize() {
        return list.size();
    }
    public E peek() {
        return list.get(getSize() - 1);
    }
    public void push(E o) {
        list.add(o);
    }
    public E pop() {
        E o = list.get(getSize() - 1);
        list.remove(getSize() - 1);
        return o;
    }
}

```

Листинг 10.3 — Тестирование программы

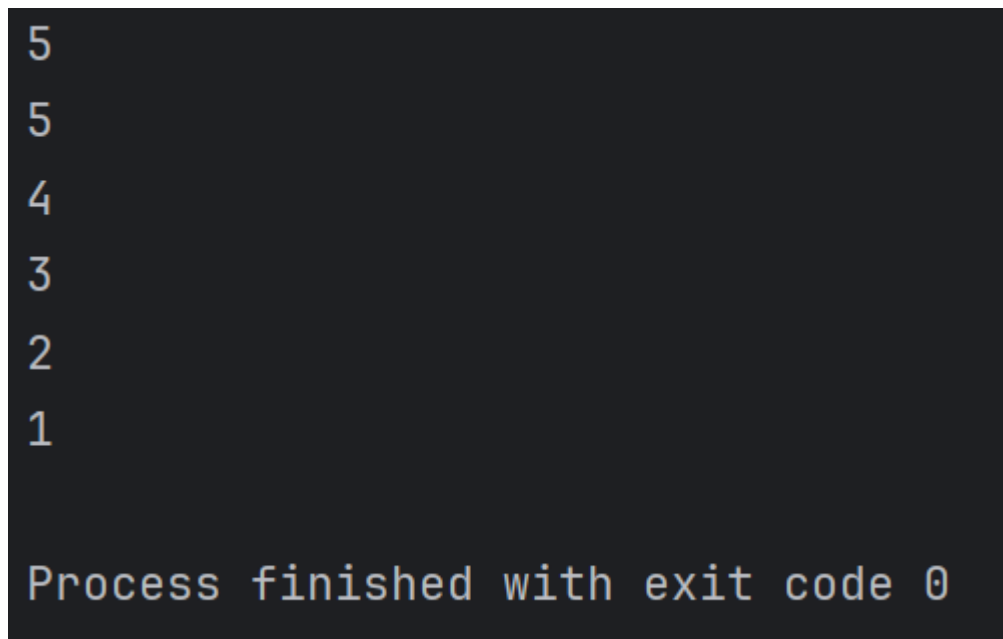
```

public class Main {
    public static void main(String[] args) {
        MyStack<String> s = new MyStack<>();
        MyStack<String> s2;
        s.push("1");
        s.push("2");
        s.push("3");
        s.push("4");
        s.push("5");
        try{
            s2 = (MyStack<String>)s.clone();
            System.out.println(s2.peek());

        }catch(CloneNotSupportedException e){
            System.out.println("Ошибка");
        }
        for (int i = 0; i < 5; i++) {
            System.out.println(s.pop());
        }
    }
}

```

Результат работы программы



```
5
5
4
3
2
1

Process finished with exit code 0
```

Рисунок 10.1 Результат работы.

Вывод

Был реализован стек и его копирование.

ПРАКТИЧЕСКАЯ РАБОТА 11

Теоретическое введение

Очереди обычно, но не обязательно, упорядочивают элементы по принципу FIFO (первым пришёл — первым ушёл). Исключением являются очереди с приоритетом, которые упорядочивают элементы в соответствии с заданным компаратором или естественным порядком элементов. Независимо от используемого порядка, первым элементом очереди является тот элемент, который будет удалён вызовом `remove()` или `poll()`. В очереди FIFO («первый пришел – первый ушел») все новые элементы добавляются в конец очереди. В других типах очередей могут использоваться другие правила размещения.

Постановка задачи

Задача 1: Реализуйте стек (LIFO — «последний пришел – первый ушел»), используя не более двух очередей (FIFO — «первый пришел – первый ушел») и только стандартные методы очереди `add/offer`, `peek`, `poll`, `size`, `isEmpty`. Для проектирования используйте отношение композиция. В реализованном стеке должны поддерживаться методы `push`, `top`, `pop`, `empty` и метод, возвращающий строковое представление всех элементов стека. Методы класса `StackOnQueue`:

- `void push(int x)` Помещает элемент `x` на вершину стека.
- `int pop()` Удаляет элемент на вершине стека и возвращает его.
- `int top()` Возвращает элемент на вершине стека.
- `boolean empty()` Возвращает `true`, если стек пуст, в ином случае `false`.

Задача 2: Создайте новый тестовый класс. Для проверки работы созданного стека создайте в тестовом классе новый экземпляр класса `StackOnQueue`, добавьте в этот стек два значения, выведите объект, находящийся на вершине стека (без удаления), выведите объект, находящийся на вершине стека и удалите его, проверьте стек на пустоту и выведите информацию о всех элементах стека.

Ход выполнения работы

Реализуем стек на двух очередях и напомним тестовый класс для проверки стека.

Программный код

Листинг 11.1 — Реализация класса Stack в Stack.java.

```
import java.util.LinkedList;
import java.util.Queue;

public class Stack {
    private Queue<Integer> queue1;
    private Queue<Integer> queue2;

    public Stack() {
        queue1 = new LinkedList<>();
        queue2 = new LinkedList<>();
    }

    public void push(int x) {
        queue2.add(x);
        while (!queue1.isEmpty()) {
            queue2.add(queue1.poll());
        }
        Queue<Integer> temp = new LinkedList<>();
        queue1 = queue2;
        queue2 = temp;
    }

    public int pop() {
        if (queue1.isEmpty()) {
            throw new IllegalStateException("Stack is empty");
        }
        return queue1.poll();
    }

    public int top() {
        if (queue1.isEmpty()) {
            throw new IllegalStateException("Stack is empty");
        }
        return queue1.peek();
    }

    public boolean empty() {
        return queue1.isEmpty();
    }

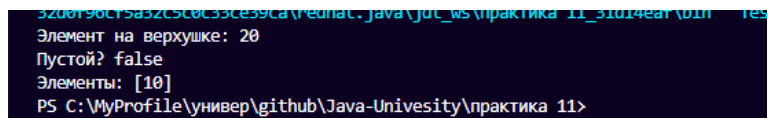
    @Override
    public String toString() {
        return queue1.toString();
    }
}
```

Листинг 11.2 — Реализация класса Test в Test.java.

```
public class Test {
    public static void main(String[] args) {
        Stack stack = new Stack();
        stack.push(10);
        stack.push(20);
    }
}
```

```
        System.out.println("Элемент на верхушке: " + stack.top());  
        stack.pop();  
        System.out.println("Пустой? " + stack.empty());  
        System.out.println("Элементы: " + stack.toString());  
    }  
}
```

Результат работы программы

A screenshot of a terminal window showing the output of a Java program. The output consists of four lines: "Элемент на верхушке: 20", "Пустой? false", "Элементы: [10]", and a prompt "PS C:\MyProfile\универ\github\Java-Univesity\практика 11>". The text is displayed in a monospaced font on a dark background.

```
Элемент на верхушке: 20  
Пустой? false  
Элементы: [10]  
PS C:\MyProfile\универ\github\Java-Univesity\практика 11>
```

Рисунок 11.1 Результат работы.

Вывод

Был реализован стек, используя только 2.

ПРАКТИЧЕСКАЯ РАБОТА 12

Теоретическое введение

Паттерны проектирования — это повторяющиеся решения типичных задач проектирования программного обеспечения. Они помогают сделать код более гибким, поддерживаемым и повторно используемым. Порождающие паттерны проектирования играют важную роль в управлении процессом создания объектов, предоставляя гибкие способы инкапсуляции создания объектов и обеспечения их корректного использования. Порождающие паттерны (Creational Patterns) — это группа паттернов проектирования, которые фокусируются на процессе создания объектов. Они помогают абстрагировать или скрыть сложность создания объектов, обеспечивая гибкость и независимость системы от конкретных классов объектов.

Основные задачи порождающих паттернов: 1. Инкапсуляция создания объектов: отделение логики создания объекта от его использования. 2. Управление сложностью: упрощение создания сложных объектов. 3. Поддержка гибкости: предоставление возможности изменения способа создания объектов без модификации кода, использующего эти объекты.

Основные порождающие паттерны

- Singleton (Одиночка)
- Factory Method (Фабричный метод)
- Abstract Factory (Абстрактная фабрика)
- Builder (Строитель)
- Prototype (Прототип)

Постановка задачи

В данной практической работе представлено 5 вариантов заданий. Выбор варианта осуществляется в соответствии с порядковым номером студента в

списке. Если порядковый номер студента равен 1, выполняется задание варианта №1; если порядковый номер равен 5 — выполняется вариант №5. В случае, если порядковый номер превышает количество вариантов (например, 6 или более), задание выбирается циклически, начиная с варианта №1..

Вариант задания 1: Реализовать систему управления настройками приложения. Необходимо создать класс `AppSettings`, который будет использоваться для хранения и управления настройками (например, тема, язык, путь к файлам). Гарантировать, что класс `AppSettings` будет иметь только один экземпляр. Реализовать методы для установки и получения настроек. Продемонстрировать работу Singleton, используя несколько потоков

Ход выполнения работы

Реализуем Singleton.

Программный код

Листинг 12.1 — Реализация класса `AppSettings`.

```
import java.util.HashMap;

public class AppSettings {
    private static HashMap<String, Integer> settings = new HashMap<>();
    private static AppSettings instance;
    private AppSettings() {}
    public static AppSettings getInstance() {
        if (instance == null) {
            instance = new AppSettings();
        }
        return instance;
    }
    public void setSettings(String key, Integer value) {
        settings.put(key, value);
    }
    public Integer getSettings(String key) {
        return settings.get(key);
    }
}
```

Листинг 12.2 – Тестирование синглтона

```
public class Main {
    public static void main(String[] args) {
        AppSettings as = AppSettings.getInstance();
        Thread thr = new Thread(new Runnable() {
            @Override
            public void run() {
```

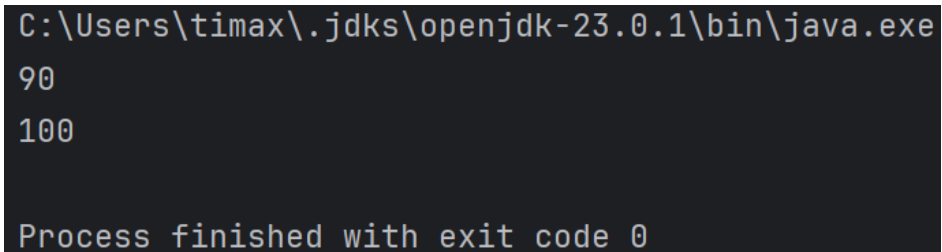
```

        as.setSettings("brightness", 100);
    }
});

Thread thr2 = new Thread(new Runnable() {
    @Override
    public void run() {
        as.setSettings("sensetivity", 90);
    }
});
Thread t = Thread.currentThread();
thr.start();
thr2.start();
try {
    t.sleep(100);
} catch (InterruptedException e) {
    throw new RuntimeException(e);
}
System.out.println(as.getSettings("sensetivity"));
System.out.println(as.getSettings("brightness"));
}
}

```

Результат работы программы



```

C:\Users\timax\.jdk\openjdk-23.0.1\bin\java.exe
90
100

Process finished with exit code 0

```

Рисунок 12.1 Результат работы.

Вывод

Был реализован Singleton

ПРАКТИЧЕСКАЯ РАБОТА 13

Теоретическое введение

Паттерны проектирования — это повторяющиеся решения типичных задач проектирования программного обеспечения. Они помогают сделать код более гибким, поддерживаемым и повторно используемым. Порождающие паттерны проектирования играют важную роль в управлении процессом создания объектов, предоставляя гибкие способы инкапсуляции создания объектов и обеспечения их корректного использования. Структурные паттерны (Structural Patterns) — это группа паттернов проектирования, которые фокусируются на способах организации классов и объектов в более крупные структуры. Они обеспечивают гибкость и удобство взаимодействия между компонентами системы, способствуя уменьшению связности и упрощению поддержки кода. Эти паттерны позволяют строить сложные структуры, сохраняя при этом простоту и читаемость кода.

Основные задачи структурных паттернов:

1. Упрощение архитектуры: Структурные паттерны позволяют объединять объекты и классы так, чтобы они работали как единое целое, сохраняя при этом чёткое разделение обязанностей.
2. Повышение гибкости системы: благодаря абстрагированию взаимодействия между компонентами системы можно изменять её структуру без затрагивания остального кода.
3. Улучшение повторного использования: Компоненты можно комбинировать и использовать повторно в различных частях приложения.

Основные структурные паттерны

- Adapter (Адаптер)
- Composite (Компоновщик)
- Decorator (Декоратор)
- Facade (Фасад)
- Flyweight (Приспособленец)
- Proxy (Заместитель)

Постановка задачи

В данной практической работе представлено 7 вариантов заданий. Выбор варианта осуществляется в соответствии с порядковым номером студента в списке. Если порядковый номер студента равен 1, выполняется задание варианта №1; если порядковый номер равен 7 — выполняется вариант №7. В случае, если порядковый номер превышает количество вариантов (например, 8 или более), задание выбирается циклически, начиная с варианта №1.

Вариант задания 3: Создать иерархическую систему управления файлами. В системе должны быть директории и файлы, причём каждая директория может содержать как файлы, так и другие директории. Реализовать интерфейс `FileComponent` с методами для добавления и отображения содержимого. Создать классы для файлов и директорий. Организовать древовидную структуру иерархии файлов.

Ход выполнения работы

Реализуем `Composite`.

Программный код

Листинг 13.1 – Реализация интерфейса `FileComponent`

```
public interface FileComponent {  
    void addNode(Node n);  
    void printNode(int n);  
}
```

Листинг 13.2 – Реализация класса узла

```
import java.util.List;  
  
public abstract class Node {  
    private String name;  
    public abstract void printNode(int n);  
    public abstract List<Node> getNodes();  
}
```

Листинг 13.3 – Реализация класса файлов

```
import java.util.List;  
  
public class File extends Node implements FileComponent{  
    private String name;  
    private int size;
```

```

public File(String name, int size) {
    this.name = name;
    this.size = size;
}

public String getName() {
    return name;
}

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

public int getSize() {
    return size;
}

public void setSize(int size) {
    this.size = size;
}

public String toString(){
    return "File: " + name + "; Size: " + size + "b";
}

@Override
public void addNode(Node n) {
    System.out.println("Нельзя добавить файл, т.к это не является
директорией");
}

@Override
public void printNode(int n) {
    String s = "";
    for (int i = 0; i < n; i++) {
        s+=" ";
    }
    System.out.println(s + this);
}

@Override
public List<Node> getNodes() {
    return null;
}
}

```

Листинг 13.4 – Реализация класса директории

```

import java.util.ArrayList;
import java.util.List;

public class Directorie extends Node implements FileComponent{
    private String name;
    private int size;
    private List<Node>nodes;
    public Directorie(String name, int size) {
        this.name = name;
        this.size = size;
        nodes = new ArrayList<>();
    }

    public void add(Node n){
        nodes.add(n);
    }
}

```

```

    }

    public void remove(Node n) {
        nodes.remove(n);
    }

    public List<Node> getNodes() {
        return nodes;
    }

    public String getName() {
        return name;
    }

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

    public int getSize() {
        return size;
    }

    public void setSize(int size) {
        this.size = size;
    }

    public String toString() {
        return "\\\" + name;
    }

    @Override
    public void addNode(Node n) {
        nodes.add(n);
    }

    @Override
    public void printNode(int n) {
        String s = "";
        for (int i = 0; i < n; i++) {
            s += " ";
        }
        System.out.println(s + toString());
    }
}

```

Листинг 13.5 – Тестировка программы

```

public class Main {
    public static void main(String[] args) {
        Directorie home = new Directorie("home", 0);
        Directorie picture = new Directorie("pictures", 0);
        Directorie docs = new Directorie("documents", 0);
        File pic1 = new File("pic1.png", 100);
        File pic2 = new File("pic2.jpeg", 200);
        File docx1 = new File("zadanie.docx", 1000);

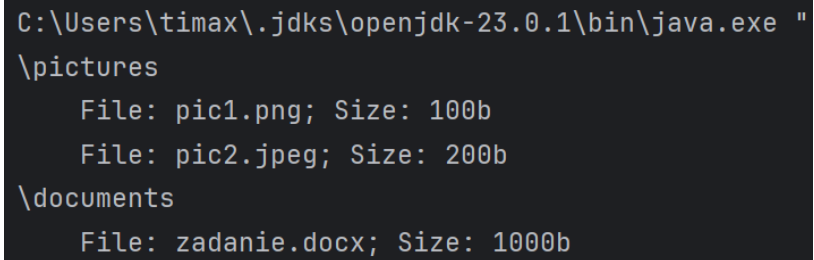
        picture.addNode(pic1);
        picture.addNode(pic2);
        docs.add(docx1);

        home.addNode(picture);
        home.addNode(docs);
        for(Node n: home.getNodes()) {
            n.printNode(0);
        }
    }
}

```

```
        for (Node m: n.getNodes()) {  
            m.printNode(4);  
        }  
    }  
}
```

Результат работы программы



```
C:\Users\timax\.jdk\openjdk-23.0.1\bin\java.exe "  
\pictures  
    File: pic1.png; Size: 100b  
    File: pic2.jpeg; Size: 200b  
\documents  
    File: zadanie.docx; Size: 1000b
```

Рисунок 13.1 Результат работы.

Вывод

Был реализован паттерн composite.

ПРАКТИЧЕСКАЯ РАБОТА 14

Теоретическое введение

Паттерны проектирования — это повторяющиеся решения типичных задач проектирования программного обеспечения. Они помогают сделать код более гибким, поддерживаемым и повторно используемым. Порождающие паттерны проектирования играют важную роль в управлении процессом создания объектов, предоставляя гибкие способы инкапсуляции создания объектов и обеспечения их корректного использования. Поведенческие паттерны (Creational Patterns) — Поведенческие паттерны проектирования описывают способы взаимодействия объектов и классов друг с другом, фокусируясь на алгоритмах и распределении обязанностей. Эти паттерны помогают обеспечить эффективное и гибкое взаимодействие между компонентами системы, упрощая коммуникацию и управление поведением объектов.

Основные задачи поведенческих паттернов: 1. Управление взаимодействием объектов: Обеспечение гибкости и надёжности при взаимодействии между объектами. Поведенческие паттерны определяют, как объекты сотрудничают и передают данные друг другу. 2. Инкапсуляция алгоритмов: позволяют отделить алгоритмы от классов, использующих их, что упрощает их замену или изменение без влияния на остальную систему. 3. Разделение обязанностей: чётко определяют роли объектов и классов в процессе выполнения операции, улучшая читаемость и поддержку кода.

Основные поведенческие паттерны

- Chain of Responsibility (Цепочка обязанностей)
- Iterator (Итератор)
- Mediator (Посредник)
- Memento (Снимок)
- Observer (Наблюдатель)
- State (Состояние)

- Strategy (Стратегия)
- Template Method (Шаблонный метод)
- Visitor (Посетитель)

Постановка задачи

В данной практической работе представлено 10 вариантов заданий. Выбор варианта осуществляется в соответствии с порядковым номером студента в списке. Если порядковый номер студента равен 1, выполняется задание варианта №1; если порядковый номер равен 10 — выполняется вариант №10. В случае, если порядковый номер превышает количество вариантов (например, 11 или более), задание выбирается циклически, начиная с варианта №1.

Вариант задания 3: Создать систему для перебора элементов коллекции (например, список студентов). Итератор должен предоставлять доступ к элементам без раскрытия структуры. Создать интерфейс `Iterator` с методами `hasNext()` и `next()`. Реализовать класс коллекции `StudentList`. Реализовать итератор для перебора студентов.

Ход выполнения работы

Реализуем `Iterator`.

Программный код

Листинг 14.1 — Реализация класса списка студентов и итератора в нем.

```
import java.util.Iterator;

public class StudentList {
    private String[] students;

    public Iterator<String> getIterator(){
        return new ArrayIterator();
    }

    private class ArrayIterator implements Iterator<String>{
        int index = 0;
        public boolean hasNext() {return index < students.length;}
        public String next(){return students[index++];}
```

```
}  
public StudentList(String[] students) {  
    this.students = students;  
}  
}
```

Вывод

Был реализован `Iterator` для перебора списков студентов.

ПРАКТИЧЕСКАЯ РАБОТА 15

Теоретическое введение

HTTP (протокол передачи гипертекста) — это основной протокол взаимодействия в сети интернет, позволяющий клиентам (например, браузерам) общаться с серверами. HTTP использует модель запрос-ответ: клиент отправляет запрос, а сервер отвечает. Рассмотрим основные компоненты HTTP-сервера. $\frac{3}{4}$ TCP-сокеты: HTTP протокол функционирует на основе TCPсоединений. Для работы с сокетами в Java используется класс `ServerSocket`. $\frac{3}{4}$ HTTP-запросы: HTTP-запрос состоит из метода (например, GET, POST), пути (урл) и HTTP-заголовков. $\frac{3}{4}$ HTTP-ответы: Сервер должен отправить клиенту ответ, который включает код статуса (200, 404 и т.д.), HTTP-заголовки и тело ответа (например, HTML). Java предоставляет широкие возможности для реализации HTTP-серверов. Ниже будет приведен листинг HTTP-сервера на Java. HTTP-сервер будет реализовывать функционал заметок. На сервере будут реализованы функции добавления заметок, удаления последней заметки, а также производится проверка на удаление несуществующей заметки.

Для запуска сервера нужно скомпилировать код в IDE.

Постановка задачи

В данной практической работе представлено 3 варианта заданий. Выбор варианта осуществляется в соответствии с порядковым номером студента в списке. Если порядковый номер студента равен 1, выполняется задание варианта №1; если порядковый номер равен 3 — выполняется вариант №3. В случае, если порядковый номер превышает количество вариантов (например, 4 или более), задание выбирается циклически, начиная с варианта №1. Также, при реализации практической работы порт, на котором будет находиться сервер должен быть двойным порядковым номером по списку. Например, если порядковый номер

студента по списку 1, то порт должен быть 11, если порядковый номер 10, то порт 1010 и так далее. Также на сервере должно быть прописано ФИО студента и его шифр.

Вариант задания 1: создать HTTP-сервер, который позволяет клиенту решать арифметические примеры. Функционал:

- Получать от клиента числа и операцию (+, -, *, /).
- Вычислять результат и возвращать клиенту.
- При ошибке (например, деление на 0) возвращать ошибку HTTP

Пример вызова:

URL: `http://localhost:порт/calculate?a=5&b=3&op=+`

Ответ: 8

Ход выполнения работы

Реализуем HTTP-сервер с возвращением результата арифметического действия

Программный код

Листинг 15.1 — Реализация метода обработки запроса

```
private static void handleCalcRequest(String path, PrintWriter out) throws
IOException{
    try{
        String query = path.split("\\?")[1];
        String[] params = query.split("&");
        int a = Integer.parseInt(params[0].split("=")[1]), b =
Integer.parseInt(params[1].split("=")[1]);
        int res = 0;
        String op = params[2].split("=")[1];
        switch (op){
            case "+":
                res = a+b;
                break;
            case "-":
                res = a-b;
                break;
            case "/":
                res = a/b;
                break;
            case "*":
                res = a*b;
                break;
            default:
```

```
        sendHttpResponse(out, 400, "<html><body><h1>Error 400:  
Invalid operation (must be + - / *</h1></body></html>");  
        return;  
    }  
    sendHttpResponse(out, 200, "<html><body><h1>Result: " + res +  
"</h1><p>" + STUDENT_INFO + "</p></body></html>");  
  
    }catch (Exception e){  
        sendHttpResponse(out, 400, "<html><body><h1>Error 400: Invalid  
request</h1></body></html>");  
    }  
}
```

Результат работы программы

Result: 8

ФИО: Туктаров Тимур Азатович Шифр: 23И0087

Рисунок 15.1 <http://localhost:2222/calculate?a=5&b=3&op=+>

Вывод

Был реализован простой HTTP-сервер способный выполнять простые арифметические действия.

ПРАКТИЧЕСКАЯ РАБОТА 16

Теоретическое введение

Spring Framework — это один из самых популярных фреймворков для разработки корпоративных приложений на языке Java. Он предоставляет мощные инструменты для упрощения разработки, управления зависимостями, обработки данных и интеграции с различными технологиями. Spring активно используется для создания веб-приложений, микросервисов и других типов программного обеспечения.

Основные принципы Spring

Инверсия управления (IoC) - Spring использует контейнер Inversion of Control для управления зависимостями между компонентами приложения. Вместо того чтобы создавать объекты и управлять их зависимостями вручную, Spring автоматически настраивает и связывает компоненты.

Внедрение зависимостей - (Dependency Injection, DI) DI позволяет передавать зависимости через конструкторы, сеттеры или поля, снижая связность компонентов и упрощая их тестирование.

Модульность и расширяемость - Spring — это набор модулей (Core, Data Access, Web, Security и др.), что позволяет использовать только те компоненты, которые нужны для проекта.

Легковесность - Несмотря на широкий функционал, Spring остается относительно легковесным, с минимальными затратами на инициализацию и управление ресурсами. Ключевые модули Spring Framework

Spring Core Основной модуль, включающий ядро фреймворка, контейнер IoC и DI. Он обеспечивает управление жизненным циклом объектов и их зависимостей.

Spring AOP (Aspect-Oriented Programming) - Модуль, позволяющий реализовывать аспекты, такие как логирование, транзакции и безопасность, не влияя на основной код

Spring Data - Модуль для упрощения работы с базами данных, включая ORM (например, Hibernate) и интеграцию с NoSQL-хранилищами.

Spring Web - Компоненты для создания веб-приложений и RESTful API. Включает поддержку MVC (Model-View-Controller).

Spring Security - Фреймворк для управления безопасностью приложения, включая аутентификацию, авторизацию и защиту от атак (CSRF, SQL-инъекции).

Spring Boot - Дополнение к Spring Framework, которое позволяет быстро создавать приложения с минимальной настройкой. Spring Boot предоставляет встроенный сервер (Tomcat/Jetty), автоконфигурацию и удобный способ управления зависимостями.

Постановка задачи

В данной практической работе вам необходимо разработать Spring – приложение с регистрацией и аутентификацией пользователя. Успешно вошедший в систему пользователь должен попасть на домашнюю страницу, на которой будет располагаться реализация задания по вариантам.

Вариант задания 2: Реализовать обработчик для формы поиска по ключевому слову.

Ход выполнения работы

Реализуем обработчик для формы поиска по ключевому слову или части слова, используя уже предоставленный код в методичке.

Программный код

Листинг 16.1 — Реализация класса AuthController в AuthController.java.

```
package com.example.demo.controller;
import java.util.ArrayList;
import java.util.List;
```

```

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestParam;

import com.example.demo.model.User;

@Controller
public class AuthController {
    private User registeredUser;
    private boolean isAuthenticated = false;

    @GetMapping("/")
    public String showRegisterPage() {
        return "register";
    }

    @PostMapping("/register")
    public String register(@RequestParam String username,
                           @RequestParam String password, Model model) {
        if (username.isEmpty() || password.isEmpty()) {
            model.addAttribute("error", "Имя пользователя и пароль не должны
быть пустыми!");
            return "register";
        }
        registeredUser = new User(username, password);
        return "redirect:/login";
    }

    @GetMapping("/login")
    public String showLoginPage() {
        return "login";
    }

    @PostMapping("/login")
    public String login(@RequestParam String username,
                        @RequestParam String password, Model model) {
        if (registeredUser == null ||
            !registeredUser.getUsername().equals(username) ||
            !registeredUser.getPassword().equals(password)) {
            model.addAttribute("error", "Неверное имя пользователя или
пароль!");
            return "login";
        }
        isAuthenticated = true;
        return "redirect:/home";
    }

    @GetMapping("/home")
    public String showHomePage(Model model) {
        if (!isAuthenticated) {
            return "redirect:/login";
        }
        model.addAttribute("username", registeredUser.getUsername());
        return "home";
    }

    @PostMapping("/home")
    public String search(@RequestParam String formType,
                         @RequestParam String query, Model model) {
        if ("search".equals(formType)) {
            List<String> results = performSearch(query);

```



```

        model.addAttribute("query", query);
        model.addAttribute("results", results);
    }
    model.addAttribute("username", registeredUser.getUsername());
    return "home";
}

private List<String> performSearch(String query) {
    List<String> allData = List.of("Лучший язык программирование", "А это
есть на Javarush?", "Стоит ли учить Java?");
    List<String> results = new ArrayList<>();
    String lowerCaseQuery = query.toLowerCase();
    for (String data : allData) {
        if (data.toLowerCase().contains(lowerCaseQuery)) {
            results.add(data);
        }
    }
    return results;
}
}

```

Листинг 16.2 — Реализация класса User в User.java.

```

package com.example.demo.model;
public class User {
    private String username;
    private String password;

    public User() {}
    public User(String username, String password) {
        this.username = username;
        this.password = password;
    }

    public String getUsername() {
        return username;
    }
    public void setUsername(String username) {
        this.username = username;
    }
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }
}

```

Листинг 16.3 — Изменения в home.html.

```

<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Home</title>
</head>
<body>
<h1>Добро пожаловать, <span th:text="${username}">Пользователь</span></h1>

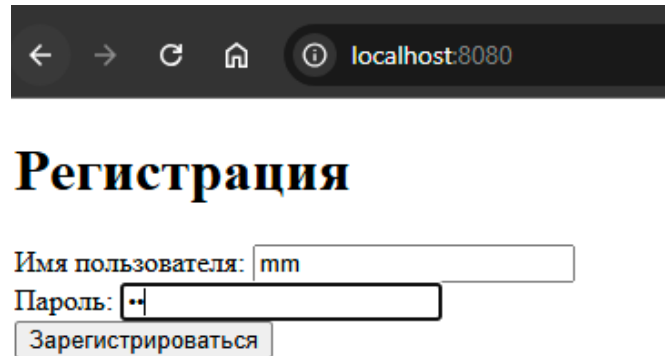
<h2>Поиск</h2>
<form th:action="@{/home}" method="post">
    <input type="hidden" name="formType" value="search" />
    <label>Ключевое слово:</label>
    <input type="text" name="query" required />
    <button type="submit">Поиск</button>

```

```
</form>

<h3 th:if="${query}">Результаты поиска для: <span
th:text="${query}"></span></h3>
<ul th:if="${results != null}" th:each="result : ${results}">
    <li th:text="${result}"></li>
</ul>
</body>
</html>
```

Результат работы программы



← → ↻ 🏠 ⓘ localhost:8080

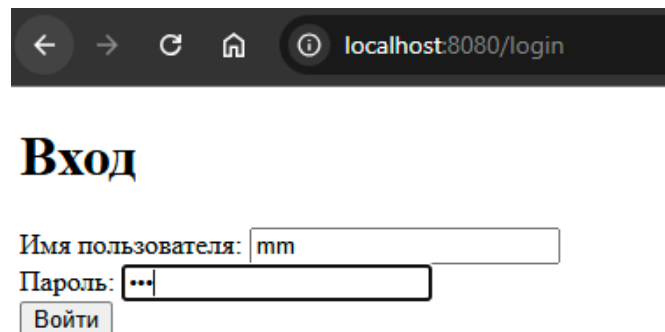
Регистрация

Имя пользователя: mm

Пароль: ..

Зарегистрироваться

Рисунок 16.1 Результат на <http://localhost:8080/>.



← → ↻ 🏠 ⓘ localhost:8080/login

Вход

Имя пользователя: mm

Пароль: ...

Войти

Рисунок 16.2 Результат на <http://localhost:8080/>.

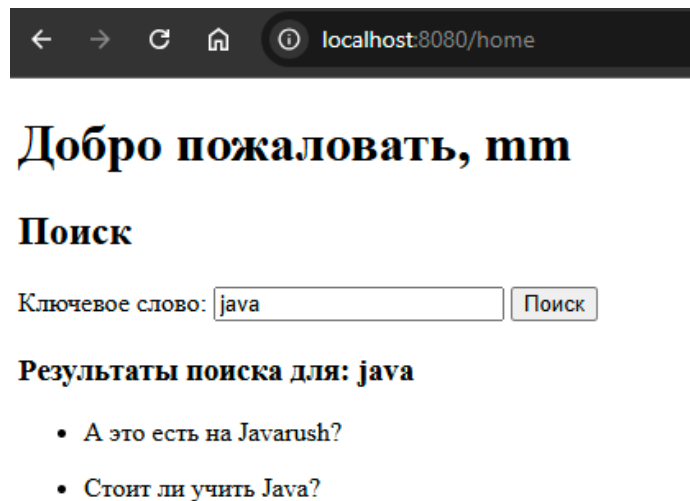


Рисунок 16.3 Результат на <http://localhost:8080/home>.

Вывод

Был реализован сервер с регистрацией, входом, а также поиском по ключевому слову или части слова.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. методическое пособие Комарова П.С - URL: <https://drive.google.com/file/d/1Gqg8w1jz9Cf-yVIuCd1GwbRlZBVOSvO4/view?pli=1>.
2. Гетц Б. - Java Concurrency на практике.pdf - URL: <https://online-edu.mirea.ru/mod/folder/view.php?id=647839#:~:text=%D0%93%D0%B5%D1%82%D1%86%20%D0%91.%20%2D%20Java%20Concurrency%D0%BD%D0%B0%20%D0%BF%D1%80%D0%B0%D0%BA%D1%82%D0%B8%D0%BA%D0%B5.pdf>
3. Лафоре Р. - Структуры данных и алгоритмы в Java, 2-е издание.pdf. -URL: https://online-edu.mirea.ru/pluginfile.php/1489776/mod_folder/content/0/%D0%9B%D0%B0%D1%84%D0%BE%D1%80%D0%B5%20%D0%A0.%20-%20%D0%A1%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D1%8B%20%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85%20%D0%B8%20%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC%D1%8B%20%D0%B2%20Java%2C%20-%D0%B5%20%D0%B8%D0%B7%D0%B4%D0%B0%D0%BD%D0%B8%D0%B5.pdf?forcedownload=1
4. Спилкэ Л. - JAVA устранение проблем. Чтение, отладка и оптимизация JVM-приложений
5. Документация к Java – Url: <https://docs.oracle.com/en/java/javase/>