|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ **Информатика и системы управления**

КАФЕДРА **Компьютерные системы и сети (ИУ6)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.04.01 Информатика и вычислительная техника**

МАГИСТЕРСКАЯ ПРОГРАММА **09.04.01/07 Интеллектуальные системы анализа, обработки и интерпретации больших данных**

**Отчет**

|  |  |
| --- | --- |
| **по лабораторной работе №** | 3 |

**Название:** Классы, наследование, полиморфизм

**Дисциплина:** Языки программирования для работы с большими данными

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Студент | ИУ6-22М |  |  | Р.В. Баканов | |
|  | (Группа) |  | (Подпись, дата) | | (И.О. Фамилия) |
|  |  |  |  | |  |
| Преподаватель |  |  |  | | П.В. Степанов |
|  |  |  | (Подпись, дата) | | (И.О. Фамилия) |

Москва, 2024

**Цель работы:** освоить базовые принципы работы с наследованием и полиморфизмом в Java.

**Вариант: 2.**

**Задание 1:** Определить класс Вектор размерности n. Определить несколько конструкторов. Реализовать методы для вычисления модуля вектора, скалярного произведения, сложения, вычитания, умножения на константу. Объявить массив объектов. Написать метод, который для заданной пары векторов будет определять, являются ли они коллинеарными или ортогональными.

Код решения приведен в листингах 1 и 2.

Листинг 1 — Задание 1 (1)

public class Main {  
 public static void main(String[] args) throws Exception {  
 var vector1 = new Vector(2, 0);  
 var vector2 = new Vector(-2, -4);  
  
 System.*out*.println(vector1.Subtract(vector2).toString());  
 }  
}

Листинг 2 – Задание 1 (2)

import jdk.jshell.spi.ExecutionControl;  
  
import java.util.ArrayList;  
import java.util.HashSet;  
import java.util.StringJoiner;  
  
public class Vector {  
 public int VectorSpaceDimension;  
 public int[] Coordinates;  
  
 public Vector(int xCoordinate) {  
 VectorSpaceDimension = 1;  
 Coordinates = new int[] { xCoordinate };  
 }  
  
 public Vector(int xCoordinate, int yCoordinate) {  
 VectorSpaceDimension = 2;  
 Coordinates = new int[] { xCoordinate, yCoordinate };  
 }  
  
 public Vector(int xCoordinate, int yCoordinate, int zCoordinate) {  
 VectorSpaceDimension = 3;  
 Coordinates = new int[] { xCoordinate, yCoordinate, zCoordinate };  
 }  
  
 public double GetModulus() {  
 var sumOfSquares = 0;  
  
 for (int i = 0; i < VectorSpaceDimension; i++) {  
 sumOfSquares += Coordinates[i] \* Coordinates[i];  
 }  
  
 return Math.*sqrt*(sumOfSquares);  
 }  
  
 public Vector Add(Vector vector) throws Exception {  
 *EnsureVectorsHaveSameVectorSpaceDimensions*(this, vector);  
  
 var resultVectorCoordinates = new ArrayList<Integer>();  
  
 for (int i = 0; i < VectorSpaceDimension; i++) {  
 resultVectorCoordinates.add(Coordinates[i] + vector.Coordinates[i]);  
 }  
  
 return *CreateVectorFromCoordinates*(resultVectorCoordinates);  
 }  
  
 public Vector Subtract(Vector vector) throws Exception {  
 *EnsureVectorsHaveSameVectorSpaceDimensions*(this, vector);  
  
 var resultVectorCoordinates = new ArrayList<Integer>();  
  
 for (int i = 0; i < VectorSpaceDimension; i++) {  
 resultVectorCoordinates.add(Coordinates[i] - vector.Coordinates[i]);  
 }  
  
 return *CreateVectorFromCoordinates*(resultVectorCoordinates);  
 }  
  
 public Vector MultiplyBy(int scalar) throws Exception {  
 var resultVectorCoordinates = new ArrayList<Integer>();  
  
 for (int i = 0; i < VectorSpaceDimension; i++) {  
 resultVectorCoordinates.add(Coordinates[i] \* scalar);  
 }  
  
 return *CreateVectorFromCoordinates*(resultVectorCoordinates);  
 }  
  
 public int MultiplyBy(Vector vector) {  
 *EnsureVectorsHaveSameVectorSpaceDimensions*(this, vector);  
  
 var scalarProduct = 0;  
  
 for (int i = 0; i < VectorSpaceDimension; i++) {  
 scalarProduct += Coordinates[i] \* vector.Coordinates[i];  
 }  
  
 return scalarProduct;  
 }  
  
 public VectorsRelation DefineRelationTo(Vector vector) {  
 if (this.IsCollinearTo(vector)) {  
 return VectorsRelation.*Collinear*;  
 }  
  
 if (this.IsOrthogonalTo(vector)) {  
 return VectorsRelation.*Orthogonal*;  
 }  
  
 return VectorsRelation.*None*;  
 }  
  
 @Override  
 public String toString() {  
 var stringJoiner = new StringJoiner(", ");  
  
 for (int coordinate : Coordinates) {  
 stringJoiner.add(Integer.toString(coordinate));  
 }  
  
 return "(" + stringJoiner + ")";  
 }  
  
 private boolean IsCollinearTo(Vector vector) {  
 EnsureVectorsHaveSameVectorSpaceDimensions(this, vector);  
  
 var coordinateRelations = new HashSet<Double>();  
  
 for (int i = 0; i < this.VectorSpaceDimension; i++) {  
 coordinateRelations.add((double) (Coordinates[i] / vector.Coordinates[i]));  
 }  
  
 return coordinateRelations.size() == 1;  
 }  
  
 private boolean IsOrthogonalTo(Vector vector) {  
 EnsureVectorsHaveSameVectorSpaceDimensions(this, vector);  
  
 var scalarProduct = this.MultiplyBy(vector);  
  
 return scalarProduct == 0;  
 }  
  
 private static void EnsureVectorsHaveSameVectorSpaceDimensions(Vector left, Vector right) {  
 if (left.VectorSpaceDimension != right.VectorSpaceDimension) {  
 throw new IllegalArgumentException("Vectors have different vector space dimensions");  
 }  
 }  
  
 private static Vector CreateVectorFromCoordinates(ArrayList<Integer> resultVectorCoordinates) throws Exception {  
 return switch (resultVectorCoordinates.size()) {  
 case 1 -> new Vector(resultVectorCoordinates.get(0));  
 case 2 -> new Vector(resultVectorCoordinates.get(0),  
 resultVectorCoordinates.get(1));  
 case 3 -> new Vector(resultVectorCoordinates.get(0),  
 resultVectorCoordinates.get(1),  
 resultVectorCoordinates.get(2));  
 default -> throw new Exception(  
 String.format(  
 "Vector space dimension %d is not supported",  
 resultVectorCoordinates.size()));  
 };  
 }  
  
 public enum VectorsRelation {  
 None,  
 Collinear,  
 Orthogonal  
 }  
}

**Задание 2:** Определить класс Вектор в R3. Реализовать методы для проверки векторов на ортогональность, проверки пересечения не ортогональных векторов, сравнения векторов. Создать массив из m объектов. Определить, какие из векторов компланарны.

Код решения приведен в листингах 3 - 7.

Листинг 3 — Задание 2 (1)

public class Main {  
 public static void main(String[] args) {  
 var vectors = new Vector3D[] {  
 new Vector3D(  
 new Point3D(0, 0, 0),  
 new Point3D(1, 1, -1)  
 ),  
 new Vector3D(  
 new Point3D(0, -1, 3),  
 new Point3D(-1, 2, -2)  
 ),  
 new Vector3D(  
 new Point3D(3, 2, -3),  
 new Point3D(3, -2, 3)  
 )  
 };  
  
 System.*out*.println(Vector3DBase.*AreCoplanarVectors*(vectors[0], vectors[1], vectors[2]));  
 }  
}

Листинг 4 — Задание 2 (2)

public class Point2D {  
 public int X;  
 public int Y;  
 public int Z;  
  
 public Point2D(int x, int y) {  
 X = x;  
 Y = y;  
 }  
}

Листинг 5 – Задание 2 (3)

public class Point3D {  
 public int X;  
 public int Y;  
 public int Z;  
 public Point2D XYProjection;  
 public Point2D XZProjection;  
 public Point2D YZProjection;  
  
 public Point3D(int x, int y, int z) {  
 X = x;  
 Y = y;  
 Z = z;  
 XYProjection = new Point2D(x, y);  
 XZProjection = new Point2D(x, z);  
 YZProjection = new Point2D(y, z);  
 }  
}

Листинг 6 – Задание 2 (4)

public class Vector3D extends Vector3DBase {  
 public Point3D A;  
 public Point3D B;  
  
 public Vector3D(Point3D a, Point3D b) {  
 super(b.X - a.X, b.Y - a.Y, b.Z - a.Z);  
 A = a;  
 B = b;  
 }  
  
 public boolean IntersectsWith(Vector3D vector) {  
 return *AreOnSamePlane*(this, vector)  
 && *ProjectionsOntoXYPlaneIntersect*(this, vector)  
 && *ProjectionsOntoXZPlaneIntersect*(this, vector)  
 && *ProjectionsOntoYZPlaneIntersect*(this, vector);  
 }  
  
 private static boolean AreOnSamePlane(Vector3D first, Vector3D second) {  
 var firstAuxVector = new Vector3D(first.A, second.A);  
 var secondAuxVector = new Vector3D(first.A, second.B);  
  
 return *AreCoplanarVectors*(first, firstAuxVector, secondAuxVector);  
 }  
  
 private static boolean ProjectionsOntoXYPlaneIntersect(Vector3D left, Vector3D right) {  
 var leftXYPlaneProjection = new Vector2DProjection(left.A.XYProjection, left.B.XYProjection);  
 var rightXYPlaneProjection = new Vector2DProjection(right.A.XYProjection, right.B.XYProjection);  
  
 return *ProjectionsOntoPlaneIntersects*(leftXYPlaneProjection, rightXYPlaneProjection);  
 }  
  
 private static boolean ProjectionsOntoXZPlaneIntersect(Vector3D left, Vector3D right) {  
 var leftXZPlaneProjection = new Vector2DProjection(left.A.XZProjection, left.B.XZProjection);  
 var rightXZPlaneProjection = new Vector2DProjection(right.A.XZProjection, right.B.XZProjection);  
  
 return *ProjectionsOntoPlaneIntersects*(leftXZPlaneProjection, rightXZPlaneProjection);  
 }  
  
 private static boolean ProjectionsOntoYZPlaneIntersect(Vector3D left, Vector3D right) {  
 var leftYZPlaneProjection = new Vector2DProjection(left.A.YZProjection, left.B.YZProjection);  
 var rightYZPlaneProjection = new Vector2DProjection(right.A.YZProjection, right.B.YZProjection);  
  
 return *ProjectionsOntoPlaneIntersects*(leftYZPlaneProjection, rightYZPlaneProjection);  
 }  
  
 private static boolean ProjectionsOntoPlaneIntersects(Vector2DProjection left, Vector2DProjection right) {  
 return *ProjectionsOntoLineIntersects*(new Vector1DProjection(left.A.X, left.B.X), new Vector1DProjection(right.A.X, right.B.X))  
 && *ProjectionsOntoLineIntersects*(new Vector1DProjection(left.A.Y, left.B.Y), new Vector1DProjection(right.A.Y, right.B.Y))  
 && *GetSignedArea*(left.A, left.B, right.A) \* *GetSignedArea*(left.A, left.B, right.B) <= 0  
 && *GetSignedArea*(right.A, right.B, left.A) \* *GetSignedArea*(right.A, right.B, left.B) <= 0;  
 }  
  
 private static boolean ProjectionsOntoLineIntersects(Vector1DProjection left, Vector1DProjection right) {  
 int leftMin;  
 int leftMax;  
  
 if (left.A > left.B) {  
 leftMin = left.B;  
 leftMax = left.A;  
 } else {  
 leftMin = left.A;  
 leftMax = left.B;  
 }  
  
 int rightMin;  
 int rightMax;  
  
 if (right.A > right.B) {  
 rightMin = right.B;  
 rightMax = right.A;  
 } else {  
 rightMin = right.A;  
 rightMax = right.B;  
 }  
  
 return Math.*max*(leftMin, rightMin) <= Math.*min*(leftMax, rightMax);  
 }  
  
 private static int GetSignedArea(Point2D a, Point2D b, Point2D c) {  
 return (b.X - a.X) \* (c.Y - a.Y) - (b.Y - a.Y) \* (c.X - a.X);  
 }  
  
 private static class Vector2DProjection {  
 public Point2D A;  
 public Point2D B;  
  
 public Vector2DProjection(Point2D a, Point2D b) {  
 A = a;  
 B = b;  
 }  
 }  
  
 private static class Vector1DProjection {  
 public int A;  
 public int B;  
  
 public Vector1DProjection(int a, int b) {  
 A = a;  
 B = b;  
 }  
 }  
}

Листинг 7 – Задание 2 (5)

public class Vector3DBase {  
 public int X;  
 public int Y;  
 public int Z;  
 public double Modulus;  
  
 public Vector3DBase(int x, int y, int z) {  
 X = x;  
 Y = y;  
 Z = z;  
 Modulus = Math.*sqrt*(X \* X + Y \* Y + Z \* Z);  
 }  
  
 public boolean IsOrthogonalTo(Vector3DBase vector) {  
 var scalarProduct = *Multiply*(this, vector);  
  
 return scalarProduct == 0;  
 }  
  
 public boolean IsEqualTo(Vector3DBase vector) {  
 return Modulus == vector.Modulus;  
 }  
  
 public static boolean AreCoplanarVectors(Vector3DBase first, Vector3DBase second, Vector3DBase third) {  
 var determinant = new int[][] {  
 { first.X, first.Y, first.Z },  
 { second.X, second.Y, second.Z },  
 { third.X, third.Y, third.Z }  
 };  
  
 var determinantValue = determinant[0][0] \* (determinant[1][1] \* determinant[2][2] - determinant[1][2] \* determinant[2][1])  
 - determinant[0][1] \* (determinant[1][0] \* determinant[2][2] - determinant[1][2] \* determinant[2][0])  
 + determinant[0][2] \* (determinant[1][0] \* determinant[2][1] - determinant[1][1] \* determinant[2][0]);  
  
 return determinantValue == 0;  
 }  
  
 private static int Multiply(Vector3DBase left, Vector3DBase right) {  
 return left.X \* right.X + left.Y \* right.Y + left.Z \* right.Z;  
 }  
}

**Задание 3:** Customer: id, Фамилия, Имя, Отчество, Адрес, Номер кредитной карточки, Номер банковского счета. Создать массив объектов. Вывести: a) список покупателей в алфавитном порядке; b) список покупателей, у которых номер кредитной карточки находится в заданном интервале.

Код решения приведен в листингах 8 - 11.

Листинг 8 – Задание 3 (1)

public class Main {  
 public static void main(String[] args) {  
 var customer1 = new Customer(  
 0,  
 "",  
 "B",  
 "",  
 "",  
 1,  
 0  
 );  
  
 var customer2 = new Customer(  
 0,  
 "",  
 "A",  
 "",  
 "",  
 2,  
 0  
 );  
  
 var customer3 = new Customer(  
 0,  
 "",  
 "C",  
 "",  
 "",  
 3,  
 0  
 );  
  
 var customers = new Customer[] { customer1, customer2, customer3 };  
  
 var customerCollection = new CustomerCollection(customers);  
  
 System.*out*.println(customerCollection);  
  
 System.*out*.println();  
  
 System.*out*.println(customerCollection.sort(CustomerComparator.*bySurname*()));  
  
 System.*out*.println();  
  
 System.*out*.println(customerCollection.filterByCardNumber(1, 2));  
 }  
}

Листинг 9 – Задание 3 (2)

import java.util.StringJoiner;  
  
public class Customer {  
 private int Id;  
 private String Name;  
 private String Surname;  
 private String Patronymic;  
 private String Address;  
 private int CardNumber;  
 private int BankAccountNumber;  
  
 public Customer(int id,  
 String name,  
 String surname,  
 String patronymic,  
 String address,  
 int cardNumber,  
 int bankAccountNumber) {  
 Id = id;  
 Name = name;  
 Surname = surname;  
 Patronymic = patronymic;  
 Address = address;  
 CardNumber = cardNumber;  
 BankAccountNumber = bankAccountNumber;  
 }  
  
 @Override  
 public String toString() {  
 var stringJoiner = new StringJoiner(", ");  
  
 stringJoiner  
 .add("id: " + Id)  
 .add("name: " + Name)  
 .add("surname: " + Surname)  
 .add("patronymic: " + Patronymic)  
 .add("address: " + Address)  
 .add("card number: " + CardNumber)  
 .add("bank account number: " + BankAccountNumber);  
  
 return stringJoiner.toString();  
 }  
  
 public int getId() {  
 return Id;  
 }  
  
 public String getName() {  
 return Name;  
 }  
  
 public String getSurname() {  
 return Surname;  
 }  
  
 public String getPatronymic() {  
 return Patronymic;  
 }  
  
 public String getAddress() {  
 return Address;  
 }  
  
 public int getCardNumber() {  
 return CardNumber;  
 }  
  
 public int getBankAccountNumber() {  
 return BankAccountNumber;  
 }  
  
 public void setId(int id) {  
 Id = id;  
 }  
  
 public void setName(String name) {  
 Name = name;  
 }  
  
 public void setSurname(String surname) {  
 Surname = surname;  
 }  
  
 public void setPatronymic(String patronymic) {  
 Patronymic = patronymic;  
 }  
  
 public void setAddress(String address) {  
 Address = address;  
 }  
  
 public void setCardNumber(int cardNumber) {  
 CardNumber = cardNumber;  
 }  
  
 public void setBankAccountNumber(int bankAccountNumber) {  
 BankAccountNumber = bankAccountNumber;  
 }  
}

Листинг 10 – Задание 3 (3)

import java.util.Arrays;  
import java.util.StringJoiner;  
import java.util.function.Predicate;  
  
public class CustomerCollection {  
 public final Customer[] Data;  
  
 public CustomerCollection(Customer[] data) {  
 Data = data;  
 }  
  
 public CustomerCollection sort(CustomerComparator comparator) {  
 var dataCopy = Data;  
  
 Arrays.*sort*(dataCopy, comparator);  
  
 return new CustomerCollection(dataCopy);  
 }  
  
 public CustomerCollection filterByCardNumber(int lowerBound, int upperBound) {  
 Predicate<Customer> predicate = (Customer customer) -> {  
 var cardNumber = customer.getCardNumber();  
  
 return cardNumber >= lowerBound && cardNumber <= upperBound;  
 };  
  
 var filteredData = Arrays.*stream*(Data).filter(predicate).toArray(Customer[]::new);  
  
 return new CustomerCollection(filteredData);  
 }  
  
 @Override  
 public String toString() {  
 var stringJoiner = new StringJoiner("\n");  
  
 for (var customer : Data) {  
 stringJoiner.add(customer.toString());  
 }  
  
 return stringJoiner.toString();  
 }  
}

Листинг 11 - Задание 3 (4)

import java.util.Comparator;  
  
public abstract class CustomerComparator implements Comparator<Customer> {  
 public static CustomerComparator bySurname() {  
 return new CustomerSurnameComparator();  
 }  
  
 private static class CustomerSurnameComparator extends CustomerComparator {  
  
 @Override  
 public int compare(Customer o1, Customer o2) {  
 return o1.getSurname().compareTo(o2.getSurname());  
 }  
 }  
}

**Задание 4:** Patient: id, Фамилия, Имя, Отчество, Адрес, Телефон, Номер медицинской карты, Диагноз. Создать массив объектов. Вывести: a) список пациентов, имеющих данный диагноз; b) список пациентов, номер медицинской карты у которых находится в заданном интервале.

Код решения приведен в листингах 12, 13 и 14.

Листинг 12 — Задание 4 (1)

public class Main {  
 public static void main(String[] args) {  
 var patients = new Patient[] {  
 new Patient(  
 1,  
 "",  
 "",  
 "",  
 "",  
 "",  
 1,  
 "A"  
 ),  
 new Patient(  
 2,  
 "",  
 "",  
 "",  
 "",  
 "",  
 2,  
 "B"  
 ),  
 new Patient(  
 3,  
 "",  
 "",  
 "",  
 "",  
 "",  
 3,  
 "B"  
 )  
 };  
  
 var patientCollection = new PatientCollection(patients);  
  
 System.*out*.println(patientCollection);  
 System.*out*.println();  
  
 System.*out*.println(patientCollection.filterByDiagnosis("B"));  
 System.*out*.println();  
  
 System.*out*.println(patientCollection.filterByCardNumber(1, 2));  
 }  
}

Листинг 13 – Задание 4 (2)

import java.util.StringJoiner;  
  
public class Patient {  
 private int Id;  
 private String Name;  
 private String Surname;  
 private String Patronymic;  
 private String Address;  
 private String PhoneNumber;  
 private int CardNumber;  
 private String Diagnosis;  
  
 public Patient(int id,  
 String name,  
 String surname,  
 String patronymic,  
 String address,  
 String phoneNumber,  
 int cardNumber,  
 String diagnosis) {  
 Id = id;  
 Name = name;  
 Surname = surname;  
 Patronymic = patronymic;  
 Address = address;  
 PhoneNumber = phoneNumber;  
 CardNumber = cardNumber;  
 Diagnosis = diagnosis;  
 }  
  
 @Override  
 public String toString() {  
 var stringJoiner = new StringJoiner(", ");  
  
 stringJoiner  
 .add("id: " + Id)  
 .add("name: " + Name)  
 .add("surname: " + Surname)  
 .add("patronymic: " + Patronymic)  
 .add("address: " + Address)  
 .add("phone number: " + PhoneNumber)  
 .add("card number: " + CardNumber)  
 .add("diagnosis: " + Diagnosis);  
  
 return stringJoiner.toString();  
 }  
  
 public int getId() {  
 return Id;  
 }  
  
 public String getName() {  
 return Name;  
 }  
  
 public String getSurname() {  
 return Surname;  
 }  
  
 public String getPatronymic() {  
 return Patronymic;  
 }  
  
 public String getAddress() {  
 return Address;  
 }  
 public String getPhoneNumber() {  
 return PhoneNumber;  
 }  
  
 public int getCardNumber() {  
 return CardNumber;  
 }  
  
 public String getDiagnosis() {  
 return Diagnosis;  
 }  
  
 public void setId(int id) {  
 Id = id;  
 }  
  
 public void setName(String name) {  
 Name = name;  
 }  
  
 public void setSurname(String surname) {  
 Surname = surname;  
 }  
  
 public void setPatronymic(String patronymic) {  
 Patronymic = patronymic;  
 }  
  
 public void setAddress(String address) {  
 Address = address;  
 }  
 public void setPhoneNumber(String phoneNumber) {  
 PhoneNumber = phoneNumber;  
 }  
  
 public void setCardNumber(int cardNumber) {  
 CardNumber = cardNumber;  
 }  
  
 public void setDiagnosis(String diagnosis) {  
 Diagnosis = diagnosis;  
 }  
}

Листинг 14 – Задание 4 (3)

import java.util.Arrays;  
import java.util.StringJoiner;  
import java.util.function.Predicate;  
  
public class PatientCollection {  
 public final Patient[] Data;  
  
 public PatientCollection(Patient[] data) {  
 Data = data;  
 }  
  
 public PatientCollection filterByDiagnosis(String targetDiagnosis) {  
 var filteredData = Arrays.*stream*(Data)  
 .filter(patient -> patient.getDiagnosis().equals(targetDiagnosis))  
 .toArray(Patient[]::new);  
  
 return new PatientCollection(filteredData);  
 }  
  
 public PatientCollection filterByCardNumber(int lowerBound, int upperBound) {  
 Predicate<Patient> predicate = (Patient customer) -> {  
 var cardNumber = customer.getCardNumber();  
  
 return cardNumber >= lowerBound && cardNumber <= upperBound;  
 };  
  
 var filteredData = Arrays.*stream*(Data).filter(predicate).toArray(Patient[]::new);  
  
 return new PatientCollection(filteredData);  
 }  
  
 @Override  
 public String toString() {  
 var stringJoiner = new StringJoiner("\n");  
  
 for (var patient : Data) {  
 stringJoiner.add(patient.toString());  
 }  
  
 return stringJoiner.toString();  
 }  
}

**Задание 5:** Создать объект класса Одномерный массив, используя класс Массив. Методы: создать, вывести на консоль, выполнить операции (сложить, вычесть, перемножить).

Код решения приведен в листингах 15 - 18.

Листинг 15 – Задание 5 (1)

public class Main {  
 public static void main(String[] args) {  
 var customArrayElement1 = new CustomArrayElementOne(1, 2);  
 var customArrayElement2 = new CustomArrayElementOne(3, 4);  
  
 var customArray = new CustomArray<CustomArrayElementOne>();  
  
 customArray.add(customArrayElement1);  
 customArray.add(customArrayElement2);  
  
 customArray.print();  
 }  
}

Листинг 16 – Задание 5 (2)

public interface ICustomArrayElement<T extends ICustomArrayElement<T>> {  
 T sum(T element);  
 T subtract(T element);  
 T multiply(T element);  
 String format();  
}

Листинг 17 – Задание 5 (3)

import java.util.\*;  
  
public class CustomArray<T extends ICustomArrayElement<T>> extends ArrayList<T> {  
 public CustomArray() {  
 super();  
 }  
  
 public CustomArray(int initialCapacity) {  
 super(initialCapacity);  
 }  
  
 public CustomArray(Collection<T> data) {  
 super(data);  
 }  
  
 @Override  
 public boolean add(T element) {  
 return super.add(element);  
 }  
  
 public CustomArray<T> sum(CustomArray<T> array) throws IllegalArgumentException {  
 CustomArray<T> result = new CustomArray<>();  
  
 if(size() != array.size()) {  
 throw new IllegalArgumentException("Arrays have different sizes");  
 }  
  
 for (int i = 0; i < size(); i++) {  
 result.add(get(i).sum(array.get(i)));  
 }  
  
 return result;  
 }  
  
 public CustomArray<T> subtract(CustomArray<T> array) throws IllegalArgumentException {  
 CustomArray<T> result = new CustomArray<>();  
  
 if(size() != array.size()) {  
 throw new IllegalArgumentException("Arrays have different sizes");  
 }  
  
 for (int i = 0; i < size(); i++) {  
 result.add(get(i).subtract(array.get(i)));  
 }  
  
 return result;  
 }  
  
 public CustomArray<T> multiply(CustomArray<T> array) throws IllegalArgumentException {  
 CustomArray<T> result = new CustomArray<>();  
  
 if(size() != array.size()) {  
 throw new IllegalArgumentException("Arrays have different sizes");  
 }  
  
 for (int i = 0; i < size(); i++) {  
 result.add(get(i).multiply(array.get(i)));  
 }  
  
 return result;  
 }  
  
 public void print() {  
 var formattedString = toString();  
  
 System.out.println(formattedString);  
 }  
  
 @Override  
 public int hashCode() {  
 return super.hashCode();  
 }  
  
 @Override  
 public boolean equals(Object object) {  
 return super.equals(object);  
 }  
  
 @Override  
 public String toString() {  
 Iterator<T> iterator = iterator();  
  
 if (!iterator.hasNext())  
 return "[]";  
  
 var stringBuilder = new StringBuilder();  
 stringBuilder.append('[');  
  
 for (;;) {  
 T element = iterator.next();  
  
 stringBuilder.append(element.format());  
  
 if (!iterator.hasNext()) {  
 stringBuilder.append(']');  
  
 return stringBuilder.toString();  
 }  
  
 stringBuilder.append(", ");  
 }  
 }  
}

Листинг 18 – Задание 5 (4)

public class CustomArrayElementOne implements ICustomArrayElement<CustomArrayElementOne> {  
 public int A;  
 public int B;  
  
 public CustomArrayElementOne(int a, int b) {  
 A = a;  
 B = b;  
 }  
  
 @Override  
 public CustomArrayElementOne sum(CustomArrayElementOne element) {  
 return new CustomArrayElementOne(A + element.A, B + element.B);  
 }  
  
 @Override  
 public CustomArrayElementOne subtract(CustomArrayElementOne element) {  
 return new CustomArrayElementOne(A - element.A, B - element.B);  
 }  
  
 @Override  
 public CustomArrayElementOne multiply(CustomArrayElementOne element) {  
 return new CustomArrayElementOne(A \* element.A, B \* element.B);  
 }  
  
 @Override  
 public String format() {  
 return String.*format*("{A: %d, B: %d}", A, B);  
 }  
}

**Задание 6:** Создать объект класса Простая дробь, используя класс Число. Методы: вывод на экран, сложение, вычитание, умножение, деление.

Код решения приведен в листингах 19 и 20.

Листинг 19 – Задание 6 (1)

public class Main {  
 public static void main(String[] args) {  
 var fraction1 = new SimpleFraction(-5, 12);  
 var fraction2 = new SimpleFraction(3, 4);  
  
 var sum = fraction1.sum(fraction2);  
 var difference = fraction1.subtract(fraction2);  
 var product = fraction1.multiply(fraction2);  
 var quotient = fraction1.divide(fraction2);  
  
 System.*out*.printf("%s + %s = %s (%s)\n", fraction1, fraction2, sum, sum.simplify());  
 System.*out*.printf("%s - %s = %s (%s)\n", fraction1, fraction2, difference, difference.simplify());  
 System.*out*.printf("%s \* %s = %s (%s)\n", fraction1, fraction2, product, product.simplify());  
 System.*out*.printf("%s / %s = %s (%s)\n", fraction1, fraction2, quotient, quotient.simplify());  
 }  
}

Листинг 20 – Задание 6 (2)

public class SimpleFraction extends Number {  
 public final int Numerator;  
 public final int Denominator;  
  
 public SimpleFraction(int numerator, int denominator) {  
 if (denominator == 0) {  
 throw new IllegalArgumentException("Denominator can not be 0");  
 }  
  
 var numeratorFactor = numerator \* denominator > 0 ? 1 : -1;  
  
 Numerator = numeratorFactor \* Math.*abs*(numerator);  
 Denominator = Math.*abs*(denominator);  
 }  
  
 public SimpleFraction sum(SimpleFraction simpleFraction) {  
 var resultNumerator = Numerator \* simpleFraction.Denominator + simpleFraction.Numerator \* Denominator;  
 var resultDenominator = Denominator \* simpleFraction.Denominator;  
  
 return new SimpleFraction(resultNumerator, resultDenominator);  
 }  
  
 public SimpleFraction subtract(SimpleFraction simpleFraction) {  
 var resultNumerator = Numerator \* simpleFraction.Denominator - simpleFraction.Numerator \* Denominator;  
 var resultDenominator = Denominator \* simpleFraction.Denominator;  
  
 return new SimpleFraction(resultNumerator, resultDenominator);  
 }  
  
 public SimpleFraction multiply(SimpleFraction simpleFraction) {  
 var resultNumerator = Numerator \* simpleFraction.Numerator;  
 var resultDenominator = Denominator \* simpleFraction.Denominator;  
  
 return new SimpleFraction(resultNumerator, resultDenominator);  
 }  
  
 public SimpleFraction divide(SimpleFraction simpleFraction) {  
 var resultNumerator = Numerator \* simpleFraction.Denominator;  
 var resultDenominator = Denominator \* simpleFraction.Numerator;  
  
 return new SimpleFraction(resultNumerator, resultDenominator);  
 }  
  
 @Override  
 public int intValue() {  
 return Numerator / Denominator;  
 }  
  
 @Override  
 public long longValue() {  
 return (long) Numerator / Denominator;  
 }  
  
 @Override  
 public float floatValue() {  
 return (float) Numerator / Denominator;  
 }  
  
 @Override  
 public double doubleValue() {  
 return (double) Numerator / Denominator;  
 }  
  
 @Override  
 public boolean equals(Object object) {  
 if (object == this) {  
 return true;  
 }  
  
 if (!(object instanceof SimpleFraction simpleFraction)) {  
 return false;  
 }  
  
 return Numerator \* simpleFraction.Denominator == simpleFraction.Numerator \* Denominator;  
 }  
  
 @Override  
 public int hashCode() {  
 return Numerator \* 31 + Denominator;  
 }  
  
 @Override  
 public String toString() {  
 return Numerator + "/" + Denominator;  
 }  
  
 public SimpleFraction simplify() {  
 var greatestCommonDivisor = getGreatestCommonDivisor();  
  
 return new SimpleFraction(  
 Numerator / greatestCommonDivisor,  
 Denominator / greatestCommonDivisor  
 );  
 }  
  
 private int getGreatestCommonDivisor() {  
 if (Numerator == 0) {  
 return 0;  
 }  
  
 var numeratorAbsoluteValue = Math.*abs*(Numerator);  
 var denominatorAbsoluteValue = Math.*abs*(Denominator);  
  
 while (numeratorAbsoluteValue != denominatorAbsoluteValue) {  
 if (numeratorAbsoluteValue > denominatorAbsoluteValue) {  
 numeratorAbsoluteValue -= denominatorAbsoluteValue;  
 } else {  
 denominatorAbsoluteValue -= numeratorAbsoluteValue;  
 }  
 }  
  
 return numeratorAbsoluteValue;  
 }  
}

**Задание 7:** Система Платежи. Клиент имеет Счет в банке и Кредитную Карту (КК). Клиент может оплатить Заказ, сделать платеж на другой Счет, заблокировать КК и аннулировать Счет. Администратор может заблокировать КК за превышение кредита.

Код решения приведен в листингах 21 - 32.

Листинг 21 – Задание 7 (1)

import Order.Order;  
import PaymentSystem.PaymentSystem;  
  
import java.math.BigDecimal;  
  
public class Main {  
 public static void main(String[] args) throws Exception {  
 var paymentSystem = new PaymentSystem();  
  
 var customer1Id = paymentSystem.createCustomer();  
 var customer2Id = paymentSystem.createCustomer();  
  
 var adminId = paymentSystem.createAdmin();  
  
 var customer1Interface = paymentSystem.createCustomerInterface(customer1Id);  
 customer1Interface.createAccount(1, new BigDecimal(100));  
 customer1Interface.createAccount(2, new BigDecimal(200));  
  
 var order = new Order(1, new BigDecimal(50));  
  
 // Оплата заказа и перевод денег на свой счет  
 customer1Interface.payOrder(order, 1);  
 customer1Interface.transferMoney(1, 2, new BigDecimal(50));  
  
 var customer2Interface = paymentSystem.createCustomerInterface(customer2Id);  
 customer2Interface.createAccount(3, new BigDecimal(0));  
  
 // Перевод средств другому пользователю  
 customer1Interface.transferMoney(2, 3, new BigDecimal(50));  
  
 // Аннуляция счета  
 customer1Interface.disableAccount(1);  
  
 customer1Interface.createCreditCard(1, new BigDecimal(50));  
 customer1Interface.createCreditCard(2, new BigDecimal(-50));  
  
 // Блокировка кредитной карты  
 customer1Interface.blockCreditCard(1);  
  
 var adminInterface = paymentSystem.createAdminInterface(adminId);  
  
 // Блокировка кредитной карты администратором  
 adminInterface.blockCustomerCreditCard(1, 2);  
 }  
}

Листинг 22 – Задание 7 (2)

package Admin;  
  
public class Admin {  
 public int Id;  
  
 public Admin(int id) {  
 Id = id;  
 }  
}

Листинг 23 – Задание 7 (3)

package Customer;  
  
import java.math.BigDecimal;  
  
public class Account extends AccountBase {  
 private boolean \_isDisabled;  
  
 public Account(int id, BigDecimal balance) {  
 super(id, balance);  
 \_isDisabled = false;  
 }  
  
 public void decreaseBalanceBy(BigDecimal amount, String operationDetails) throws Exception {  
 if (amount.compareTo(Balance) > 0) {  
 throw new Exception("На счету недостаточно средств");  
 }  
  
 Balance = Balance.subtract(amount);  
 OperationHistory.add(operationDetails);  
 }  
  
 public void disable() {  
 \_isDisabled = true;  
 }  
}

Листинг 24 – Задание 7 (4)

package Customer;  
  
import java.math.BigDecimal;  
import java.util.ArrayList;  
  
public abstract class AccountBase {  
 protected BigDecimal Balance;  
 protected ArrayList<String> OperationHistory;  
  
 public final int Id;  
  
 public AccountBase(int id, BigDecimal balance) {  
 Id = id;  
 Balance = balance;  
 OperationHistory = new ArrayList<>();  
 }  
  
 public void increaseBalanceBy(BigDecimal amount, String operationDetails) {  
 Balance = Balance.add(amount);  
 OperationHistory.add(operationDetails);  
 }  
}

Листинг 25 – Задание 7 (5)

package Customer;  
  
import java.math.BigDecimal;  
  
public class CreditCard {  
 private BigDecimal \_balance;  
 private boolean \_isBlocked;  
  
 public final int Id;  
  
 public CreditCard(int id, BigDecimal balance) {  
 Id = id;  
 \_balance = balance;  
 \_isBlocked = false;  
 }  
  
 public void block() {  
 \_isBlocked = true;  
 }  
  
 public BigDecimal getBalance() {  
 return \_balance;  
 }  
}

Листинг 26 – Задание 7 (6)

package Customer;  
  
import java.math.BigDecimal;  
  
public class Customer extends CustomerForAdmin {  
 public Customer(int id) {  
 super(id);  
 }  
  
 @Override  
 public Account getAccount(int accountId) throws Exception {  
 return (Account) super.getAccount(accountId);  
 }  
  
 public void createAccount(int accountId, BigDecimal balance) {  
 var account = new Account(accountId, balance);  
  
 Accounts.put(accountId, account);  
 }  
  
 public void createCreditCard(int creditCardId, BigDecimal balance) {  
 var creditCard = new CreditCard(creditCardId, balance);  
  
 CreditCards.put(creditCardId, creditCard);  
 }  
}

Листинг 27 – Задание 7 (7)

package Customer;  
  
public class CustomerForAdmin extends CustomerForCustomer {  
 public CustomerForAdmin(int id) {  
 super(id);  
 }  
  
 public CreditCard getCreditCard(int creditCardId) throws Exception {  
 if (!CreditCards.containsKey(creditCardId)) {  
 throw new Exception("У пользователя нет такой кредитной карты");  
 }  
  
 return CreditCards.get(creditCardId);  
 }  
}

Листинг 28 – Задание 7 (8)

package Customer;  
  
import java.util.HashMap;  
  
public class CustomerForCustomer {  
 protected HashMap<Integer, Account> Accounts;  
 protected HashMap<Integer, CreditCard> CreditCards;  
  
 public int Id;  
  
 public CustomerForCustomer(int id) {  
  
 Id = id;  
 Accounts = new HashMap<>();  
 CreditCards = new HashMap<>();  
 }  
  
 public AccountBase getAccount(int accountId) throws Exception {  
 if (!Accounts.containsKey(accountId)) {  
 throw new Exception("У пользователя нет такого счета");  
 }  
  
 return Accounts.get(accountId);  
 }  
}

Листинг 29 – Задание 7 (9)

package Order;  
  
import java.math.BigDecimal;  
  
public class Order {  
 public final int Id;  
 public final BigDecimal TotalAmount;  
  
 public Order(int id, BigDecimal totalAmount) {  
 Id = id;  
 TotalAmount = totalAmount;  
 }  
}

Листинг 30 – Задание 7 (10)

package PaymentSystem;  
  
import Admin.Admin;  
import Customer.CustomerForAdmin;  
  
import java.math.BigDecimal;  
  
public class AdminInterface {  
 private final Admin \_admin;  
 private PaymentSystem \_paymentSystem;  
  
 public AdminInterface(Admin admin, PaymentSystem paymentSystem) {  
 \_admin = admin;  
 \_paymentSystem = paymentSystem;  
 }  
  
 public void blockCustomerCreditCard(int customerId, int creditCardId) throws Exception {  
 var creditCard = getCustomer(customerId).getCreditCard(creditCardId);  
  
 if (creditCard.getBalance().compareTo(new BigDecimal(0)) >= 0) {  
 throw new Exception("Невозможно заблокировать кредитную карту, у которой не превышен лимит");  
 }  
  
 creditCard.block();  
 }  
  
 protected CustomerForAdmin getCustomer(int customerId) throws Exception {  
 return (CustomerForAdmin) \_paymentSystem.getCustomer(customerId);  
 }  
}

Листинг 31 – Задание 7 (11)

package PaymentSystem;  
  
import Customer.Customer;  
import Order.Order;  
  
import java.math.BigDecimal;  
  
public class CustomerInterface {  
 private final Customer \_customer;  
 private PaymentSystem \_paymentSystem;  
  
 protected CustomerInterface(Customer customer, PaymentSystem paymentSystem) {  
 \_customer = customer;  
 \_paymentSystem = paymentSystem;  
 }  
  
 public void createAccount(int accountId, BigDecimal balance) {  
 \_customer.createAccount(accountId, balance);  
 \_paymentSystem.addAccountId(\_customer.Id, accountId);  
 }  
  
 public void createCreditCard(int creditCardId, BigDecimal balance) {  
 \_customer.createCreditCard(creditCardId, balance);  
 }  
  
 public void payOrder(Order order, int sourceAccountId) throws Exception {  
 var account = \_customer.getAccount(sourceAccountId);  
  
 var operationDetails = String.format("Оплата заказа %d", order.Id);  
  
 account.decreaseBalanceBy(order.TotalAmount, operationDetails);  
 }  
  
 public void transferMoney(int sourceAccountId, int targetAccountId, BigDecimal amount) throws Exception {  
 var sourceAccount = \_customer.getAccount(sourceAccountId);  
  
 var targetCustomerId = \_paymentSystem.getUserIdByAccountId(targetAccountId);  
 var targetAccount = \_paymentSystem.getCustomer(targetCustomerId).getAccount(targetAccountId);  
  
 var operationDetails = String.format("Перевод между счетами %d и %d", sourceAccountId, targetAccountId);  
  
 sourceAccount.decreaseBalanceBy(amount, operationDetails);  
 targetAccount.increaseBalanceBy(amount, operationDetails);  
 }  
  
 public void disableAccount(int accountId) throws Exception {  
 \_customer.getAccount(accountId).disable();  
 }  
  
 public void blockCreditCard(int creditCardId) throws Exception {  
 \_customer.getCreditCard(creditCardId).block();  
 }  
}

Листинг 32 – Задание 7 (12)

package PaymentSystem;  
  
import Customer.Customer;  
import Customer.CustomerForCustomer;  
  
import Admin.Admin;  
  
import java.util.HashMap;  
  
public class PaymentSystem {  
 private int \_customerIdSequence;  
 private int \_adminIdSequence;  
  
 private final HashMap<Integer, Customer> \_customers;  
 private final HashMap<Integer, Integer> \_accountIdToCustomerIdMap;  
  
 private final HashMap<Integer, Admin> \_admins;  
  
 public PaymentSystem() {  
 \_customerIdSequence = 1;  
 \_adminIdSequence = 1;  
  
 \_customers = new HashMap<>();  
 \_accountIdToCustomerIdMap = new HashMap<>();  
  
 \_admins = new HashMap<>();  
 }  
  
 public int createCustomer() {  
 var customerId = \_customerIdSequence++;  
 var customer = new Customer(customerId);  
  
 \_customers.put(customerId, customer);  
  
 return customerId;  
 }  
  
 public int createAdmin() {  
 var adminId = \_adminIdSequence++;  
 var admin = new Admin(adminId);  
  
 \_admins.put(adminId, admin);  
  
 return adminId;  
 }  
  
 public CustomerInterface createCustomerInterface(int customerId) throws Exception {  
 if (!\_customers.containsKey(customerId)) {  
 throw new Exception("Не удалось найти пользователя");  
 }  
  
 var customer = \_customers.get(customerId);  
  
 return new CustomerInterface(customer, this);  
 }  
  
 public AdminInterface createAdminInterface(int adminId) throws Exception {  
 if (!\_admins.containsKey(adminId)) {  
 throw new Exception("Не удалось найти администратора");  
 }  
  
 var admin = \_admins.get(adminId);  
  
 return new AdminInterface(admin, this);  
 }  
  
 protected CustomerForCustomer getCustomer(int customerId) throws Exception {  
 if (!\_customers.containsKey(customerId)) {  
 throw new Exception("Не удалось найти пользователя");  
 }  
  
 return \_customers.get(customerId);  
 }  
  
 protected int getUserIdByAccountId(int accountId) throws Exception {  
 if (!\_accountIdToCustomerIdMap.containsKey(accountId)) {  
 throw new Exception("Не удалось определить клиента по номеру счета");  
 }  
  
 return \_accountIdToCustomerIdMap.get(accountId);  
 }  
  
 protected void addAccountId(int customerId, int accountId) {  
 \_accountIdToCustomerIdMap.put(accountId, customerId);  
 }  
}

**Задание 8:** Система Больница. Пациенту назначается лечащий Врач. Врач может сделать назначение Пациенту (процедуры, лекарства, операции). Медсестра или другой Врач выполняют назначение. Пациент может быть выписан из Больницы по окончании лечения, при нарушении режима или при иных обстоятельствах.

Код решения приведен в листингах 33 - 40.

Листинг 33 – Задание 8 (1)

import Hospital.\*;  
  
public class Main {  
 public static void main(String[] args) throws Exception {  
 var hospitalSystem = new HospitalSystem();  
  
 var doctorId = hospitalSystem.createDoctor();  
 var nurseId = hospitalSystem.createNurse();  
  
 // Назначение пациенту лечащего врача  
 var patientId = hospitalSystem.createPatient(doctorId);  
  
 DoctorInterface doctorInterface = hospitalSystem.createDoctorInterface(doctorId);  
  
 // Выдача пациенту назначений  
 var prescriptionId1 = doctorInterface.givePrescription(patientId, Prescription.Type.*Procedure*);  
 var prescriptionId2 = doctorInterface.givePrescription(patientId, Prescription.Type.*Surgery*);  
  
 NurseInterface nurseInterface = hospitalSystem.createNurseInterface(nurseId);  
  
 // Выполнение выданных назначений  
 nurseInterface.performPrescription(patientId, prescriptionId1);  
 doctorInterface.performPrescription(patientId, prescriptionId2);  
  
 // Выписка пациента из больницы  
 doctorInterface.dischargePatient(patientId, DischargeReason.*EndOfTreatment*);  
 }  
}

Листинг 34 – Задание 8 (2)

package Hospital;  
  
public enum DischargeReason {  
 *EndOfTreatment*,  
 *RestViolation*,  
 *Other*}

Листинг 35 – Задание 8 (3)

package Hospital;  
  
public class DoctorInterface {  
 private final SeniorMedicalStaffMember \_staffMember;  
 private final HospitalSystem \_hospitalSystem;  
  
 DoctorInterface(SeniorMedicalStaffMember staffMember, HospitalSystem hospitalSystem) {  
 \_staffMember = staffMember;  
 \_hospitalSystem = hospitalSystem;  
 }  
  
 public int givePrescription(int patientId, Prescription.Type prescriptionType) throws Exception {  
 var assignedDoctorId = \_hospitalSystem.getDoctorIdByPatientId(patientId);  
  
 if (\_staffMember.Id != assignedDoctorId) {  
 throw new Exception("Пациенту назначен другой врач");  
 }  
  
 var patient = \_hospitalSystem.getPatient(patientId);  
  
 return \_staffMember.givePrescription(patient, prescriptionType);  
 }  
  
 public void performPrescription(int patientId, int prescriptionId) throws Exception {  
 var patient = \_hospitalSystem.getPatient(patientId);  
  
 \_staffMember.performPrescription(patient, prescriptionId);  
 }  
  
 public void dischargePatient(int patientId, DischargeReason reason) throws Exception {  
 var assignedDoctorId = \_hospitalSystem.getDoctorIdByPatientId(patientId);  
  
 if (\_staffMember.Id != assignedDoctorId) {  
 throw new Exception("Пациенту назначен другой врач");  
 }  
  
 var patient = \_hospitalSystem.getPatient(patientId);  
  
 patient.discharge(reason);  
 }  
}

Листинг 36 – Задание 8 (4)

package Hospital;  
  
import java.util.HashMap;  
  
public class HospitalSystem {  
 private int \_medicalStaffMemberIdSequence;  
 private int \_patientIdSequence;  
  
 private HashMap<Integer, MedicalStaffMember> \_medicalStaff;  
 private HashMap<Integer, Patient> \_patients;  
 private HashMap<Integer, Integer> \_patientIdToDoctorIdMap;  
  
 public HospitalSystem() {  
 \_medicalStaff = new HashMap<>();  
 \_patients = new HashMap<>();  
 \_patientIdToDoctorIdMap = new HashMap<>();  
 \_medicalStaffMemberIdSequence = 1;  
 \_patientIdSequence = 1;  
 }  
  
 public int createPatient(int doctorId) throws Exception {  
 var medicalStaffMember = \_medicalStaff.get(doctorId);  
  
 if (!(medicalStaffMember instanceof SeniorMedicalStaffMember doctor)) {  
 throw new Exception("Не удалось найти врача");  
 }  
  
 var patient = new Patient(\_patientIdSequence++);  
  
 \_patients.put(patient.Id, patient);  
 \_patientIdToDoctorIdMap.put(patient.Id, doctor.Id);  
  
 return patient.Id;  
 }  
  
 public int createDoctor() {  
 var doctor = new SeniorMedicalStaffMember(\_medicalStaffMemberIdSequence++);  
  
 \_medicalStaff.put(doctor.Id, doctor);  
  
 return doctor.Id;  
 }  
  
 public int createNurse() {  
 var nurse = new JuniorMedicalStaffMember(\_medicalStaffMemberIdSequence++);  
  
 \_medicalStaff.put(nurse.Id, nurse);  
  
 return nurse.Id;  
 }  
  
 public DoctorInterface createDoctorInterface(int doctorId) throws Exception {  
 var staffMember = \_medicalStaff.get(doctorId);  
  
 if (!(staffMember instanceof SeniorMedicalStaffMember seniorMedicalStaffMember)) {  
 throw new Exception("Не удалось найти врача");  
 }  
  
 return new DoctorInterface(seniorMedicalStaffMember, this);  
 }  
  
 public NurseInterface createNurseInterface(int nurseId) throws Exception {  
 var staffMember = \_medicalStaff.get(nurseId);  
  
 if (!(staffMember instanceof JuniorMedicalStaffMember juniorMedicalStaffMember)) {  
 throw new Exception("Не удалось найти врача");  
 }  
  
 return new NurseInterface(juniorMedicalStaffMember, this);  
 }  
  
 Patient getPatient(int patientId) throws Exception {  
 if (!\_patients.containsKey(patientId)) {  
 throw new Exception("Не удалось найти пациента");  
 }  
  
 return \_patients.get(patientId);  
 }  
  
 int getDoctorIdByPatientId(int patientId) {  
 return \_patientIdToDoctorIdMap.get(patientId);  
 }  
}

Листинг 37 – Задание 8 (5)

package Hospital;  
  
import java.util.Arrays;  
  
abstract class MedicalStaffMember {  
 public final int Id;  
  
 public MedicalStaffMember(int id) {  
 Id = id;  
 }  
}  
  
class JuniorMedicalStaffMember extends MedicalStaffMember {  
 public JuniorMedicalStaffMember(int id) {  
 super(id);  
 }  
  
 public void performPrescription(Patient patient, int prescriptionId) throws Exception {  
 var prescription = patient.getPrescription(prescriptionId);  
  
 prescription.markAsDone();  
 }  
}  
  
class SeniorMedicalStaffMember extends JuniorMedicalStaffMember {  
 public SeniorMedicalStaffMember(int id) {  
 super(id);  
 }  
  
 public int givePrescription(Patient patient, Prescription.Type prescriptionType) throws Exception {  
 var patientPrescriptions = patient.getPrescriptions();  
  
 var newPrescriptionId = Arrays.*stream*(patientPrescriptions)  
 .mapToInt(prescription -> prescription.Id)  
 .max().orElse(0) + 1;  
 var newPrescription = new Prescription(newPrescriptionId, prescriptionType);  
  
 patient.addPrescription(newPrescription);  
  
 return newPrescriptionId;  
 }  
}

Листинг 38 – Задание 8 (6)

package Hospital;  
  
public class NurseInterface {  
 private final JuniorMedicalStaffMember \_staffMember;  
 private final HospitalSystem \_hospitalSystem;  
  
 NurseInterface(JuniorMedicalStaffMember staffMember, HospitalSystem hospitalSystem) {  
 \_staffMember = staffMember;  
 \_hospitalSystem = hospitalSystem;  
 }  
  
 public void performPrescription(int patientId, int prescriptionId) throws Exception {  
 var patient = \_hospitalSystem.getPatient(patientId);  
  
 \_staffMember.performPrescription(patient, prescriptionId);  
 }  
}

Листинг 39 – Задание 8 (7)

package Hospital;  
  
import java.util.HashMap;  
  
class Patient {  
 private final HashMap<Integer, Prescription> \_prescriptions;  
 private boolean \_discharged;  
 private DischargeReason \_dischargeReason;  
  
 public final int Id;  
  
 public Patient(int id) {  
 Id = id;  
 \_prescriptions = new HashMap<>();  
 \_discharged = false;  
 \_dischargeReason = null;  
 }  
  
 public boolean isDischarged() {  
 return \_discharged;  
 }  
  
 public DischargeReason getDischargeReason() throws Exception {  
 if (!\_discharged) {  
 throw new Exception("Пациент не был выписан");  
 }  
  
 return \_dischargeReason;  
 }  
  
 public void discharge(DischargeReason reason) {  
 \_discharged = true;  
 \_dischargeReason = reason;  
 }  
  
 public Prescription[] getPrescriptions() {  
 return \_prescriptions.values().toArray(Prescription[]::new);  
 }  
  
 public Prescription getPrescription(int prescriptionId) throws Exception {  
 if (!\_prescriptions.containsKey(prescriptionId)) {  
 throw new Exception("У пациента нет такого назначения");  
 }  
  
 return \_prescriptions.get(prescriptionId);  
 }  
  
 public void addPrescription(Prescription prescription) throws Exception {  
 if (\_discharged) {  
 throw new Exception("Пациент был выписан");  
 }  
  
 \_prescriptions.put(prescription.Id, prescription);  
 }  
}

Листинг 40 – Задание 8 (8)

package Hospital;  
  
public class Prescription {  
 private boolean \_done;  
 private final Type \_type;  
  
 public final int Id;  
  
 protected Prescription(int id, Type type) {  
 Id = id;  
 \_type = type;  
 \_done = false;  
 }  
  
 public boolean isDone() {  
 return \_done;  
 }  
  
 public Type getType() {  
 return \_type;  
 }  
  
 public void markAsDone() {  
 \_done = true;  
 }  
  
 public enum Type {  
 *Procedure*,  
 *Medicine*,  
 *Surgery* }  
}

**Вывод:** в результате выполнения лабораторной работы были разработаны несколько программ в соответствие с требованиями задания.