**1.    Описать класс «поезд», содержащий следующие закрытые поля: название пункта назначения; номер поезда (может содержать буквы и цифры); время отправления. Предусмотреть свойства для получения состояния объекта. Описать класс «вокзал», содержащий закрытый массив поездов. Обеспечить следующие возможности: вывод информации о поезде по номеру с помощью индекса; вывод информации о поездах, отправляющихся после введенного с клавиатуры времени; перегруженную операцию сравнения, выполняющую сравнение времени отправления двух поездов; вывод информации о поездах, отправляющихся в заданный пункт назначения. Информация должна быть отсортирована по времени отправления. Написать программу, демонстрирующую все разработанные элементы классов.**

Класс Train

class Train

{

private string destination;

private string trainNumber;

private TimeSpan departureTime;

// Конструктор

public Train(string destination, string trainNumber, TimeSpan departureTime)

{

this.destination = destination;

this.trainNumber = trainNumber;

this.departureTime = departureTime;

}

// Свойства для получения состояния объекта

public string Destination => destination;

public string TrainNumber => trainNumber;

public TimeSpan DepartureTime => departureTime;

// Перегрузка оператора сравнения (по времени)

public static bool operator >(Train a, Train b)

{

return a.departureTime > b.departureTime;

}

public static bool operator <(Train a, Train b)

{

return a.departureTime < b.departureTime;

}

// Печать информации

public void PrintInfo()

{

Console.WriteLine($"Поезд № {trainNumber}, пункт назначения: {destination}, отправление: {departureTime}");

}

}

Класс Station

class Station

{

private Train[] trains;

public Station(Train[] trains)

{

this.trains = trains;

}

// Индексатор по номеру поезда

public Train this[string trainNumber]

{

get

{

foreach (var train in trains)

if (train.TrainNumber == trainNumber)

return train;

return null;

}

}

// Поезда после заданного времени

public void PrintTrainsAfter(TimeSpan time)

{

foreach (var train in trains)

if (train.DepartureTime > time)

train.PrintInfo();

}

// Поезда по пункту назначения (отсортированные по времени)

public void PrintTrainsToDestination(string destination)

{

var filtered = trains.Where(t => t.Destination == destination)

.OrderBy(t => t.DepartureTime);

foreach (var train in filtered)

train.PrintInfo();

}

}

Программа

class Program

{

static void Main()

{

Train[] trains = {

new Train("Москва", "А123", new TimeSpan(14, 30, 0)),

new Train("Сочи", "B456", new TimeSpan(16, 45, 0)),

new Train("Москва", "C789", new TimeSpan(12, 15, 0))

};

Station station = new Station(trains);

Console.WriteLine("Поезд B456:");

station["B456"]?.PrintInfo();

Console.WriteLine("\nПоезда после 13:00:");

station.PrintTrainsAfter(new TimeSpan(13, 0, 0));

Console.WriteLine("\nПоезда в Москву:");

station.PrintTrainsToDestination("Москва");

}

}

**2.    Описать класс «товар», содержащий следующие закрытые поля: название товара; название магазина, в котором продается товар; стоимость товара в рублях. Предусмотреть свойства для получения состояния объекта. Описать класс «склад», содержащий закрытый массив товаров. Обеспечить следующие возможности: вывод информации о товаре по номеру с помощью индекса; вывод на экран информации о товаре, название которого введено с клавиатуры; если таких товаров нет, выдать соответствующее сообщение; сортировку товаров по названию магазина, по наименованию и по цене; перегруженную операцию сложения товаров, выполняющую сложение их цен. Написать программу, демонстрирующую все разработанные элементы классов.**

Класс Product (Товар):

public class Product {

private String productName;

private String storeName;

private double price;

public Product(String productName, String storeName, double price) {

this.productName = productName;

this.storeName = storeName;

this.price = price;

}

// Геттеры (свойства для получения состояния объекта)

public String getProductName() {

return productName;

}

public String getStoreName() {

return storeName;

}

public double getPrice() {

return price;

}

// Метод для вывода информации о товаре

public void printInfo() {

System.out.println("Товар: " + productName + ", Магазин: " + storeName + ", Цена: " + price + " руб.");

}

// "Сложение" двух товаров по цене

public static double sumPrices(Product a, Product b) {

return a.price + b.price;

}

}

Класс Warehouse (Склад):

java

КопироватьРедактировать

import java.util.\*;

public class Warehouse {

private Product[] products;

public Warehouse(Product[] products) {

this.products = products;

}

// Вывод по индексу

public void printByIndex(int index) {

if (index >= 0 && index < products.length) {

products[index].printInfo();

} else {

System.out.println("Ошибка: индекс вне диапазона.");

}

}

// Поиск по названию товара

public void printByProductName(String name) {

boolean found = false;

for (Product product : products) {

if (product.getProductName().equalsIgnoreCase(name)) {

product.printInfo();

found = true;

}

}

if (!found) {

System.out.println("Товары с названием \"" + name + "\" не найдены.");

}

}

// Сортировка по магазину

public void sortByStoreName() {

Arrays.sort(products, Comparator.comparing(Product::getStoreName));

}

// Сортировка по названию товара

public void sortByProductName() {

Arrays.sort(products, Comparator.comparing(Product::getProductName));

}

// Сортировка по цене

public void sortByPrice() {

Arrays.sort(products, Comparator.comparingDouble(Product::getPrice));

}

// Печать всех товаров

public void printAll() {

for (Product product : products) {

product.printInfo();

}

}

}

Класс Main для демонстрации:

import java.util.Scanner;

public class Main {

public static void main(String[] args) {

Product[] productList = {

new Product("Хлеб", "Магнит", 40.0),

new Product("Молоко", "Пятёрочка", 65.5),

new Product("Сыр", "Лента", 230.0),

new Product("Хлеб", "Пятёрочка", 42.0)

};

Warehouse warehouse = new Warehouse(productList);

Scanner scanner = new Scanner(System.in);

System.out.println("=== Все товары на складе ===");

warehouse.printAll();

System.out.println("\n=== Поиск товара по индексу ===");

warehouse.printByIndex(2);

System.out.println("\n=== Поиск товара по названию ===");

System.out.print("Введите название товара: ");

String name = scanner.nextLine();

warehouse.printByProductName(name);

System.out.println("\n=== Сортировка по магазину ===");

warehouse.sortByStoreName();

warehouse.printAll();

System.out.println("\n=== Сортировка по названию ===");

warehouse.sortByProductName();

warehouse.printAll();

System.out.println("\n=== Сортировка по цене ===");

warehouse.sortByPrice();

warehouse.printAll();

System.out.println("\n=== Сложение цен двух товаров ===");

double total = Product.sumPrices(productList[0], productList[1]);

System.out.println("Сумма цен двух товаров: " + total + " руб.");

}

}

**3. Описать класс «самолет», содержащий следующие закрытые поля: название пункта назначения; шестизначный номер рейса; время отправления. Предусмотреть свойства для получения состояния объекта. Описать класс «аэропорт», содержащий закрытый массив самолетов. Обеспечить следующие возможности: вывод информации о самолете по номеру рейса с помощью индекса; вывод информации о самолетах, отправляющихся в течение часа после введенного с клавиатуры времени; вывод информации о самолетах, отправляющихся в заданный пункт назначения; перегруженную операцию сравнения, выполняющую сравнение времени отправления двух самолетов. Информация должна быть отсортирована по времени отправления. Написать программу, демонстрирующую все разработанные элементы классов.**

// Класс Самолет

class Airplane {

private String destination;

private int flightNumber;

private LocalTime departureTime;

// Конструктор

public Airplane(String destination, int flightNumber, LocalTime departureTime) {

this.destination = destination;

this.flightNumber = flightNumber;

this.departureTime = departureTime;

}

// Свойства для получения состояния объекта

public String getDestination() {

return destination;

}

public int getFlightNumber() {

return flightNumber;

}

public LocalTime getDepartureTime() {

return departureTime;

}

// Переопределение метода toString для удобного вывода информации

@Override

public String toString() {

DateTimeFormatter formatter = DateTimeFormatter.ofPattern("HH:mm");

return "Рейс №" + flightNumber + " в " + destination +

", время вылета: " + departureTime.format(formatter);

}

}

// Класс Аэропорт

class Airport {

private List<Airplane> airplanes;

// Конструктор

public Airport() {

this.airplanes = new ArrayList<>();

}

// Добавление самолета в аэропорт

public void addAirplane(Airplane airplane) {

airplanes.add(airplane);

}

// Вывод информации о самолете по номеру рейса

public void printFlightInfo(int flightNumber) {

boolean found = false;

for (Airplane airplane : airplanes) {

if (airplane.getFlightNumber() == flightNumber) {

System.out.println(airplane);

found = true;

break;

}

}

if (!found) {

System.out.println("Рейс с номером " + flightNumber + " не найден.");

}

}

// Вывод информации о самолетах, отправляющихся в течение часа после введенного времени

public void printFlightsWithinHourAfter(LocalTime time) {

List<Airplane> result = new ArrayList<>();

LocalTime endTime = time.plusHours(1);

for (Airplane airplane : airplanes) {

LocalTime depTime = airplane.getDepartureTime();

if (depTime.isAfter(time) && (depTime.isBefore(endTime) || depTime.equals(endTime))) {

result.add(airplane);

}

}

// Сортировка по времени отправления

Collections.sort(result, Comparator.comparing(Airplane::getDepartureTime));

if (result.isEmpty()) {

System.out.println("Нет рейсов в течение часа после " + time);

} else {

System.out.println("Рейсы в течение часа после " + time + ":");

for (Airplane airplane : result) {

System.out.println(airplane);

}

}

}

// Вывод информации о самолетах, отправляющихся в заданный пункт назначения

public void printFlightsToDestination(String destination) {

List<Airplane> result = new ArrayList<>();

for (Airplane airplane : airplanes) {

if (airplane.getDestination().equalsIgnoreCase(destination)) {

result.add(airplane);

}

}

// Сортировка по времени отправления

Collections.sort(result, Comparator.comparing(Airplane::getDepartureTime));

if (result.isEmpty()) {

System.out.println("Нет рейсов в пункт назначения " + destination);

} else {

System.out.println("Рейсы в " + destination + ":");

for (Airplane airplane : result) {

System.out.println(airplane);

}

}

}

// Перегруженная операция сравнения времени отправления двух самолетов

public static int compareDepartureTimes(Airplane a1, Airplane a2) {

return a1.getDepartureTime().compareTo(a2.getDepartureTime());

}

}

// Демонстрационная программа

public class AirportDemo {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

DateTimeFormatter timeFormatter = DateTimeFormatter.ofPattern("HH:mm");

// Создаем аэропорт и добавляем несколько рейсов

Airport airport = new Airport();

airport.addAirplane(new Airplane("Москва", 123456, LocalTime.of(10, 30)));

airport.addAirplane(new Airplane("Санкт-Петербург", 654321, LocalTime.of(11, 15)));

airport.addAirplane(new Airplane("Новосибирск", 456789, LocalTime.of(9, 45)));

airport.addAirplane(new Airplane("Москва", 987654, LocalTime.of(12, 0)));

airport.addAirplane(new Airplane("Сочи", 111222, LocalTime.of(14, 30)));

airport.addAirplane(new Airplane("Москва", 333444, LocalTime.of(10, 45)));

// Меню для демонстрации функционала

while (true) {

System.out.println("\nМеню:");

System.out.println("1. Информация о рейсе по номеру");

System.out.println("2. Рейсы в течение часа после указанного времени");

System.out.println("3. Рейсы в указанный пункт назначения");

System.out.println("4. Сравнить время отправления двух рейсов");

System.out.println("0. Выход");

System.out.print("Выберите действие: ");

int choice = scanner.nextInt();

scanner.nextLine(); // очистка буфера

switch (choice) {

case 1:

System.out.print("Введите номер рейса (6 цифр): ");

int flightNumber = scanner.nextInt();

scanner.nextLine();

airport.printFlightInfo(flightNumber);

break;

case 2:

System.out.print("Введите время (HH:mm): ");

String timeStr = scanner.nextLine();

LocalTime time = LocalTime.parse(timeStr, timeFormatter);

airport.printFlightsWithinHourAfter(time);

break;

case 3:

System.out.print("Введите пункт назначения: ");

String destination = scanner.nextLine();

airport.printFlightsToDestination(destination);

break;

case 4:

System.out.print("Введите номер первого рейса: ");

int flight1 = scanner.nextInt();

System.out.print("Введите номер второго рейса: ");

int flight2 = scanner.nextInt();

scanner.nextLine();

Airplane a1 = null, a2 = null;

// Находим оба рейса

for (Airplane airplane : airport.getAirplanes()) {

if (airplane.getFlightNumber() == flight1) a1 = airplane;

if (airplane.getFlightNumber() == flight2) a2 = airplane;

}

if (a1 != null && a2 != null) {

int comparison = Airport.compareDepartureTimes(a1, a2);

if (comparison < 0) {

System.out.println("Рейс " + flight1 + " вылетает раньше рейса " + flight2);

} else if (comparison > 0) {

System.out.println("Рейс " + flight1 + " вылетает позже рейса " + flight2);

} else {

System.out.println("Оба рейса вылетают в одно время");

}

} else {

System.out.println("Один или оба рейса не найдены");

}

break;

case 0:

System.out.println("Выход из программы.");

scanner.close();

return;

default:

System.out.println("Неверный выбор. Попробуйте снова.");

}

}

}

}

**4.Описать класс «запись», содержащий следующие закрытые поля: фамилия, имя; номер телефона; дата рождения (массив из трех чисел). Предусмотреть свойства для получения состояния объекта. Описать класс «записная книжка», содержащий закрытый массив записей. Обеспечить следующие возможности: вывод на экран информации о человеке, номер телефона которого введен с клавиатуры; если такого нет, выдать на дисплей соответствующее сообщение; поиск людей, день рождения которых сегодня или в заданный день; поиск людей, день рождения которых будет на следующей неделе; поиск людей, номер телефона которых начинается на три заданных цифры. Написать программу, демонстрирующую все разработанные элементы классов.**

// Класс Запись

class Record {

private String lastName;

private String firstName;

private String phoneNumber;

private int[] birthDate; // массив [день, месяц, год]

// Конструктор

public Record(String lastName, String firstName, String phoneNumber, int[] birthDate) {

this.lastName = lastName;

this.firstName = firstName;

this.phoneNumber = phoneNumber;

this.birthDate = birthDate;

}

// Свойства для получения состояния объекта

public String getLastName() {

return lastName;

}

public String getFirstName() {

return firstName;

}

public String getPhoneNumber() {

return phoneNumber;

}

public int[] getBirthDate() {

return birthDate;

}

// Получение даты рождения в виде строки

public String getBirthDateString() {

return String.format("%02d.%02d.%04d", birthDate[0], birthDate[1], birthDate[2]);

}

// Переопределение метода toString для удобного вывода информации

@Override

public String toString() {

return lastName + " " + firstName +

", тел.: " + phoneNumber +

", дата рождения: " + getBirthDateString();

}

}

// Класс Записная книжка

class AddressBook {

private List<Record> records;

// Конструктор

public AddressBook() {

this.records = new ArrayList<>();

}

// Добавление записи в книжку

public void addRecord(Record record) {

records.add(record);

}

// Поиск по номеру телефона

public void searchByPhoneNumber(String phoneNumber) {

boolean found = false;

for (Record record : records) {

if (record.getPhoneNumber().equals(phoneNumber)) {

System.out.println(record);

found = true;

break;

}

}

if (!found) {

System.out.println("Человек с номером телефона " + phoneNumber + " не найден.");

}

}

// Поиск по дате рождения (день и месяц)

public void searchByBirthDate(int day, int month) {

List<Record> result = new ArrayList<>();

for (Record record : records) {

int[] birthDate = record.getBirthDate();

if (birthDate[0] == day && birthDate[1] == month) {

result.add(record);

}

}

if (result.isEmpty()) {

System.out.println("Нет людей с днем рождения " + day + "." + month);

} else {

System.out.println("Люди с днем рождения " + day + "." + month + ":");

for (Record record : result) {

System.out.println(record);

}

}

}

// Поиск людей, у которых день рождения на следующей неделе

public void searchBirthdaysNextWeek() {

LocalDate today = LocalDate.now();

LocalDate nextWeek = today.plusWeeks(1);

List<Record> result = new ArrayList<>();

System.out.println("Сегодня: " + today.format(DateTimeFormatter.ofPattern("dd.MM.yyyy")));

System.out.println("Дни рождения на следующей неделе:");

for (Record record : records) {

int[] birthDate = record.getBirthDate();

LocalDate thisYearBirthday = LocalDate.of(today.getYear(), birthDate[1], birthDate[0]);

if (thisYearBirthday.isAfter(today) &&

(thisYearBirthday.isBefore(nextWeek) || thisYearBirthday.isEqual(nextWeek))) {

result.add(record);

}

}

if (result.isEmpty()) {

System.out.println("На следующей неделе дней рождения нет.");

} else {

for (Record record : result) {

System.out.println(record + " (будет " +

record.getBirthDate()[0] + "." + record.getBirthDate()[1] + ")");

}

}

}

// Поиск по первым трем цифрам телефона

public void searchByPhonePrefix(String prefix) {

List<Record> result = new ArrayList<>();

for (Record record : records) {

if (record.getPhoneNumber().startsWith(prefix)) {

result.add(record);

}

}

if (result.isEmpty()) {

System.out.println("Нет людей с номерами, начинающимися на " + prefix);

} else {

System.out.println("Люди с номерами, начинающимися на " + prefix + ":");

for (Record record : result) {

System.out.println(record);

}

}

}

}

// Демонстрационная программа

public class AddressBookDemo {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Создаем записную книжку и добавляем несколько записей

AddressBook addressBook = new AddressBook();

addressBook.addRecord(new Record("Иванов", "Иван", "1234567", new int[]{15, 5, 1990}));

addressBook.addRecord(new Record("Петров", "Петр", "2345678", new int[]{20, 6, 1985}));

addressBook.addRecord(new Record("Сидорова", "Мария", "3456789", new int[]{10, 5, 1995}));

addressBook.addRecord(new Record("Кузнецов", "Алексей", "4567890", new int[]{25, 12, 1980}));

addressBook.addRecord(new Record("Смирнова", "Ольга", "5678901", new int[]{5, 5, 1992}));

addressBook.addRecord(new Record("Васильев", "Дмитрий", "6789012", new int[]{12, 5, 1988}));

// Меню для демонстрации функционала

while (true) {

System.out.println("\nМеню записной книжки:");

System.out.println("1. Поиск по номеру телефона");

System.out.println("2. Поиск по дате рождения");

System.out.println("3. Показать дни рождения на следующей неделе");

System.out.println("4. Поиск по первым трем цифрам телефона");

System.out.println("0. Выход");

System.out.print("Выберите действие: ");

int choice = scanner.nextInt();

scanner.nextLine(); // очистка буфера

switch (choice) {

case 1:

System.out.print("Введите номер телефона: ");

String phoneNumber = scanner.nextLine();

addressBook.searchByPhoneNumber(phoneNumber);

break;

case 2:

System.out.print("Введите день рождения (день месяц через пробел): ");

int day = scanner.nextInt();

int month = scanner.nextInt();

scanner.nextLine();

addressBook.searchByBirthDate(day, month);

break;

case 3:

addressBook.searchBirthdaysNextWeek();

break;

case 4:

System.out.print("Введите первые 3 цифры номера: ");

String prefix = scanner.nextLine();

addressBook.searchByPhonePrefix(prefix);

break;

case 0:

System.out.println("Выход из программы.");

scanner.close();

return;

default:

System.out.println("Неверный выбор. Попробуйте снова.");

}

}

}

}

**5. Описать базовый класс «Строка». Обязательные поля класса: поле для хранения символов строки; значение типа word для хранения длины строки в байтах. Реализовать обязательные методы следующего назначения: конструктор без параметров; конструктор, принимающий в качестве параметра строковый литерал; конструктор, принимающий в качестве параметра символ; метод получения длины строки; метод очистки строки. Описать производный от «Строка» класс «Комплексное число». Строки данного класса состоят из двух полей, разделенных символом i. Первое поле задает значение действительной части числа, второе – значение мнимой. Каждое из полей может содержать только символы десятичных цифр и символы + и -, задающие знак числа. Символы + и – могут находиться только в первой позиции числа, причем символ + может отсутствовать, в этом случае число считается положительным. Для класса «Комплексное\_число» определить следующие методы: проверка на равенство; сложение чисел; умножение чисел.**

public class Main {

public static void main(String[] args) {

// Демонстрация работы со строками

Stroka s1 = new Stroka();

Stroka s2 = new Stroka("Hello");

Stroka s3 = new Stroka('A');

System.out.println("s1: " + s1 + ", длина: " + s1.getLength());

System.out.println("s2: " + s2 + ", длина: " + s2.getLength());

System.out.println("s3: " + s3 + ", длина: " + s3.getLength());

s1.clear();

System.out.println("После очистки s1: " + s1);

// Демонстрация работы с комплексными числами

ComplexNumber cn1 = new ComplexNumber("3+5i");

ComplexNumber cn2 = new ComplexNumber("-2-4i");

ComplexNumber cn3 = new ComplexNumber("1+i");

ComplexNumber cn4 = new ComplexNumber("3+5i");

System.out.println("\nКомплексные числа:");

System.out.println("cn1: " + cn1);

System.out.println("cn2: " + cn2);

System.out.println("cn3: " + cn3);

System.out.println("\ncn1 equals cn2: " + cn1.equals(cn2));

System.out.println("cn1 equals cn4: " + cn1.equals(cn4));

ComplexNumber sum = cn1.add(cn2);

System.out.println("\nСумма cn1 + cn2: " + sum);

ComplexNumber product = cn1.multiply(cn2);

System.out.println("Произведение cn1 \* cn2: " + product);

}

}

// Базовый класс Строка

class Stroka {

protected char[] characters;

protected int length; // длина строки в байтах (в Java char = 2 байта)

// Конструктор без параметров

public Stroka() {

this.characters = new char[0];

this.length = 0;

}

// Конструктор, принимающий строковый литерал

public Stroka(String str) {

this.characters = str.toCharArray();

this.length = str.length() \* 2; // каждый char = 2 байта

}

// Конструктор, принимающий символ

public Stroka(char c) {

this.characters = new char[]{c};

this.length = 2; // 1 char = 2 байта

}

// Метод получения длины строки в байтах

public int getLength() {

return this.length;

}

// Метод очистки строки

public void clear() {

this.characters = new char[0];

this.length = 0;

}

// Переопределение метода toString для вывода строки

@Override

public String toString() {

return new String(characters);

}

}

// Производный класс Комплексное число

class ComplexNumber extends Stroka {

private double real;

private double imaginary;

// Конструктор

public ComplexNumber(String str) {

super(str);

parseComplexNumber(str);

}

// Парсинг строки в комплексное число

private void parseComplexNumber(String str) {

String[] parts = str.split("i");

if (parts.length != 1) {

String numStr = parts[0].trim();

// Обработка действительной части

int plusIndex = numStr.lastIndexOf('+');

int minusIndex = numStr.lastIndexOf('-');

int splitIndex = Math.max(plusIndex, minusIndex);

if (splitIndex <= 0) {

// Только действительная или только мнимая часть

if (numStr.contains("+") || numStr.contains("-")) {

// Только мнимая часть

this.real = 0;

this.imaginary = parsePart(numStr);

} else {

// Только действительная часть

this.real = parsePart(numStr);

this.imaginary = 0;

}

} else {

// И действительная, и мнимая части

this.real = parsePart(numStr.substring(0, splitIndex));

this.imaginary = parsePart(numStr.substring(splitIndex));

}

} else {

// Нет 'i' - только действительная часть

this.real = parsePart(str);

this.imaginary = 0;

}

}

// Парсинг части числа

private double parsePart(String part) {

part = part.trim();

if (part.isEmpty()) {

return 1; // Для случая, когда просто "i" (подразумевается 1i)

}

if (part.equals("+")) {

return 1;

}

if (part.equals("-")) {

return -1;

}

return Double.parseDouble(part);

}

// Проверка на равенство

@Override

public boolean equals(Object obj) {

if (this == obj) return true;

if (obj == null || getClass() != obj.getClass()) return false;

ComplexNumber other = (ComplexNumber) obj;

return Double.compare(other.real, real) == 0 &&

Double.compare(other.imaginary, imaginary) == 0;

}

// Сложение комплексных чисел

public ComplexNumber add(ComplexNumber other) {

double newReal = this.real + other.real;

double newImaginary = this.imaginary + other.imaginary;

return createFromParts(newReal, newImaginary);

}

// Умножение комплексных чисел

public ComplexNumber multiply(ComplexNumber other) {

double newReal = this.real \* other.real - this.imaginary \* other.imaginary;

double newImaginary = this.real \* other.imaginary + this.imaginary \* other.real;

return createFromParts(newReal, newImaginary);

}

// Создание комплексного числа из частей

private ComplexNumber createFromParts(double real, double imaginary) {

String str;

if (imaginary >= 0) {

str = real + "+" + imaginary + "i";

} else {

str = real + "" + imaginary + "i"; // минус уже в числе

}

return new ComplexNumber(str);

}

// Переопределение метода toString

@Override

public String toString() {

if (imaginary >= 0) {

return real + " + " + imaginary + "i";

} else {

return real + " - " + (-imaginary) + "i";

}

}

// Геттеры для частей числа

public double getReal() {

return real;

}

public double getImaginary() {

return imaginary;

}

}