**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 Poezd**

java

КопироватьРедактировать

import java.time.LocalTime;

import java.time.format.DateTimeFormatter;

public class Poezd implements Comparable<Poezd> {

private String destination;

private String trainNumber;

private LocalTime departureTime;

public Poezd(String destination, String trainNumber, String timeStr) {

this.destination = destination;

this.trainNumber = trainNumber;

this.departureTime = LocalTime.parse(timeStr, DateTimeFormatter.ofPattern("HH:mm"));

}

// Свойства (геттеры)

public String getDestination() {

return destination;

}

public String getTrainNumber() {

return trainNumber;

}

public LocalTime getDepartureTime() {

return departureTime;

}

// Сравнение поездов по времени

@Override

public int compareTo(Poezd other) {

return this.departureTime.compareTo(other.departureTime);

}

// Переопределение toString для вывода

@Override

public String toString() {

return String.format("Поезд №%s -> %s, время отправления: %s",

trainNumber, destination, departureTime.toString());

}

}

**Класс Vokzal**

java

КопироватьРедактировать

import java.time.LocalTime;

import java.time.format.DateTimeFormatter;

import java.util.\*;

public class Vokzal {

private Poezd[] trains;

public Vokzal(Poezd[] trains) {

this.trains = trains;

}

// Индексатор — поиск по номеру поезда

public Poezd getByTrainNumber(String number) {

for (Poezd p : trains) {

if (p.getTrainNumber().equalsIgnoreCase(number)) {

return p;

}

}

return null;

}

// Поезда, отправляющиеся после заданного времени

public List<Poezd> getAfterTime(String timeStr) {

LocalTime time = LocalTime.parse(timeStr, DateTimeFormatter.ofPattern("HH:mm"));

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

for (Poezd p : trains) {

if (p.getDepartureTime().isAfter(time)) {

result.add(p);

}

}

return result;

}

// Поезда по пункту назначения, отсортированные по времени

public List<Poezd> getByDestination(String destination) {

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

for (Poezd p : trains) {

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

result.add(p);

}

}

Collections.sort(result);

return result;

}

}

**Программа демонстрации**

java

КопироватьРедактировать

import java.util.List;

import java.util.Scanner;

public class Main {

public static void main(String[] args) {

Poezd[] allTrains = {

new Poezd("Москва", "A123", "08:30"),

new Poezd("Санкт-Петербург", "B456", "11:45"),

new Poezd("Москва", "C789", "14:10"),

new Poezd("Казань", "D001", "06:55"),

new Poezd("Сочи", "E333", "20:00")

};

Vokzal vokzal = new Vokzal(allTrains);

// Поиск по номеру

System.out.println("Информация о поезде с номером B456:");

Poezd train = vokzal.getByTrainNumber("B456");

if (train != null)

System.out.println(train);

else

System.out.println("Поезд не найден.");

// Поезда после определенного времени

Scanner scanner = new Scanner(System.in);

System.out.print("\nВведите время в формате HH:mm, чтобы найти поезда после этого времени: ");

String timeInput = scanner.nextLine();

List<Poezd> afterTime = vokzal.getAfterTime(timeInput);

System.out.println("Поезда после " + timeInput + ":");

for (Poezd p : afterTime) {

System.out.println(p);

}

// Поезда по пункту назначения

System.out.print("\nВведите пункт назначения: ");

String dest = scanner.nextLine();

List<Poezd> byDest = vokzal.getByDestination(dest);

System.out.println("Поезда в " + dest + ":");

for (Poezd p : byDest) {

System.out.println(p);

}

// Демонстрация перегруженной операции сравнения

System.out.println("\nСравнение времени отправления поездов A123 и C789:");

Poezd p1 = vokzal.getByTrainNumber("A123");

Poezd p2 = vokzal.getByTrainNumber("C789");

int cmp = p1.compareTo(p2);

if (cmp < 0)

System.out.println(p1.getTrainNumber() + " отправляется раньше " + p2.getTrainNumber());

else if (cmp > 0)

System.out.println(p2.getTrainNumber() + " отправляется раньше " + p1.getTrainNumber());

else

System.out.println("Поезда отправляются одновременно.");

}

}

**Что реализовано:**

* Класс Poezd: скрытые поля, свойства (геттеры), парсинг времени, переопределение toString, метод compareTo() для сравнения.
* Класс Vokzal: хранение массива поездов, поиск по номеру, фильтрация по времени и пункту назначения, сортировка.
* Демонстрация работы: поиск, фильтрация, сравнение.

Код совместим с Java 8+, поскольку использует LocalTime.

**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.**

Ниже приведена реализация на Java базового класса Stroka и производного от него класса KompleksnoeChislo в соответствии с заданием. Используются только необходимые поля и методы.

### Класс Stroka

public class Stroka {

protected char[] chars;

protected short length; // Аналог типа word (2 байта)

// Конструктор без параметров

public Stroka() {

this.chars = new char[0];

this.length = 0;

}

// Конструктор с строковым литералом

public Stroka(String input) {

this.chars = input.toCharArray();

this.length = (short) (input.length() \* 2); // char в Java — 2 байта

}

// Конструктор с одним символом

public Stroka(char c) {

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

this.length = 2; // один символ = 2 байта

}

// Метод получения длины строки в байтах

public short getLength() {

return length;

}

// Метод очистки строки

public void clear() {

this.chars = new char[0];

this.length = 0;

}

// Метод для получения строки

public String getString() {

return new String(chars);

}

}

### Класс KompleksnoeChislo

public class KompleksnoeChislo extends Stroka {

private int real;

private int imag;

// Конструктор из строкового представления комплексного числа

public KompleksnoeChislo(String complexStr) {

super(complexStr);

parseComplex(complexStr);

}

// Разбор строки вида "+3-5i" или "4+2i"

private void parseComplex(String str) {

str = str.replace(" ", "");

int indexOfI = str.indexOf('i');

if (indexOfI == -1) {

throw new IllegalArgumentException("Отсутствует символ 'i'");

}

String realPart = str.substring(0, indexOfI);

String imagPart = "";

// Найдём позицию второго знака: + или - после первого символа

for (int i = 1; i < realPart.length(); i++) {

if (realPart.charAt(i) == '+' || realPart.charAt(i) == '-') {

imagPart = realPart.substring(i);

realPart = realPart.substring(0, i);

break;

}

}

if (imagPart.isEmpty()) {

imagPart = "0";

}

this.real = Integer.parseInt(realPart);

this.imag = Integer.parseInt(imagPart);

}

// Проверка на равенство

public boolean equals(KompleksnoeChislo other) {

return this.real == other.real && this.imag == other.imag;

}

// Сложение комплексных чисел

public KompleksnoeChislo add(KompleksnoeChislo other) {

int newReal = this.real + other.real;

int newImag = this.imag + other.imag;

return new KompleksnoeChislo(formatComplex(newReal, newImag));

}

// Умножение комплексных чисел

public KompleksnoeChislo multiply(KompleksnoeChislo other) {

int newReal = this.real \* other.real - this.imag \* other.imag;

int newImag = this.real \* other.imag + this.imag \* other.real;

return new KompleksnoeChislo(formatComplex(newReal, newImag));

}

// Форматирование в строку

private String formatComplex(int real, int imag) {

String imagStr = (imag >= 0 ? "+" : "") + imag;

return real + imagStr + "i";

}

// Переопределение toString

@Override

public String toString() {

return formatComplex(real, imag);

}

}

### Пример использования:

public class Main {

public static void main(String[] args) {

KompleksnoeChislo c1 = new KompleksnoeChislo("3+4i");

KompleksnoeChislo c2 = new KompleksnoeChislo("-1-2i");

KompleksnoeChislo sum = c1.add(c2);

KompleksnoeChislo product = c1.multiply(c2);

System.out.println("Сумма: " + sum); // 2+2i

System.out.println("Произведение: " + product); // 5-10i

System.out.println("Равны: " + c1.equals(c2)); // false

}

}