

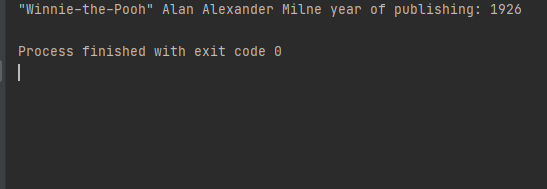
**Первая практика**

Упражнение 3. Реализуйте простейший класс «Книга»

public class Main {  
  
 public static void main(String[] args) {  
 BookPr1 b1 = new BookPr1("Alan Alexander Milne", "Winnie-the-Pooh", 1926);  
 BookPr1 b2 = new BookPr1("HJ.K. Rowling", "Harry Potter and the Philosopher's Stone");  
 BookPr1 b3 = new BookPr1("Arthur Conan Doyle");  
 BookPr1 b4 = new BookPr1();  
  
 b3.setTitle("The Adventures of Sherlock Holmes");  
 System.out.println(b1);  
 }  
}

public class BookPr1 {  
 private String author;  
 private String title;  
 private int year\_of\_publish;  
  
 public BookPr1(String a, String t, int year)  
 {  
 author = a;  
 title = t;  
 year\_of\_publish = year;  
 }  
  
 public BookPr1(String a, String t)  
 {  
 author = a;  
 title = t;  
 year\_of\_publish = 0;  
 }  
  
 public BookPr1(String a)  
 {  
 author = a;  
 title = "Test title";  
 year\_of\_publish = 0;  
 }  
  
 public BookPr1()  
 {  
 author = "Test author";  
 title = "Test title";  
 year\_of\_publish = 0;  
 }  
  
 public void setAuthor(String author) { this.author = author; }  
 public void setTitle(String title) { this.title = title; }  
 public void setYear\_of\_publish(int year) { this.year\_of\_publish = year; }  
  
 public String getAuthor() { return author; }  
 public String getTitle() { return title; }  
 public int getYear\_of\_publish() { return year\_of\_publish; }  
  
 public String toString() {  
 return "\"" + this.title + "\" " + this.author + " year of publishing: " + this.year\_of\_publish;  
 }  
}

Вывод программы:



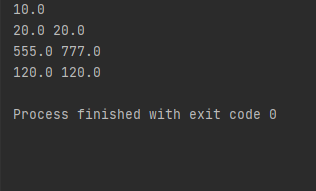
**Вторая практика**

Упражнение 2. По UML диаграмме класса, представленной на рисунке 2.5. написать программу, которая состоит из двух классов. Один из них Ball должен реализовывать сущность мяч, а другой с названием TestBall тестировать работу созданного класса. Класс Ball должен содержать реализацию методов, представленных на UML. Диаграмма на рисунке описывает сущность Мяч написать программу.

public class TestBall {  
 public static void main(String[] args) {  
 Ball b1 = new Ball(10, 10);  
 Ball b2 = new Ball();  
  
 System.*out*.println(b1.getX());  
 b1.setXY(20, 20);  
 System.*out*.println(b1.getX() + " " + b1.getY());  
 b2.setXY(555, 777);  
 System.*out*.println(b2.getX() + " " + b2.getY());  
  
 b1.move(100, 100);  
 System.*out*.println(b1.getX() + " " + b1.getY());  
  
 }  
}

public class Ball {  
 private double x;  
 private double y;  
  
 public Ball(double x, double y)  
 {  
 this.x = x;  
 this.y = y;  
 }  
  
 public Ball()  
 {  
 x = 0.0;  
 y = 0.0;  
 }  
  
 public double getX() { return x; }  
 public double getY() { return y; }  
  
 public void setX(double x) { this.x = x; }  
 public void setY(double y) { this.y = y; }  
  
 public void setXY(double x, double y)  
 {  
 this.x = x;  
 this.y = y;  
 }  
  
 public void move(double xDisp, double yDisp)  
 {  
 this.x += xDisp;  
 this.y += yDisp;  
 }  
  
 public String toString() {  
 return "Ball @" +  
 "(" + x +  
 ", " + y +  
 ')';  
 }  
}

**Вывод программы:**

****

**Третья практика**

Задача: Напишите два класса MovablePoint и MovableCircle - которые реализуют интерфейс Movable.

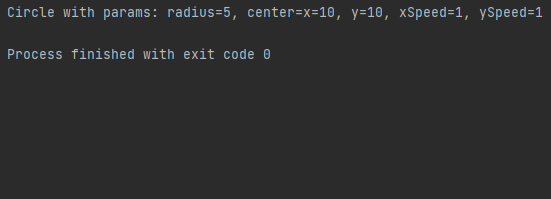
public class MovableTest {  
 public static void main(String[] args) {  
 MovableCircle circle = new MovableCircle(10, 10, 1, 1, 5);  
 System.*out*.println(circle);  
 }  
}

public class MovablePoint implements Movable{  
 private int x;  
 private int y;  
 private int xSpeed;  
 private int ySpeed;  
  
 MovablePoint(int x, int y, int xSpeed, int ySpeed)  
 {  
 this.x = x;  
 this.y = y;  
 this.xSpeed = xSpeed;  
 this.ySpeed = ySpeed;  
 }  
  
 public String toString() {  
 return "x=" + x +  
 ", y=" + y +  
 ", xSpeed=" + xSpeed +  
 ", ySpeed=" + ySpeed;  
 }  
  
 public void moveUp() {  
 y += ySpeed;  
 }  
  
 public void moveDown() {  
 y -= ySpeed;  
 }  
  
 public void moveLeft() {  
 x -= xSpeed;  
 }  
  
 public void moveRight() {  
 x += xSpeed;  
 }  
}

public class MovableCircle implements Movable{  
 int radius;  
 MovablePoint center;  
  
 MovableCircle(int x, int y, int xSpeed, int ySpeed, int radius){  
 center = new MovablePoint(x, y, xSpeed, ySpeed);  
 this.radius = radius;  
 }  
  
 public String toString() {  
 return "Circle with params: " +  
 "radius=" + radius +  
 ", center=" + center;  
 }  
  
 public void moveUp() {  
 center.moveUp();  
 }  
  
 public void moveDown() {  
 center.moveDown();  
 }  
  
 public void moveLeft() {  
 center.moveLeft();  
 }  
  
 public void moveRight() {  
 center.moveRight();  
 }  
}

public interface Movable {  
 void moveUp();  
 void moveDown();  
 void moveLeft();  
 void moveRight();  
}

Вывод программы:



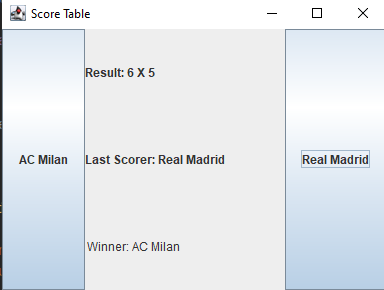
**Чётвертая практика**

Упражнение 1.

Напишите интерактивную программу с использованием GUI имитирует таблицу результатов матчей между командами Милан и Мадрид. Создайте JFrame приложение у которого есть следующие компоненты GUI: • одна кнопка JButton labeled “AC Milan” • другая JButton подписана “Real Madrid” • надпись JLabel содержит текст “Result: 0 X 0” • надпись JLabel содержит текст “Last Scorer: N/A” • надпись Label содержит текст “Winner: DRAW”; Всякий раз, когда пользователь нажимает на кнопку AC Milan, результат будет увеличиваться для Милана, сначала 1 X 0, затем 2 X 0 и так далее. Last Scorer означает последнюю забившую команду. В этом случае: AC Milan. Если пользователь нажимает кнопку для команды Мадрид, то счет приписывается ей. Победителем становится команда, которая имеет больше кликов кнопку на соответствующую, чем другая.

public class ScoreTable extends JFrame {  
 private JButton button1 = new JButton("AC Milan");  
 private JButton button2 = new JButton("Real Madrid");  
  
 private JLabel label1 = new JLabel("Result: 0 X 0");  
 private JLabel label2 = new JLabel("Last Scorer: N/A");  
  
 private Label winLabel = new Label("Winner: DRAW");  
  
 private int firstCommandScore = 0;  
 private int secondCommandScore = 0;  
  
  
 public ScoreTable(){  
 super("Score Table");  
 setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
 setSize(400, 300);  
 setLayout(new BorderLayout());  
  
 button1.addActionListener(new ButtonEventListener());  
 button2.addActionListener(new ButtonEventListener());  
  
  
 add(button1, BorderLayout.*WEST*);  
  
 JPanel container = new JPanel();  
 container.setLayout(new GridLayout(3,1,2,2));  
 container.add(label1);  
 container.add(label2);  
 container.add(winLabel);  
 add(container, BorderLayout.*CENTER*);  
  
 add(button2, BorderLayout.*EAST*);  
 }  
  
 class ButtonEventListener implements ActionListener {  
 public void actionPerformed(ActionEvent e) {  
 String message = "Result: ";  
 String lastGoal = "Last Scorer: ";  
 String winner = "Winner: ";  
  
 if (e.getActionCommand().equals("AC Milan"))  
 firstCommandScore++;  
 else  
 secondCommandScore++;  
 message += firstCommandScore + " X " + secondCommandScore;  
 label1.setText(message);  
  
 lastGoal += e.getActionCommand();  
 label2.setText(lastGoal);  
  
 if (firstCommandScore > secondCommandScore)  
 winner += "AC Milan";  
 else  
 winner += "Real Madrid";  
 winLabel.setText(winner);  
 }  
 }  
  
 public static void main(String[] args)  
 {  
 ScoreTable app = new ScoreTable();  
 app.setVisible(true);  
 }  
}

**Вывод программы:**

****

**Пятая практика**

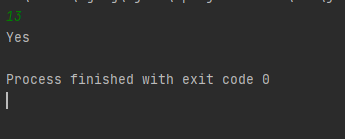
**Задача:**

6. Проверка числа на простоту Дано натуральное число n>1. Проверьте, является ли оно простым. Программа должна вывести слово YES, если число простое и NO, если число составное. Алгоритм должен иметь сложность O(logn).

8. Палиндром Дано слово, состоящее только из строчных латинских букв. Проверьте, является ли это слово палиндромом. Выведите YES или NO.

public class PrimeNumber {  
 static public boolean IsPrime(int n,int k) {  
 if (n==1 || k > Math.*sqrt*(n))  
 return true;  
 if (n % k == 0)  
 return false;  
 return *IsPrime*(n,k+1);  
 }  
  
 static boolean checkPalindrome(String word) {  
 if (word.length() < 2)  
 return true;  
 char first = word.charAt(0);  
 char last = word.charAt(word.length()-1);  
 if (first != last)  
 return false;  
 else  
 return *checkPalindrome*(word.substring(1,word.length()-1));  
 }  
  
  
  
 public static void main(String[] args) {  
 Scanner in = new Scanner(System.*in*);  
 int n = in.nextInt();  
  
 if (*IsPrime*(n, 2))  
 System.*out*.println("Yes");  
 else  
 System.*out*.println("No");  
 in.close();  
  
 }  
}

Вывод программы:



**Шестая практика**

Задча:

Написать тестовый класс, который создает массив класса Student и сортирует массив iDNumber и сортирует его вставками.

public class Main {  
  
 public static void main(String[] args) throws StudentNameException {  
 ArrayList<Student> arrayList = new ArrayList<>(20);  
  
 /\*  
 \* Есть массив студентов, нужно иметь возможность его отсортировать (реализована быстрая сортировка)  
 \* и найти балл по ФИО студента, если ФИО не найдено, выдавать исключение.  
 \* \*/  
  
  
 for (int i = 0; i < 5; i++){  
 arrayList.add(new Student("Test " + i, (int)(Math.*random*() \* 10 + 20)));  
 }  
 System.*out*.println("Неотсортированный список студентов: ");  
 for (int i = 0; i < 5; i++){  
 System.*out*.println(arrayList.get(i).getName() + " " + arrayList.get(i).getSumOrder());  
 }  
  
 Student.sort\_students(arrayList);  
  
 System.*out*.println("Отсортированный список студентов: ");  
 for (int i = 0; i < 5; i++){  
 System.*out*.println(arrayList.get(i).getName() + " " + arrayList.get(i).getSumOrder());  
 }  
  
 System.*out*.println("\nПолучение баллов студента Test 3: ");  
 Student.find\_student(arrayList, "Test 3");  
 Student.find\_student(arrayList, "Test 223");  
 }  
}

public interface I\_FindSort {  
 static void find\_student(ArrayList<Student> listStudents, String student\_name) throws StudentNameException { }  
 static void sort\_students(ArrayList<Student> listStudents) { }  
}

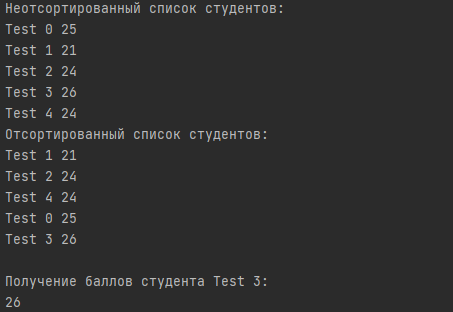
public class StudentNameException extends Exception{  
 public StudentNameException() { super("Invalid name."); }  
 public StudentNameException(String message) { super("Invalid name. Message: '" + message + " '."); };  
}

public class StudentComparator implements Comparator<Student> {  
 public int compare(Student o1, Student o2) {  
 return o1.getSumOrder() - o2.getSumOrder();  
 }  
}

public class Student implements I\_FindSort {  
 private String name;  
 private int sumOrder;  
  
 public Student(String name, int sumOrder){  
 this.name = name;  
 this.sumOrder = sumOrder;  
 }  
  
 public int getSumOrder() { return sumOrder; }  
 public String getName() { return name; }  
  
 public void setName(String name) { this.name = name; }  
 public void setSumOrder(int sumOrder) { this.sumOrder = sumOrder; }  
  
 static void find\_student(ArrayList<Student> listStudents, String student\_name) throws StudentNameException {  
 for (Student student : listStudents) {  
 if (student.getName().equals(student\_name)) {  
 System.*out*.println(student.getSumOrder());  
 return;  
 }  
 }  
 throw new StudentNameException();  
 }  
  
 static void sort\_students(ArrayList<Student> listStudents)  
 {  
 QuickSort qs = new QuickSort();  
 qs.quickSort(listStudents, 0, listStudents.size() - 1);  
 }  
}

public class QuickSort {  
 public void quickSort(ArrayList<Student> arr, int from, int to) {  
 if (from < to) {  
 int divideIndex = *partition*(arr, from, to);  
  
 quickSort(arr, from, divideIndex - 1);  
 quickSort(arr, divideIndex, to);  
 }  
 }  
  
 private static int partition(ArrayList<Student> arr, int from, int to) {  
 StudentComparator scomp = new StudentComparator();  
 int indexLeft = from;  
 int indexRight = to;  
  
 Student pivot = arr.get(from + (to - from) / 2);  
 while (indexLeft <= indexRight) {  
  
 while (scomp.compare(arr.get(indexLeft), pivot) < 0) {  
 indexLeft++;  
 }  
  
 while (scomp.compare(arr.get(indexRight), pivot) > 0) {  
 indexRight--;  
 }  
  
 if (indexLeft <= indexRight) {  
 *swap*(arr, indexLeft, indexRight);  
 indexLeft++;  
 indexRight--;  
 }  
 }  
  
 return indexLeft;  
 }  
  
 private static void swap(ArrayList<Student> arr, int indexLeft, int indexRight)  
 {  
 String tmp\_name;  
 int tmp\_orders;  
  
 // Меняем местами поля двух объектов-студентов.  
 tmp\_name = arr.get(indexLeft).getName();  
 tmp\_orders = arr.get(indexLeft).getSumOrder();  
  
 arr.get(indexLeft).setName(arr.get(indexRight).getName());  
 arr.get(indexLeft).setSumOrder(arr.get(indexRight).getSumOrder());  
  
 arr.get(indexRight).setName(tmp\_name);  
 arr.get(indexRight).setSumOrder(tmp\_orders);  
 }  
}

**Вывод программы:**

****

**Седьмая практика**

**Задача:**

Напишите программу в виде консольного приложения, которая моделирует карточную игру «пьяница» и определяет, кто выигрывает. В игре участвует 10 карт, имеющих значения от 0 до 9, большая карта побеждает меньшую; карта «0» побеждает карту «9»

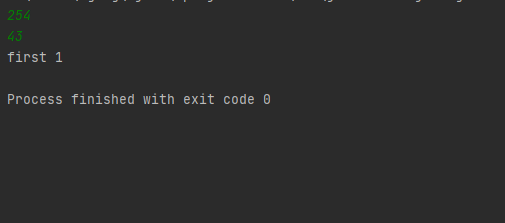
public class Main {  
 public static void main(String[] args) {  
 Scanner sc = new Scanner(System.*in*);  
  
 String[] s1 = sc.nextLine().split(" ");  
 String[] s2 = sc.nextLine().split(" ");  
  
 Game game = new Game(s1, s2);  
 game.startGame();  
 }  
}

public class Game {  
 private SimpleQueue<Integer> q1 = new SimpleQueue<>();  
 private SimpleQueue<Integer> q2 = new SimpleQueue<>();  
  
 public Game(String[] s1, String[] s2) {  
 for (int i = 0; i < s1.length; i++){  
 q1.add(Integer.*parseInt*(s1[i]));  
 q2.add(Integer.*parseInt*(s2[i]));  
 }  
 }  
  
 public void startGame() {  
 int cardFromFirst;  
 int cardFromSecond;  
 int amountTurn = 0;  
  
 while (!q1.isEmpty() && !q2.isEmpty() && amountTurn < 106) {  
 cardFromFirst = q1.remove();  
 cardFromSecond = q2.remove();  
  
 if (cardFromFirst > cardFromSecond) {  
 q1.add(cardFromFirst);  
 q1.add(cardFromSecond);  
 }  
 else {  
 q2.add(cardFromSecond);  
 q2.add(cardFromFirst);  
 }  
 amountTurn++;  
 }  
 if (q1.isEmpty())  
 System.*out*.println("second " + amountTurn);  
 else if (q2.isEmpty())  
 System.*out*.println("first " + amountTurn);  
 else  
 System.*out*.println("botva");  
 }  
}

public class SimpleQueue<T> implements Queue<T> {  
  
 private ArrayList<T> arrayList = new ArrayList<>();  
  
 public void add(T item) {  
 arrayList.add(item);  
 }  
  
 public T remove() {  
 return arrayList.remove(0);  
 }  
  
 public boolean isEmpty() {  
 return arrayList.isEmpty();  
 }  
}

public interface Queue<T> {  
 void add(T item);  
 T remove();  
 boolean isEmpty();  
}

**Вывод программы:**

**-**

**Восьмая практика**

**Задача:**

1. Исследуйте UML диаграмму классов на рисунке 1 и понаблюдайте, как она выражает то, что мы говорили выше в словах. Убедитесь, что вы понимаете все аспекты диаграммы.

2. Расширить и модифицировать исходный код WaitList, как необходимо, чтобы полностью реализовать всю схему UML. Включить комментарии Javadoc. Обратите внимание на переключение ролей после реализации каждого интерфейса / класса!

3. Изучение работу метода main(), которая использует ваши новые классы и интерфейс

public class Main {  
 public static void main(String[] args) {  
  
 IWaitList<Integer> test1 = new WaitList<>();  
 test1.add(10);  
 test1.add(20);  
 test1.add(30);  
 System.*out*.println(test1);  
 test1.remove();  
 System.*out*.println(test1);  
 System.*out*.println("WaitList content contains 30? Answer: " + test1.contains(30) + "\n");  
  
 IWaitList<Integer> test2 = new BoundedWaitList<>(10);  
 for (int i = 0; i < 11; i++)  
 {  
 test2.add(i);  
 }  
 System.*out*.println(test2 + "\n");  
  
 UnfairWaitList<String> test3 = new UnfairWaitList<>();  
 test3.add("A");  
 test3.add("B");  
 test3.add("C");  
 System.*out*.println(test3);  
 test3.moveToBack();  
 System.*out*.println(test3);  
 }  
  
}

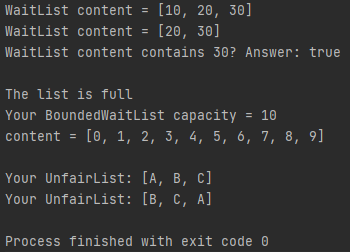
public interface IWaitList<T> {  
 void add(T element);  
 T remove();  
 boolean contains(T element);  
 boolean containsAll(Collection<T> c);  
 boolean isEmpty();  
}

public class UnfairWaitList<T> extends WaitList<T>{  
 UnfairWaitList(){  
 super();  
 }  
  
 public void remove(T element){  
 if (element != null)  
 content.remove(element);  
 }  
  
 public void moveToBack(){  
 T peek = super.content.remove();  
 super.content.add(peek);  
 }  
  
 public String toString(){  
 return "Your UnfairList: " + content;  
 }  
}

public class WaitList<T> implements IWaitList<T> {  
 protected ConcurrentLinkedQueue<T> content;  
  
 public WaitList(){  
 content = new ConcurrentLinkedQueue<>();  
 }  
 public WaitList(Collection<T> c){ this.content = (ConcurrentLinkedQueue<T>) c; }  
  
 public String toString() { return "WaitList content = " + content; }  
 public void add(T element) { content.add(element); }  
 public T remove() { return content.remove(); }  
 public boolean contains(T element) {  
 return content.contains(element);  
 }  
 public boolean containsAll(Collection<T> c) { return content.containsAll(c); }  
 public boolean isEmpty() { return content.isEmpty(); }  
}

public class BoundedWaitList<T> extends WaitList<T> {  
 private int capacity;  
  
 public BoundedWaitList(int capacity)  
 {  
 super();  
 this.capacity = capacity;  
 }  
 public int getCapacity() {  
 return capacity;  
 }  
  
 public void add(T element) {  
 if (super.content.size() < capacity)  
 super.add(element);  
 else  
 System.*out*.println("The list is full");  
 }  
  
 public String toString() {  
 return "Your BoundedWaitList " +  
 "capacity = " + capacity +  
 "\ncontent = " + content;  
 }  
}

**Вывод программы:**

****

**Девятая практика**

**Задача:**

Клиент совершает покупку онлайн. При оформлении заказа у пользователя запрашивается фио и номер ИНН. В программе проверяется, действителен ли номер ИНН для такого клиента. Исключение будет выдано в том случае, если введен недействительный ИНН.

Предлагается модернизировать задачу из предыдущей лабораторной работы (см. методические указания по выполнению лабораторных работ №1-8) – задача сортировки студентов по среднему баллу. Необходимо разработать пользовательский интерфейс для задачи поиска и сортировки (использовать массив интерфейсных ссылок- пример в лекции 5). Дополнить ее поиском студента по фио – в случае отсутствия такого студента необходимо выдавать собственное исключение

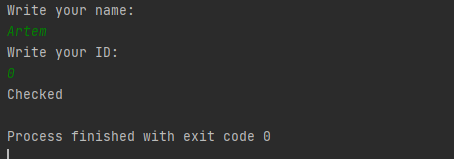
public class Main {  
  
 public static void main(String[] args) {  
  
 PersonBase personBase = new PersonBase();  
 Scanner sc = new Scanner(System.*in*);  
 String textField;  
 System.*out*.println("Write your name: ");  
 String name = sc.nextLine();  
 System.*out*.println("Write your ID: ");  
 Integer ID = sc.nextInt();  
 try{  
 personBase.check\_ID(name, ID);  
 } catch (NotFoundEx err){  
 System.*err*.println(err.getMessage());  
  
 } catch (NotCorrectIDEx err){  
 System.*err*.println(err.getMessage());  
 }  
 }  
}

public class NotCorrectIDEx extends RuntimeException{  
  
 public NotCorrectIDEx(){  
 super("Wrong person ID");  
 }  
}

public class NotFoundEx extends RuntimeException {  
  
 public NotFoundEx(){  
 super("Person is not found in Base");  
 }  
}

public class PersonBase {  
 HashMap<String, Integer> Base;  
 String[] names = {  
 "Artem",  
 "Yaroslav",  
 "Petr",  
 "Ilya",  
 "Anastasia",  
 "Dmitriy",  
 "Pavel",  
 "Konstantin",  
 "Grigoriy",  
 "Maria",  
 "Lize",  
 "Katya",  
 "Nadejda",  
 "Michail"  
  
 };  
  
 public void check\_ID(String name, int ID) throws NotCorrectIDEx, NotFoundEx{  
 if (Base.containsKey(name)){  
 if (Base.get(name)==ID){  
 System.*out*.println("Checked");  
 }  
 else{  
 throw new NotCorrectIDEx();  
 }  
 }  
 else{  
 throw new NotFoundEx();  
 }  
 }  
  
 public PersonBase(){  
 Base = new HashMap<>();  
 for (int i = 0; i < names.length; i++) {  
 Base.put(names[i], i);  
 }  
 }  
}

Вывод программы:



**Десятая практика**

**Задача:**

Упражнение 1

Реализовать класс Абстрактная фабрика для комплексных чисел

Упражнение 2

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

Упражнение 3\*

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

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

• При нажатии пунктов меню File -> New и File -> Open требуется создать новый экземпляр документа (конкретного подкласса). Однако каркас не должен быть привязан ни к какому конкретному виду документов.

• Нужно создать фабричный интерфейс ICreateDocument. Этот интерфейс содержит два абстрактных фабричных метода: CreateNew и CreateOpen, оба возвращают экземпляр IDocument

• Каркас оперирует одним экземпляром IDocument и одним экземпляром ICreateDocument. Какие конкретные классы будут подставлены сюда, определяется во время запуска приложения.

Требуется:

1. создать перечисленные классы. Создать каркас приложения — окно редактора с меню File. В меню File реализовать пункты New, Open, Save, Exit

2. продемонстрировать работу каркаса на примере текстового редактора. Потребуется создать конкретный унаследованный класс TextDocument и фабрику для него — CreateTextDocument.

Упражнение 1

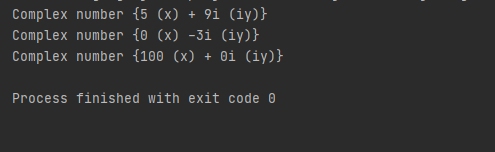
public class Main {  
 public static void main(String[] args) {  
 // Создание объекта класса Complex  
 Complex a = new Complex(5, 9);  
 System.*out*.println(a);  
  
 // Наследуем от абстрактной фабрики конкретную, и создаем новый конкретный объект  
 ComplexAbstractFactory b = new ConcreteFactory();  
 Complex c = b.createComplex(0, -3);  
 System.*out*.println(c);  
  
 ConcreteFactory d = new ConcreteFactory();  
 Complex f = d.createComplex(100, 0);  
 System.*out*.println(f);  
 }  
}

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

public class ConcreteFactory implements ComplexAbstractFactory {  
 @Override  
 public Complex createComplex() {  
 return new Complex(0, 0);  
 }  
  
 @Override  
 public Complex createComplex(int real, int image) {  
 return new Complex(real, image);  
 }  
}

public class Complex {  
 int real;  
 int imagine;  
  
 public Complex(int real, int imagine) {  
 this.real = real;  
 this.imagine = imagine;  
 }  
  
 public int getReal() {  
 return real;  
 }  
  
 public int getImagine() {  
 return imagine;  
 }  
  
 @Override  
 public String toString() {  
 if (imagine < 0) {  
 return "Complex number {" + real + " (x) " +  
 + imagine + "i (iy)" +  
 "}";  
 }  
 else return "Complex number {" + real + " (x)" + " + "  
 + imagine + "i (iy)" +  
 "}";  
 }  
}

Вывод программы:



Упражнение 2

public class Client {  
 private static Chair *chair*;  
  
 public static void sit(){  
 System.*out*.println("Sitting on " + *chair*);  
 }  
  
 public static void setChair(Chair chaiir){  
 *chair* = chaiir;  
 }  
  
 public static void main(String[] args) {  
 AbstractChairFactory factory = new ChairFactory();  
 String chair\_type = "Magic"; // тип стула. Изменить для тестов на Functional или Victorian  
 if(chair\_type.equals("Magic"))  
 *setChair*(factory.createMagicChair());  
 else if(chair\_type.equals("Functional"))  
 *setChair*(factory.createFunctionalChair());  
 else if(chair\_type.equals("Victorian"))  
 *setChair*(factory.createVictorianChair());  
  
 *sit*();  
 }  
}

public class FunctionalChair implements Chair {  
 public int sum(int a, int b){  
 System.*out*.println(a + " + " + b + " = " + (a+b));  
 return a+b;  
 }  
  
 FunctionalChair(){  
 sum((int)(Math.*random*()\*10), (int)(Math.*random*()\*10));  
 }  
  
 @Override  
 public String toString() {  
 return "Functional Chair";  
 }  
}

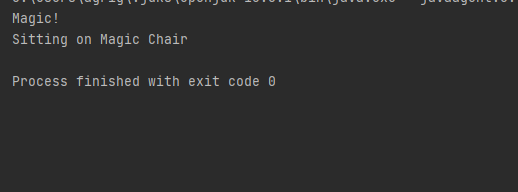
public class MagicChair implements Chair{  
 public void doMagic(){  
 System.*out*.println("Magic!");  
 }  
  
 public MagicChair(){  
 doMagic();  
 }  
  
 @Override  
 public String toString() {  
 return "Magic Chair";  
 }  
}

public class VictorianChair implements Chair{  
 private int age;  
  
 public int getAge() {  
 return age;  
 }  
  
 public VictorianChair(int age){  
 this.age = age;  
 }  
  
 @Override  
 public String toString() {  
 return "Victorian Chair " + age + " age";  
 }  
}

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

public interface Chair {  
 String toString();  
}

**Вывод программы:**

****

**Упражнение 3**

public class Main extends JFrame {  
  
 static JFrame *f* = new JFrame();  
 static JTextArea *area* = new JTextArea();  
  
 private static String readFile(String fileName) throws IOException {  
 return new String(Files.*readAllBytes*(Paths.*get*(fileName)));  
 }  
  
 public static void workWithTextFile(JFrame f, String path) throws IOException {  
 f.add(*area*, BorderLayout.*CENTER*);  
 String text = *readFile*(path);  
 *area*.setText(text);  
 if (text.isEmpty())  
 *area*.setText("Text... ");  
 }  
  
 public static void main(String[] args) {  
  
 AbstractFactory factory = new CreateTextDocument();  
 String file\_name = "New\_file";  
  
 *f*.setSize(800, 600);  
 *f*.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
 *f*.setLocationRelativeTo(null);  
 *f*.setLayout(new BorderLayout());  
  
 JMenuBar bar = new JMenuBar();  
 JMenu file\_menu = new JMenu("File");  
 JMenu window\_menu = new JMenu("Window");  
 bar.add(file\_menu);  
 bar.add(window\_menu);  
  
 // Создание файла  
 JMenu new\_menu = new JMenu("New");  
 JMenuItem text\_menu\_item = new JMenuItem("Text");  
 text\_menu\_item.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 try {  
 factory.createDoc("text", "New\_file");  
 *workWithTextFile*(*f*, System.*getProperty*("user.dir") + "\\" + file\_name + ".txt");  
 } catch (IOException ioException) {  
 // Возникает необрабатываемое исключение без блока try-catch  
 // Метод printStackTrace () класса Java.lang.Throwable, используемый для печати этого  
 // Throwable, наряду с другими деталями, такими как имя класса и номер строки, где  
 // произошло исключение, означает его обратную трассировку (Процесс пошагового выполнения программы).  
 ioException.printStackTrace();  
 }  
 }  
 });  
  
 // Здесь можно будет создать изображение  
 JMenuItem img\_menu\_item = new JMenuItem("Image (decorative)");  
  
 // Здесь можно будет создать аудио  
 JMenuItem music\_menu\_item = new JMenuItem("Music (decorative)");  
 file\_menu.add(new\_menu);  
 new\_menu.add(text\_menu\_item);  
 new\_menu.add(music\_menu\_item);  
 new\_menu.add(img\_menu\_item);  
  
 // Открытие файла  
  
 JMenuItem open\_menu\_item = new JMenuItem("Open recent");  
 open\_menu\_item.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 final String[] path\_file = {""};  
 //указание пути  
 JFrame f1 = new JFrame();  
 f1.setSize(250, 100);  
 f1.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
 f1.setLocationRelativeTo(null);  
 f1.setLayout(new GridLayout(2, 1));  
 JTextField field = new JTextField(System.*getProperty*("user.dir") + "\\" + file\_name + ".txt");  
 f1.add(field);  
 JButton butOK = new JButton("ОК");  
 f1.add(butOK);  
 butOK.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
  
 path\_file[0] = field.getText();  
 f1.setVisible(false);  
 if (path\_file[0].contains(".txt")) {  
 try {  
 *workWithTextFile*(*f*, path\_file[0]);  
 } catch (IOException ioException) {  
 ioException.printStackTrace();  
 }  
 }  
 }  
 });  
 f1.setVisible(true);  
  
 }  
 });  
  
 // Сохранение файла  
  
 JMenuItem save\_menu\_item = new JMenuItem("Save");  
 save\_menu\_item.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
  
 if (*area*.isVisible()) {  
 try (FileWriter writer = new FileWriter(file\_name + ".txt", false)) {  
 writer.write(*area*.getText());  
 writer.flush();  
 } catch (IOException ex) {  
 System.*out*.println(ex.getMessage());  
 }  
 }  
 }  
 });  
  
  
 // Выход  
 JMenuItem exit\_menu\_item = new JMenuItem("Close");  
  
 exit\_menu\_item.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 *f*.getContentPane().removeAll();  
 *f*.setJMenuBar(bar);  
 }  
 });  
 file\_menu.add(open\_menu\_item);  
 file\_menu.add(save\_menu\_item);  
 window\_menu.add(exit\_menu\_item);  
  
 *f*.setJMenuBar(bar);  
 *f*.setVisible(true);  
 }  
}

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

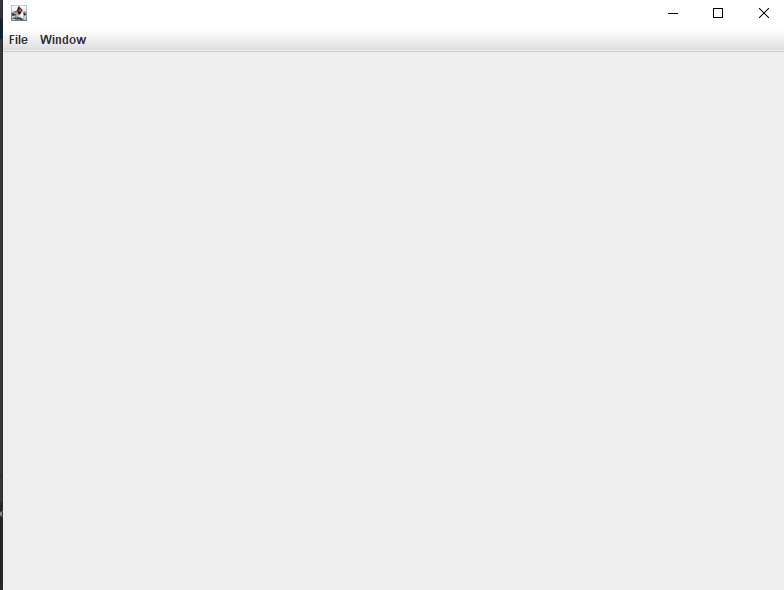
public class MusicDocument implements IDocument {  
 MusicDocument(){}  
 MusicDocument(String name){}  
}

public class TextDocument implements IDocument{  
  
 TextDocument() throws IOException {  
 Files.*createFile*(Path.*of*(System.*getProperty*("user.dir")+ "\\file.txt"));  
 }  
  
 TextDocument(String name) throws IOException {  
 name += ".txt";  
 Files.*createFile*(Path.*of*(System.*getProperty*("user.dir")+ "\\" + name));  
 }  
}

public interface IDocument {}

public class CreateTextDocument implements AbstractFactory {  
  
 @Override  
 public IDocument createDoc(String type, String name) throws IOException {  
 return switch (type) {  
 case "text" -> new TextDocument(name);  
 case "image" -> new ImageDocument(name);  
 case "music" -> new MusicDocument(name);  
 default -> null;  
 };  
 }  
}

public interface AbstractFactory {  
 IDocument createDoc(String type, String name) throws IOException;  
}

**Вывод программы: **

**Одиннадцатая практика**

**Задача:**

Задание 1. Очередь на массиве ● Найдите инвариант структуры данных «очередь». Определите функции, которые необходимы для реализации очереди. Найдите их пред- и постусловия.

● Реализуйте классы, представляющие циклическую очередь с применением массива.

o Класс ArrayQueueModule должен реализовывать один экземпляр очереди с использованием переменных класса.

o Класс ArrayQueueADT должен реализовывать очередь в виде абстрактного типа данных (с явной передачей ссылки на экземпляр очереди).

o Класс ArrayQueue должен реализовывать очередь в виде класса (с неявной передачей ссылки на экземпляр очереди).

o Должны быть реализованы следующие функции(процедуры) / методы:

▪ enqueue – добавить элемент в очередь;

▪ element – первый элемент в очереди;

▪ dequeue – удалить и вернуть первый элемент в очереди;

▪ size – текущий размер очереди;

▪ isEmpty – является ли очередь пустой;

▪ clear – удалить все элементы из очереди.

o Инвариант, пред- и постусловия записываются в исходном коде в виде комментариев.

o Обратите внимание на инкапсуляцию данных и кода во всех трех реализациях.

● Напишите тесты реализованным классам.

Задание 2. Очереди

4 Определите интерфейс очереди Queue и опишите его контракт.

5 Реализуйте класс LinkedQueue — очередь на связном списке.

6 Выделите общие части классов LinkedQueue и ArrayQueue в базовый класс AbstractQueue.

Дополнительные задания

Задание 3. Вычисление выражений

1. Разработайте классы Const, Variable, Add, Subtract, Multiply, Divide для вычисления выражений с одной переменной.

2. Классы должны позволять составлять выражения вида

new Subtract(

new Multiply(

new Const(2),

new Variable("x")

),

new Const(3)

).evaluate(5)

 При вычислении такого выражения вместо каждой переменной подставляется значение, переданное в качестве параметра методу evaluate (на данном этапе имена переменных игнорируются). Таким образом, результатом вычисления приведенного примера должно стать число 7.

3. Для тестирования программы должен быть создан класс Main, который вычисляет значение выражения x2−2x+1, для x, заданного в командной строке.

4. При выполнение задания следует обратить внимание на:

● Выделение общего интерфейса создаваемых классов.

● Выделение абстрактного базового класса для бинарных операций.

Задание 4

1. Доработайте предыдущее задание, так что бы выражение строилось по записи вида

x \* (y - 2)\*z + 1

 Для этого реализуйте класс ExpressionParser с методом TripleExpression parse(String).

2. В записи выражения могут встречаться: умножение \*, деление /, сложение +, вычитание -, унарный минус -, целочисленные константы (в десятичной системе счисления, которые помещаются в 32-битный знаковый целочисленный тип), круглые скобки, переменные (x, y и z) и произвольное число пробельных символов в любом месте (но не внутри констант).

3. Приоритет операторов, начиная с наивысшего

● унарный минус;

● умножение и деление;

● сложение и вычитание.

4. Для выражения 1000000\*x\*x\*x\*x\*x/(x-1) вывод программы должен иметь следующий вид:

x f

0 0

1 division by zero

2 32000000

3 121500000

4 341333333

5 overflow

6 overflow

7 overflow

8 overflow

9 overflow

10 overflow

5. Результат division by zero (overflow) означает, что в процессе вычисления произошло деление на ноль (переполнение). 6. Разбор выражений рекомендуется производить методом рекурсивного спуска. Алгоритм должен работать за линейное время.

7. При выполнении задания следует обратить внимание на дизайн и обработку исключений.

8. Человеко-читаемые сообщения об ошибках должны выводится на консоль.

9. Программа не должна «вылетать» с исключениями (как стандартными, так и добавленными).

**Задача первая:**

public class ArrayQueueTest {  
 public static void main(String[] args) {  
 ArrayQueue queue = new ArrayQueue();  
 *fill*(queue);  
 *dump*(queue);  
 *fill*(queue);  
 *clear*(queue);  
 *fill*(queue);  
 *getArray*(queue);  
 }  
  
 public static void fill(ArrayQueue queue) {  
 for (int i = 0; i < 10; i++) {  
 queue.enqueue(i);  
 }  
 }  
  
 public static void dump(ArrayQueue queue) {  
 while (!queue.isEmpty()) {  
 System.*out*.println(queue.size() + " " + queue.element() + " " + queue.dequeue());  
 }  
 }  
  
 public static void getArray(ArrayQueue queue) {  
 Object[] arr = queue.toArray();  
 for (Object o : arr) {  
 System.*out*.print(o + " ");  
 }  
 }  
  
 public static void clear(ArrayQueue queue) {  
 queue.clear();  
 System.*out*.println("Queue is empty? " + queue.isEmpty());  
 }  
}

public class ArrayQueueModuleTest {  
 public static void main(String[] args) {  
 *fill*();  
 *dump*();  
 *fill*();  
 *clear*();  
 *fill*();  
 *getArray*();  
 }  
  
 public static void fill() {  
 for (int i = 0; i < 10; i++) {  
 ArrayQueueModule.enqueue(i);  
 }  
 }  
  
 public static void getArray() {  
 Object[] arr = ArrayQueueModule.toArray();  
 for (Object o : arr) {  
 System.*out*.print(o + " ");  
 }  
 }  
  
 public static void dump() {  
 while (!(ArrayQueueModule.isEmpty())) {  
 System.*out*.println(ArrayQueueModule.size() + " " + ArrayQueueModule.element() + " " + ArrayQueueModule.dequeue());  
 }  
 }  
  
 public static void clear() {  
 ArrayQueueModule.clear();  
 System.*out*.println(ArrayQueueModule.isEmpty());  
 }  
}

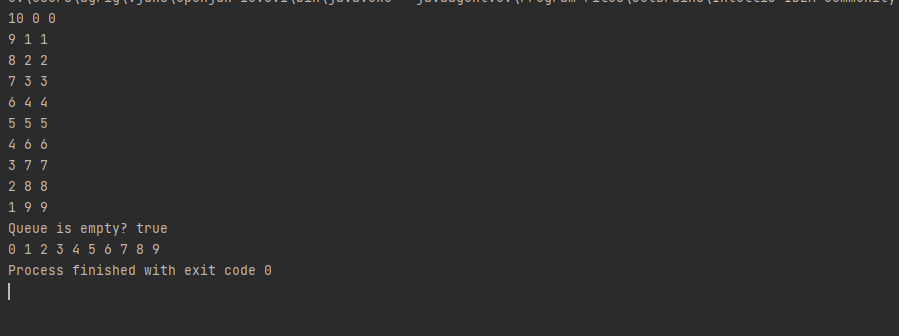
public class ArrayQueueModule {  
 private static final int *START\_CAPACITY* = 100;  
 private static Object[] *array*;  
 private static int *size*;  
 private static int *head*;  
 private static int *tail*;  
  
 public ArrayQueueModule() {  
 *array* = new Object[*START\_CAPACITY*];  
 *size* = *head* = *tail* = 0;  
 }  
  
 private static void ensureCapacity(int capacity) {  
 if (capacity >= *array*.length) {  
 Object[] temp = new Object[*array*.length << 1];  
 int count = *tail* < *head* ? *array*.length - *head* : *size*;  
 System.*arraycopy*(*array*, *head*, temp, 0, count);  
 if (*tail* < *head*)  
 System.*arraycopy*(*array*, 0, temp, count, *tail*);  
 *array* = temp;  
 *head* = 0;  
 *tail* = *size*;  
 }  
 }  
  
 public static void enqueue(Object element) {  
 assert element != null;  
 *ensureCapacity*(*size* + 1);  
 *array*[*tail*] = element;  
 *tail* = (*tail* + 1) % *array*.length;  
 *size*++;  
 }  
  
 public static Object element() {  
 assert *size* > 0;  
 return *array*[*head*];  
 }  
  
 public static Object dequeue() {  
 assert *size* > 0;  
 Object result = *element*();  
 *array*[*head*] = null;  
 *size*--;  
 *head* = (*head* + 1) % *array*.length;  
 return result;  
 }  
  
 public static void push(Object element) {  
 assert element != null;  
 *ensureCapacity*(*size* + 1);  
 *head* = (*head* == 0) ? *array*.length - 1 : *head* - 1;  
 *array*[*head*] = element;  
 *size*++;  
 }  
  
 public static Object peek() {  
 assert *size* > 0;  
 return *array*[(*tail* == 0) ? *array*.length - 1 : *tail* - 1];  
 }  
  
 public static Object remove() {  
 assert *size* > 0;  
 Object result = *peek*();  
 *tail* = *tail* == 0 ? *array*.length - 1 : --*tail*;  
 *array*[*tail*] = null;  
 *size*--;  
 return result;  
 }  
  
 public static int size() {  
 return *size*;  
 }  
  
 public static boolean isEmpty() {  
 return *size* == 0;  
 }  
  
 public static void clear() {  
 *size* = *head* = *tail* = 0;  
 *array* = new Object[*START\_CAPACITY*];  
 }  
  
 public static Object[] toArray() {  
 Object[] temp = new Object[*size*];  
 int count = *tail* < *head* ? *array*.length - *head* : *size*;  
 System.*arraycopy*(*array*, *head*, temp, 0, count);  
 if (*tail* < *head*)  
 System.*arraycopy*(*array*, 0, temp, count, *tail*);  
 return temp;  
 }  
}

public class ArrayQueueADTTest {  
 public static void main(String[] args) {  
 ArrayQueueADT queue = new ArrayQueueADT();  
 *fill*(queue);  
 *dump*(queue);  
 *fill*(queue);  
 *clear*(queue);  
 *fill*(queue);  
 *getArray*(queue);  
 }  
  
 public static void fill(ArrayQueueADT queue) {  
 for (int i = 0; i < 10; i++) {  
 ArrayQueueADT.enqueue(queue, i);  
 }  
 }  
  
 public static void getArray(ArrayQueueADT queue) {  
 Object[] arr = ArrayQueueADT.toArray(queue);  
 for (Object o : arr) {  
 System.*out*.print(o + " ");  
 }  
 }  
  
 public static void dump(ArrayQueueADT queue) {  
 while (!ArrayQueueADT.isEmpty(queue)) {  
 System.*out*.println(  
 ArrayQueueADT.size(queue) + " " + ArrayQueueADT.element(queue) + " " + ArrayQueueADT.dequeue(queue));  
 }  
 }  
  
 public static void clear(ArrayQueueADT queue) {  
 ArrayQueueADT.clear(queue);  
 System.*out*.println(ArrayQueueADT.isEmpty(queue));  
 }  
}

public class ArrayQueueADT {  
 // final - значение переменной не изменяется после инициализации  
 private static final int *START\_CAPACITY* = 10;  
 private Object[] array;  
 private int size;  
 private int head;  
 private int tail;  
  
 // Конструктор  
 public ArrayQueueADT() {  
 array = new Object[*START\_CAPACITY*];  
 size = head = tail = 0;  
 }  
  
 private static void ensureCapacity(ArrayQueueADT adt, int capacity) {  
 if (capacity >= adt.array.length) {  
 Object[] temp = new Object[adt.array.length << 1];  
  
 // Если adt.tail < adt.head верно, то выполняем adt.array.length - adt.head, иначе adt.size  
 int count = adt.tail < adt.head ? adt.array.length - adt.head : adt.size;  
 System.*arraycopy*(adt.array, adt.head, temp, 0, count);  
  
 if (adt.tail < adt.head)  
 System.*arraycopy*(adt.array, 0, temp, count, adt.tail);  
   
 adt.array = temp;  
 adt.head = 0;  
 adt.tail = adt.size;  
 }  
 }  
  
 // Добавить элемент в очередь  
 public static void enqueue(ArrayQueueADT adt, Object element) {  
 assert element != null;  
 *ensureCapacity*(adt, adt.size + 1);  
 adt.array[adt.tail] = element;  
 adt.tail = (adt.tail + 1) % adt.array.length;  
 adt.size++;  
 }  
  
 public static Object element(ArrayQueueADT adt) {  
 assert adt.size > 0;  
 return adt.array[adt.head];  
 }  
  
 public static Object dequeue(ArrayQueueADT adt) {  
 assert adt.size > 0;  
 Object result = *element*(adt);  
 adt.array[adt.head] = null;  
 adt.size--;  
 adt.head = (adt.head + 1) % adt.array.length;  
 return result;  
 }  
  
 public static void push(ArrayQueueADT adt, Object element) {  
 assert element != null;  
 *ensureCapacity*(adt, adt.size + 1);  
 adt.head = (adt.head == 0) ? adt.array.length - 1 : adt.head - 1;  
 adt.array[adt.head] = element;  
 adt.size++;  
 }  
  
 public static Object peek(ArrayQueueADT adt) {  
 assert adt.size > 0;  
 return adt.array[(adt.tail == 0) ? adt.array.length - 1 : adt.tail - 1];  
 }  
  
 public static Object remove(ArrayQueueADT adt) {  
 assert adt.size > 0;  
 Object result = *peek*(adt);  
 adt.tail = adt.tail == 0 ? adt.array.length - 1 : --adt.tail;  
 adt.array[adt.tail] = null;  
 adt.size--;  
 return result;  
 }  
  
 public static int size(ArrayQueueADT adt) {  
 return adt.size;  
 }  
  
 public static boolean isEmpty(ArrayQueueADT adt) {  
 return adt.size == 0;  
 }  
  
 public static void clear(ArrayQueueADT adt) {  
 adt.size = adt.head = adt.tail = 0;  
 adt.array = new Object[*START\_CAPACITY*];  
 }  
  
 public static Object[] toArray(ArrayQueueADT adt) {  
 Object[] temp = new Object[adt.size];  
 int count = adt.tail < adt.head ? adt.array.length - adt.head : adt.size;  
 System.*arraycopy*(adt.array, adt.head, temp, 0, count);  
 if (adt.tail < adt.head)  
 System.*arraycopy*(adt.array, 0, temp, count, adt.tail);  
 return temp;  
 }  
}

public class ArrayQueue {  
 private final int START\_CAPACITY = 10;  
 private Object[] array;  
 private int size;  
 private int head;  
 private int tail;  
  
 public ArrayQueue() {  
 array = new Object[START\_CAPACITY];  
 size = head = tail = 0;  
 }  
  
 // Pre: capacity' > 0  
 // Post: if required capacity of array bigger then current  
 // Post: capacity' = capacity \* 2  
 // Inv: if (capacity' = capacity \* 2) head < tail  
 private void ensureCapacity(int capacity) {  
 if (capacity >= array.length) {  
 Object[] temp = new Object[array.length << 1];  
 int count = tail < head ? array.length - head : size;  
 System.*arraycopy*(array, head, temp, 0, count);  
 if (tail < head)  
 System.*arraycopy*(array, 0, temp, count, tail);  
 array = temp;  
 head = 0;  
 tail = size;  
 }  
 }  
  
 // Pre: value != null  
 // Pre: value is immutable  
 // Post: queue[size - 1] = element  
 // Post: size' = size + 1  
 public void enqueue(Object element) {  
 assert element != null;  
 ensureCapacity(size + 1);  
 array[tail] = element;  
 tail = (tail + 1) % array.length;  
 size++;  
 }  
  
 // Pre: size > 0  
 // Post: Result = queue[0]  
 public Object element() {  
 assert size > 0;  
 return array[head];  
 }  
  
 // Pre: size > 0  
 // Post: Result = queue[0]  
 // Post: size' = size - 1  
 public Object dequeue() {  
 assert size > 0;  
 Object result = element();  
 array[head] = null;  
 size--;  
 head = (head + 1) % array.length;  
 return result;  
 }  
  
 // Pre: element != null  
 // Pre: element is immutable  
 // Post: queue[0] = element  
 // Post: size' = size + 1  
 public void push(Object element) {  
 assert element != null;  
 ensureCapacity(size + 1);  
 head = (head == 0) ? array.length - 1 : head - 1;  
 array[head] = element;  
 size++;  
 }  
  
 // Pre: size > 0  
 // Post: Result = queue[size - 1]  
 public Object peek() {  
 assert size > 0;  
 return array[(tail == 0) ? array.length - 1 : tail - 1];  
 }  
  
 // Pre: size > 0  
 // Post: Result = queue[size - 1]  
 // Post: size' = size - 1  
 public Object remove() {  
 assert size > 0;  
 Object result = peek();  
 tail = tail == 0 ? array.length - 1 : --tail;  
 array[tail] = null;  
 size--;  
 return result;  
 }  
  
 // Post: Result = size  
 public int size() {  
 return size;  
 }  
  
 // Post: Result = size > 0?  
 public boolean isEmpty() {  
 return size == 0;  
 }  
  
 // INV: size' = 0;  
 // Post: Clear the queue  
 public void clear() {  
 size = head = tail = 0;  
 array = new Object[START\_CAPACITY];  
 }  
  
 // Post: Result = queue[0...size - 1]  
 public Object[] toArray() {  
 Object[] temp = new Object[size];  
 int count = tail < head ? array.length - head : size;  
 System.*arraycopy*(array, head, temp, 0, count);  
 if (tail < head)  
 System.*arraycopy*(array, 0, temp, count, tail);  
 return temp;  
 }  
}

**Вывод программы:**

****

**Упражнение 2**

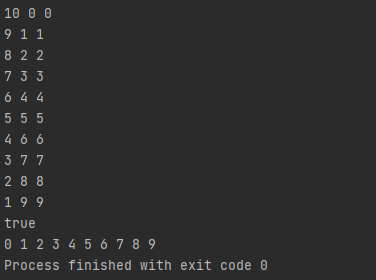
public abstract class AbstractQueue implements Queue {  
 protected int size = 0;  
  
 protected abstract void enqueueRealisation(Object element);  
  
 protected abstract Object elementRealisation();  
  
 protected abstract void dequeueRealisation();  
  
 protected abstract void pushRealisation(Object element);  
  
 protected abstract Object peekRealisation();  
  
 protected abstract void removeRealisation();  
  
 public void enqueue(Object element) {  
 assert element != null;  
 enqueueRealisation(element);  
 size++;  
 }  
  
 public Object element() {  
 assert size > 0;  
 return elementRealisation();  
 }  
  
 public Object dequeue() {  
 assert size > 0;  
 Object result = elementRealisation();  
 dequeueRealisation();  
 --size;  
 return result;  
 }  
  
 public void push(Object element) {  
 System.*out*.println("Push");  
 assert element != null;  
 pushRealisation(element);  
 size++;  
 }  
  
 public Object peek() {  
 assert size > 0;  
 return peekRealisation();  
 }  
  
 public Object remove() {  
 assert size > 0;  
 Object result = peek();  
 removeRealisation();  
 --size;  
 return result;  
 }  
  
 public int size() {  
 return size;  
 }  
  
 public boolean isEmpty() {  
 return size == 0;  
 }  
  
 public void clear() {  
 while (!isEmpty()) {  
 dequeue();  
 }  
 }  
  
 public Object[] toArray() {  
 Object[] newArray = new Object[size];  
 for (int i = 0; i < size; i++) {  
 newArray[i] = dequeue();  
 enqueue(newArray[i]);  
 }  
 return newArray;  
 }  
}

public class ArrayQueue extends AbstractQueue {  
 private final int START\_CAPACITY = 100;  
 private Object[] array;  
 private int head;  
 private int tail;  
  
 public ArrayQueue() {  
 array = new Object[START\_CAPACITY];  
 head = tail = 0;  
 }  
  
 private void ensureCapacity(int capacity) {  
 if (capacity >= array.length) {  
 Object[] temp = new Object[array.length << 1];  
 int count = tail < head ? array.length - head : size;  
 System.*arraycopy*(array, head, temp, 0, count);  
 if (tail < head)  
 System.*arraycopy*(array, 0, temp, count, tail);  
 array = temp;  
 head = 0;  
 tail = size;  
 }  
  
 }  
  
 protected void enqueueRealisation(Object element) {  
 ensureCapacity(size + 1);  
 array[tail] = element;  
 tail = (tail + 1) % array.length;  
 }  
  
 public Object elementRealisation() {  
 return array[head];  
 }  
  
 protected void dequeueRealisation() {  
 array[head] = null;  
 head = (head + 1) % array.length;  
 }  
  
 protected void pushRealisation(Object element) {  
 ensureCapacity(size + 1);  
 head = (this.head - 1 >= 0) ? array.length - 1 : head - 1;  
 array[head] = element;  
 }  
  
 protected Object peekRealisation() {  
 return array[(tail == 0) ? array.length - 1 : tail - 1];  
 }  
  
 protected void removeRealisation() {  
 tail = tail == 0 ? array.length - 1 : --tail;  
 array[tail] = null;  
 }  
}

public class LinkedQueue extends AbstractQueue {  
 private Node tail;  
 private Node head;  
  
 protected void enqueueRealisation(Object element) {  
 tail = new Node(element, null, tail);  
 if (head == null)  
 head = tail;  
 else  
 tail.next.prev = tail;  
  
  
 }  
  
 protected Object elementRealisation() {  
 return head.value;  
 }  
  
 protected void dequeueRealisation() {  
 if (head.prev != null) head.prev.next = null;  
 head = head.prev;  
  
 }  
  
 protected void pushRealisation(Object element) {  
 head = new Node(element, head, null);  
 if (tail == null) {  
 tail = head;  
 } else  
 head.prev.next = head;  
  
 }  
  
 protected Object peekRealisation() {  
 return tail.value;  
 }  
  
 protected void removeRealisation() {  
 tail = tail.next;  
 if (tail != null) {  
 tail.prev = null;  
 }  
 }  
  
 private static class Node {  
 Object value;  
 Node prev;  
 Node next;  
  
 Node(Object element, Node prev, Node next) {  
 value = element;  
 this.prev = prev;  
 this.next = next;  
 }  
 }  
}

public interface Queue {  
 void enqueue(Object element);  
  
 Object element();  
  
 Object dequeue();  
  
 Object remove();  
  
 Object peek();  
  
 void push(Object element);  
  
 int size();  
  
 boolean isEmpty();  
  
 void clear();  
  
 Object[] toArray();  
}

**Вывод программы:**

****

**Упражнение 3**

public class Main {  
 public static void main(String[] args) {  
 Scanner scanner = new Scanner(System.*in*);  
 int x = scanner.nextInt();  
  
 System.*out*.println(  
 new Add(  
 new Subtract(  
 new Multiply(  
 new Variable("x"),  
 new Variable("x")  
 ),  
 new Multiply(  
 new Const(2),  
 new Variable("x")  
 )  
 ),  
 new Const(1)  
 ).evaluate(x, x, x));  
 }  
}

public interface DoubleExpression {  
 double evaluate(double x);  
}

public interface Expression extends TripleExpression {  
}

public interface TripleExpression {  
 int evaluate(int x, int y, int z);  
}

public class Const implements Expression {  
 private final int iVal;  
 private final double dVal;  
  
 public Const(int value) {  
 iVal = value;  
 dVal = value;  
 }  
  
 public Const(double value) {  
 iVal = 0;  
 dVal = value;  
 }  
  
 public int evaluate(int x) {  
 return iVal;  
 }  
  
 public double evaluate(double x) {  
 return dVal;  
 }  
  
 public int evaluate(int x, int y, int z) {  
 return iVal;  
 }  
}

public class Variable implements Expression {  
 private final String name;  
  
 public Variable(String name) {  
 this.name = name;  
 }  
  
 public int evaluate(int x, int y, int z) {  
 return name.equals("x") ? x : name.equals("y") ? y : name.equals("z") ? z : 0;  
 }  
}

public class Add extends Operation {  
 public Add(TripleExpression left, TripleExpression right) {  
 super(left, right);  
 }  
  
 protected int evaluate(int left, int right) {  
 return left + right;  
 }  
  
 protected double evaluate(double left, double right) {  
 return left + right;  
 }  
}

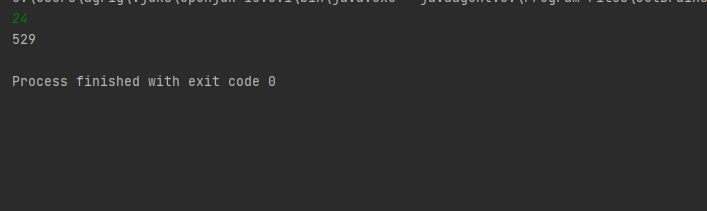
public class Divide extends Operation {  
 public Divide(TripleExpression left, TripleExpression right) {  
 super(left, right);  
 }  
  
 protected int evaluate(int left, int right) {  
 return left / right;  
 }  
  
 protected double evaluate(double left, double right) {  
 return left / right;  
 }  
}

public class Multiply extends Operation {  
 public Multiply(TripleExpression left, TripleExpression right) {  
 super(left, right);  
 }  
  
 protected int evaluate(int left, int right) {  
 return left \* right;  
 }  
  
 protected double evaluate(double left, double right) {  
 return left \* right;  
 }  
}

public abstract class Operation implements Expression {  
 private final TripleExpression left;  
 private final TripleExpression right;  
  
 public Operation(TripleExpression left, TripleExpression right) {  
 this.left = left;  
 this.right = right;  
 }  
  
 protected abstract int evaluate(int left, int right);  
  
 public int evaluate(int x, int y, int z) {  
 return evaluate(left.evaluate(x, y, z), right.evaluate(x, y, z));  
 }  
}

public class Subtract extends Operation {  
 public Subtract(TripleExpression left, TripleExpression right) {  
 super(left, right);  
 }  
  
 protected int evaluate(int left, int right) {  
 return left - right;  
 }  
  
 protected double evaluate(double left, double right) {  
 return left - right;  
 }  
}

**Вывод программы:**

****

**Упражнение 4**

public class Main {  
 public static void main(String[] args) throws FormatParserException {  
 ExpressionParser parse = new ExpressionParser();  
  
 Scanner sc = new Scanner(System.*in*);  
 int x = sc.nextInt();  
 int y = sc.nextInt();  
 int z = sc.nextInt();  
  
 System.*out*.println(parse.parse("x \* (y - 2)\*z + 1").evaluate(x, y, z));  
 System.*out*.println(parse.parse("1000000\*x\*x\*x\*x\*x/(x-1)").evaluate(x, x, x));  
 }  
}

public class ExpressionParser implements Parser {  
 private String rest;  
 private int firstIndex;  
 private boolean intMin = false;  
 private int len;  
  
 public TripleExpression parse(String s) throws FormatParserException {  
 firstIndex = 0;  
 rest = s;  
 len = rest.length();  
 TripleExpression result = addSubtract();  
 if (firstIndex != len) {  
 String message;  
 if (rest.charAt(firstIndex) == ')') {  
 message = "Can't find open parenthesis at index " + firstIndex +  
 "\n\t" + getOpenBracketException();  
 } else {  
 message = getUnknownLexemeExceptionMessage();  
 }  
 throw new FormatParserException(message);  
 }  
 return result;  
 }  
  
 private void skipSpace() {  
 while (firstIndex < len && Character.*isWhitespace*(rest.charAt(firstIndex))) {  
 firstIndex++;  
 }  
 }  
  
 private TripleExpression addSubtract() throws FormatParserException {  
 TripleExpression current;  
 TripleExpression acc = mulDiv();  
 while (firstIndex < len && rest.charAt(firstIndex) != ')') {  
 if (!(rest.charAt(firstIndex) == '+' || rest.charAt(firstIndex) == '-')) break;  
 char sign = rest.charAt(firstIndex);  
 firstIndex++;  
 current = mulDiv();  
 if (sign == '+') {  
 acc = new CheckedAdd(acc, current);  
 } else {  
 acc = new CheckedSubtract(acc, current);  
 }  
 }  
 return acc;  
 }  
  
 private TripleExpression mulDiv() throws FormatParserException {  
 TripleExpression current = operand();  
 while (true) {  
 skipSpace();  
 if (firstIndex >= len) {  
 return current;  
 }  
 char sign = rest.charAt(firstIndex);  
 if ((sign != '\*' && sign != '/')) return current;  
 firstIndex++;  
 TripleExpression right = operand();  
  
 if (sign == '\*') {  
 current = new CheckedMultiply(current, right);  
 } else {  
 current = new CheckedDivide(current, right);  
 }  
 }  
 }  
  
 private TripleExpression operand() throws FormatParserException {  
 skipSpace();  
 if (len == firstIndex) {  
 String prevExpression = getPrevArg(false);  
 throw new FormatParserException("Expected argument, found end of task4.expression at index " + (firstIndex - 1) +  
 "\n\t"  
 + prevExpression + "<missing argument>");  
 }  
 char c = rest.charAt(firstIndex);  
 switch (c) {  
 case 's':  
 if (checkSqrt()) {  
 return sqrtAbs(true);  
 } else {  
 throw new FormatParserException("expected sqrt, found unknown lexeme at index " + firstIndex  
 + "\n\t" + getWord());  
 }  
 case 'a':  
 if (checkAbs()) {  
 return sqrtAbs(false);  
 } else {  
 throw new FormatParserException("expected abs, found " + c + "\* at index " + firstIndex  
 + "\n\t" + getWord());  
 }  
 case ')':  
 throw new FormatParserException("Syntax parser exception : unexpected close bracket at index" + firstIndex  
 + "\n\t" + getCloseOpenBracketException());  
 case '(':  
 return bracket();  
 case 'x':  
 case 'y':  
 case 'z':  
 firstIndex++;  
 return new Variable(String.*valueOf*(c));  
 case '-':  
 firstIndex++;  
 TripleExpression current = operand();  
 if (intMin) {  
 intMin = false;  
 return current;  
 } else  
 return new CheckedNegate(current);  
 default:  
 skipSpace();  
 StringBuilder strNumber = new StringBuilder();  
 while (firstIndex < rest.length() && (Character.*isDigit*(rest.charAt(firstIndex)))) {  
 strNumber.append(rest.charAt(firstIndex++));  
 }  
 if (strNumber.toString().equals("")) {  
 String message;  
 char op = rest.charAt(firstIndex);  
 if (op != '+' && op != '-' && op != '\*' && op != '/') {  
 message = getUnknownLexemeExceptionMessage();  
 throw new FormatParserException(message);  
 }  
 message = getArgumentException();  
 throw new FormatParserException(message);  
 }  
 if (strNumber.toString().equals("2147483648")) {  
 intMin = true;  
 return new Const(-2147483648);  
 }  
 int val;  
 try {  
 val = Integer.*parseInt*(strNumber.toString());  
 } catch (NumberFormatException e) {  
 throw new FormatParserException("Illegal number at index " + firstIndex +  
 "\n\t" + strNumber.toString());  
 }  
 return new Const(val);  
 }  
 }  
  
 private TripleExpression bracket() throws FormatParserException {  
 firstIndex++;  
 TripleExpression acc = addSubtract();  
 if (rest.length() == firstIndex) {  
 throw new FormatParserException("Can't find close parenthesis at index " + firstIndex  
 + "\n\t" + getCloseOpenBracketException());  
 }  
 if (rest.charAt(firstIndex) == ')') {  
 firstIndex++;  
 return acc;  
 } else {  
 throw new FormatParserException("Space in constant at index " + firstIndex +  
 "\n" + getPrevArg(false) + rest.charAt(firstIndex));  
 }  
 }  
  
 private TripleExpression sqrtAbs(boolean type) throws FormatParserException {  
 TripleExpression acc;  
 firstIndex += type ? 4 : 3;  
 acc = operand();  
 if (type) {  
 return new CheckedSqrt(acc);  
 } else {  
 return new CheckedABS(acc);  
 }  
 }  
  
 private boolean checkSqrt() {  
 skipSpace();  
 return firstIndex + 4 < rest.length() && getFunc(4).equals("sqrt");  
 }  
  
 private boolean checkAbs() {  
 skipSpace();  
 return firstIndex + 3 < rest.length() && getFunc(3).equals("abs");  
 }  
  
 private String getFunc(int c) {  
 StringBuilder sb = new StringBuilder();  
 for (int i = 0; i < c; i++) {  
 sb.append(rest.charAt(firstIndex + i));  
 }  
 return sb.toString();  
 }  
  
 private String getArgumentException() {  
 String message;  
 String nextArg = getNextArg();  
 char curOp = rest.charAt(firstIndex);  
 String prevExpression = getPrevArg(true);  
 if (prevExpression.equals("")) {  
 message = "Can't find first argument at index " + firstIndex +  
 "\n\t <missing argument>" + curOp + nextArg;  
 } else {  
 message = "Can't find middle argument at index " + firstIndex +  
 "\n\t" + prevExpression + "<missing argument>" + curOp + nextArg;  
 }  
 return message;  
 }  
  
 private String getUnknownLexemeExceptionMessage() {  
 String message;  
 String prevExpression = getPrevArg(true);  
 String nextExpression = getNextArg();  
 char op = rest.charAt(firstIndex);  
 if (prevExpression.equals("")) {  
 message = "Unexpected start character " + op + " at index " + firstIndex  
 + "\n\t" + op + nextExpression;  
 } else if (nextExpression.equals("")) {  
 message = "Unexpected end character " + op + " at index " + firstIndex  
 + "\n\t" + prevExpression + op;  
 } else {  
 message = "Unexpected middle character " + rest.charAt(firstIndex) + " at index " + firstIndex  
 + "\n\t" + getPrevArg(true) + rest.charAt(firstIndex) + getNextArg();  
 }  
 return message;  
 }  
  
 private String getNextArg() {  
 int start = firstIndex + 1;  
 if (start == len) return "";  
 char c = rest.charAt(start);  
 int balance = 0;  
 StringBuilder sb = new StringBuilder();  
 while (start < len && !Character.*isDigit*(c) && c != 'x' && c != 'y' && c != 'z') {  
 if (c == '(') {  
 balance++;  
 } else if (c == ')') {  
 balance--;  
 }  
 sb.append(c);  
 if (++start == len) return sb.toString();  
 c = rest.charAt(start);  
 }  
 if (balance > 0) {  
 while (start != len && balance != 0) {  
 c = rest.charAt(start++);  
 if (c == '(') {  
 balance++;  
 } else if (c == ')') {  
 balance--;  
 }  
 sb.append(c);  
 }  
 } else {  
 if (c == 'x' || c == 'y' || c == 'z') {  
 sb.append(c);  
 } else {  
 while (start != len && Character.*isDigit*(c)) {  
 sb.append(c);  
 if (++start == len) break;  
 c = rest.charAt(start);  
 }  
 }  
 }  
 return sb.toString();  
 }  
  
 private String getPrevArg(boolean withOp) {  
 int start = Math.*max*(0, firstIndex - 1);  
 if (start == 0)  
 return "";  
  
 int end = firstIndex;  
 while (start > 0 && Character.*isWhitespace*(rest.charAt(start))) {  
 start--;  
 }  
 if (start == 0)  
 return String.*valueOf*(rest.charAt(0));  
  
 start--;  
 char c = rest.charAt(start);  
 while (start > 0 && c != ')' && c != '(' && c != '\*' && c != '/' && c != '+' && c != '-') {  
 c = rest.charAt(--start);  
 }  
  
 int balance = 0;  
 if (c == ')') {  
 balance++;  
 start--;  
 while (start > 0 && balance != 0) {  
 c = rest.charAt(start--);  
 if (c == ')') {  
 balance++;  
 } else if (c == '(') {  
 balance--;  
 }  
 }  
 if (start == 0)  
 return substring(firstIndex, end);  
  
 c = rest.charAt(start);  
 while (start > 0 && c != '\*' && c != '/' && c != '+' && c != '-') {  
 c = rest.charAt(--start);  
 }  
 }  
 return substring(withOp ? start : start + 1, end);  
  
 }  
  
 private String getCloseOpenBracketException() {  
 int start = firstIndex - 1;  
 while (start > 0 && rest.charAt(start) != ')') {  
 start--;  
 }  
 return substring(start, firstIndex);  
 }  
  
 private String getOpenBracketException() {  
 int start = firstIndex - 1;  
 while (start > 0 && rest.charAt(start) != '(') {  
 start--;  
 }  
 return substring(start, firstIndex + 1);  
 }  
  
 private String getWord() {  
 int start = firstIndex;  
 StringBuilder sb = new StringBuilder();  
 while (start < len && Character.*isAlphabetic*(rest.charAt(start))) {  
 sb.append(rest.charAt(start++));  
 }  
 return sb.toString();  
 }  
  
 private String substring(int start, int end) {  
 StringBuilder sb = new StringBuilder();  
 for (int i = start; i < end; i++) {  
 sb.append(rest.charAt(i));  
 }  
 return sb.toString();  
 }  
}

public interface Parser {  
 TripleExpression parse(String expression) throws FormatParserException;  
}

public abstract class UnaryOperation implements TripleExpression {  
 private final TripleExpression object;  
  
 public UnaryOperation(TripleExpression object) {  
 this.object = object;  
 }  
  
 abstract protected int evaluate(int object);  
  
 public int evaluate(int x, int y, int z) {  
 return evaluate(object.evaluate(x, y, z));  
 }  
}

public abstract class Operation implements TripleExpression {  
 private final TripleExpression left;  
 private final TripleExpression right;  
  
 public Operation(TripleExpression left, TripleExpression right) {  
 this.left = left;  
 this.right = right;  
 }  
  
 protected abstract int evaluate(int left, int right);  
  
  
 public int evaluate(int x, int y, int z) {  
 return evaluate(left.evaluate(x, y, z), right.evaluate(x, y, z));  
 }  
  
}

public abstract class BinaryOperation implements TripleExpression {  
 private final TripleExpression left;  
 private final TripleExpression right;  
  
 public BinaryOperation(TripleExpression left, TripleExpression right) {  
 this.left = left;  
 this.right = right;  
 }  
  
 protected abstract int evaluate(int left, int right);  
  
  
 public int evaluate(int x, int y, int z) {  
 return evaluate(left.evaluate(x, y, z), right.evaluate(x, y, z));  
 }  
}

public class Subtract extends Operation {  
 public Subtract(TripleExpression left, TripleExpression right) {  
 super(left, right);  
 }  
  
 protected int evaluate(int left, int right) {  
 return left - right;  
 }  
  
 protected double evaluate(double left, double right) {  
 return left - right;  
 }  
}

public class Multiply extends Operation {  
 public Multiply(TripleExpression left, TripleExpression right) {  
 super(left, right);  
 }  
  
 protected int evaluate(int left, int right) {  
 return left \* right;  
 }  
  
 protected double evaluate(double left, double right) {  
 return left \* right;  
 }  
}

public class Divide extends Operation {  
 public Divide(TripleExpression left, TripleExpression right) {  
 super(left, right);  
 }  
  
 protected int evaluate(int left, int right) {  
 return left / right;  
 }  
  
 protected double evaluate(double left, double right) {  
 return left / right;  
 }  
}

public class Add extends Operation {  
 public Add(TripleExpression left, TripleExpression right) {  
 super(left, right);  
 }  
  
 protected int evaluate(int left, int right) {  
 return left + right;  
 }  
  
 protected double evaluate(double left, double right) {  
 return left + right;  
 }  
}

public class CheckedSubtract extends BinaryOperation {  
 public CheckedSubtract(TripleExpression left, TripleExpression right) {  
 super(left, right);  
 }  
  
 protected int evaluate(int left, int right) {  
 if (right > 0 ? left < Integer.*MIN\_VALUE* + right  
 : left > Integer.*MAX\_VALUE* + right) {  
 throw new ArithmeticParserException("Integer overflow");  
 }  
 return left - right;  
 }  
}

public class CheckedSqrt extends UnaryOperation {  
 public CheckedSqrt(TripleExpression object) {  
 super(object);  
 }  
  
 protected int evaluate(int x) {  
 if (x < 0)  
 throw new ArithmeticParserException("negative radicand : sqrt(" + x + ")");  
 if (x == 0)  
 return 0;  
 int result = 1;  
 boolean decreased = false;  
 int nx;  
 for (; ; ) {  
 nx = (result + x / result) >> 1;  
 if (result == nx || nx > result && decreased) {  
 break;  
 }  
 decreased = nx < result;  
 result = nx;  
 }  
 return result;  
 }  
}

public class CheckedNegate extends UnaryOperation {  
 public CheckedNegate(TripleExpression object) {  
 super(object);  
 }  
  
 protected int evaluate(int object) {  
 if (object == Integer.*MIN\_VALUE*)  
 throw new ArithmeticParserException("Overflow exception : -(" + object + ")");  
 return -object;  
 }  
}

public class CheckedMultiply extends BinaryOperation {  
 public CheckedMultiply(TripleExpression left, TripleExpression right) {  
 super(left, right);  
 }  
  
 protected int evaluate(int left, int right) {  
 if (right > 0 ? left > Integer.*MAX\_VALUE* / right  
 || left < Integer.*MIN\_VALUE* / right  
 : (right < -1 ? left > Integer.*MIN\_VALUE* / right  
 || left < Integer.*MAX\_VALUE* / right  
 : right == -1  
 && left == Integer.*MIN\_VALUE*)) {  
 throw new ArithmeticParserException("Overflow exception : " + left + "\*" + right);  
 }  
 return left \* right;  
 }  
}

public class CheckedDivide extends BinaryOperation {  
 public CheckedDivide(TripleExpression left, TripleExpression right) {  
 super(left, right);  
 }  
  
 protected int evaluate(int left, int right) {  
 if ((left == Integer.*MIN\_VALUE*) && (right == -1)) {  
 throw new ArithmeticParserException("Overflow : " + left + "/" + right);  
 }  
 if (right == 0)  
 throw new ArithmeticParserException(left + "/" + right);  
 return left / right;  
 }  
}

public class CheckedAdd extends BinaryOperation {  
 public CheckedAdd(TripleExpression left, TripleExpression right) {  
 super(left, right);  
 }  
  
 protected int evaluate(int left, int right) {  
 if (right > 0 ? left > Integer.*MAX\_VALUE* - right  
 : left < Integer.*MIN\_VALUE* - right) {  
 throw new ArithmeticParserException("Overflow exception : " + left + " + " + right);  
 }  
 return left + right;  
 }  
}

public class CheckedABS extends UnaryOperation {  
 public CheckedABS(TripleExpression object) {  
 super(object);  
 }  
  
 protected int evaluate(int object) {  
 if (object == Integer.*MIN\_VALUE*) {  
 throw new ArithmeticParserException("Overflow exception : abs(" + object + ")");  
 }  
 return object < 0 ? -object : object;  
 }  
}

public class Variable implements TripleExpression {  
 private final String name;  
  
 public Variable(String name) {  
 this.name = name;  
 }  
  
 public int evaluate(int x, int y, int z) {  
 return name.equals("x") ? x : name.equals("y") ? y : name.equals("z") ? z : 0;  
  
 }  
}

public class Const implements TripleExpression {  
 private final int iVal;  
 private final double dVal;  
  
 public Const(int value) {  
 iVal = value;  
 dVal = value;  
 }  
  
 public Const(double value) {  
 iVal = 0;  
 dVal = value;  
 }  
  
 public int evaluate(int x) {  
 return iVal;  
 }  
  
 public double evaluate(double x) {  
 return dVal;  
 }  
  
 public int evaluate(int x, int y, int z) {  
 return iVal;  
 }  
}

public interface TripleExpression {  
 int evaluate(int x, int y, int z);  
}

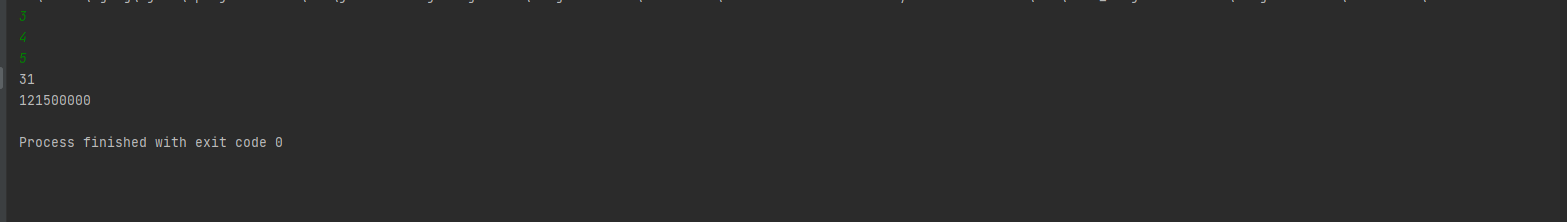
public interface Expression extends TripleExpression {  
}

public interface DoubleExpression {  
 double evaluate(double x);  
}

public class FormatParserException extends Exception {  
 public FormatParserException(String s) {  
 super(s);  
 }  
}

public class ArithmeticParserException extends ArithmeticException {  
 public ArithmeticParserException(String message) {  
 super(message);  
 }  
}

Вывод программы:



**Двенадцатая практика**

Задание 1. (10%)

Разработать класс Person, в котором имеется функция, возвращающая Фамилию И.О. Функция должна учитывать возможность отсутствия значений в полях Имя и Отчество. Программу оптимизировать с точки зрения быстродействия.

Задание 2. (20%)

Доработать класс адреса, который из полученной строки формата "Страна[d] Регион[d] Город[d] Улица[d] Дом[d] Корпус[d] Квартира" ([d] – разделитель, например, «запятая») выбирает соответствующие части и записывает их в соответствующие поля класса Address. Учесть, что в начале и конце разобранной части адреса не должно быть пробелов. Все поля адреса строковые. Разработать проверочный класс не менее чем на четыре адресных строки. В программе предусмотреть две реализации этого метода:

а) разделитель – только запятая (использовать метод split()); Внимание, при разработке нужно учесть, что

б) разделитель – любой из символов ,.;- (класс StringTokenizer).

Задание 3. (30%)

Реализуйте класс Shirt: Метод toString() выводит объяснение и значение полей построчно.

Дан также строковый массив: shirts[0] = "S001,Black Polo Shirt,Black,XL"; shirts[1] = "S002,Black Polo Shirt,Black,L"; shirts[2] = "S003,Blue Polo Shirt,Blue,XL"; shirts[3] = "S004,Blue Polo Shirt,Blue,M"; shirts[4] = "S005,Tan Polo Shirt,Tan,XL"; shirts[5] = "S006,Black T-Shirt,Black,XL"; shirts[6] = "S007,White T-Shirt,White,XL"; shirts[7] = "S008,White T-Shirt,White,L"; shirts[8] = "S009,Green T-Shirt,Green,S"; shirts[9] = "S010,Orange T-Shirt,Orange,S"; shirts[10] = "S011,Maroon Polo Shirt,Maroon,S";

Преобразуйте строковый массив в массив класса Shirt и выведите его на консоль.

Задание 4. (30%)

Разработайте класс, который получает строковое представление телефонного номера в одном из двух возможных строковых форматов:

+<Код страны><Номер 10 цифр>, например “+79175655655” или

“+104289652211” или

8<Номер 10 цифр> для России, например “89175655655”

и преобразует полученную строку в формат:

+<Код страны><Три цифры>–<Три цифры>–<Четыре цифры>

Задание 4. (30%)

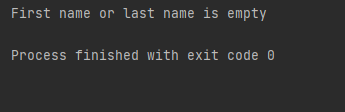
В методе main считай с консоли имя файла, который содержит слова, разделенные пробелом. В методе getLine() используя StringBuilder расставьте все слова в таком порядке, чтобы последняя буква данного слова совпадала с первой буквой следующего не учитывая регистр. Каждое слово должно участвовать 1 раз.

**Задача 1**

public class Person {  
 private String firstName = "";  
 private String lastName = "";  
  
 public Person() {}  
  
 public Person(String firstName) {  
 this.firstName = firstName;  
 }  
 public Person(String firstName, String lastName) {  
 this.firstName = firstName;  
 this.lastName = lastName;  
 }  
  
 public String getFirstName() {  
 return firstName;  
 }  
  
 public String getLastName() {  
 return lastName;  
 }  
  
 public String getName() {  
 if (firstName.isEmpty() || lastName.isEmpty())  
 return "First name or last name is empty";  
  
 return firstName + " " + lastName;  
 }  
}

public class TestPerson {  
 public static void main(String[] args) {  
 Person person = new Person("1");  
 System.*out*.println(person.getName());  
 }  
}

**Вывод программы:**

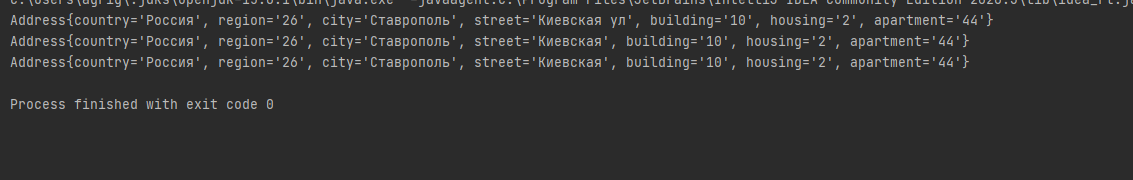
****

**Задача 2**

public class Address {  
 private String country;  
 private String reg;  
 private String city;  
 private String street;  
 private String building;  
 private String housing;  
 private String apartment;  
  
 private static final int *countField* = 7;  
  
  
 private Address(String country, String region, String city, String street, String building, String housing, String apartment) {  
 this.country = country;  
 this.reg = region;  
 this.city = city;  
 this.street = street;  
 this.building = building;  
 this.housing = housing;  
 this.apartment = apartment;  
 }  
  
  
 public static Address commaSeparator(String src) throws IllegalArgumentException {  
 src = src.trim();  
 String[] fields = src.split(", ");  
  
 if (fields.length != *countField*)  
 throw new IllegalArgumentException();  
  
 return new Address(fields[0], fields[1], fields[2], fields[3], fields[4], fields[5], fields[6]);  
 }  
  
  
 public static Address anySeparator(String src) throws IllegalArgumentException {  
 src = src.trim();  
 StringTokenizer stringTokenizer = new StringTokenizer(src, " ,.;-");  
 String[] fields = new String[7];  
  
 if (stringTokenizer.countTokens() < *countField*)  
 throw new IllegalArgumentException();  
  
 for (int i = 0; i < *countField*; ++i)  
 fields[i] = stringTokenizer.nextToken();  
  
 return new Address(fields[0], fields[1], fields[2], fields[3], fields[4], fields[5], fields[6]);  
 }  
  
 public String toString() {  
 return "Address{" +  
 "country='" + country + '\'' +  
 ", region='" + reg + '\'' +  
 ", city='" + city + '\'' +  
 ", street='" + street + '\'' +  
 ", building='" + building + '\'' +  
 ", housing='" + housing + '\'' +  
 ", apartment='" + apartment + '\'' +  
 '}';  
 }  
}

public class TestAddress {  
  
 public static void main(String[] args) {  
 Address address1 = Address.commaSeparator("Россия, 26, Ставрополь, Киевская ул, 10, 2, 44");  
 Address address2 = Address.anySeparator("Россия; 26; Ставрополь; Киевская; 10; 2; 44");  
 Address address3 = Address.anySeparator("Россия 26 Ставрополь Киевская 10 2 44");  
  
 System.*out*.println(address1);  
 System.*out*.println(address2);  
 System.*out*.println(address3);  
 }  
}

**Вывод программы:**

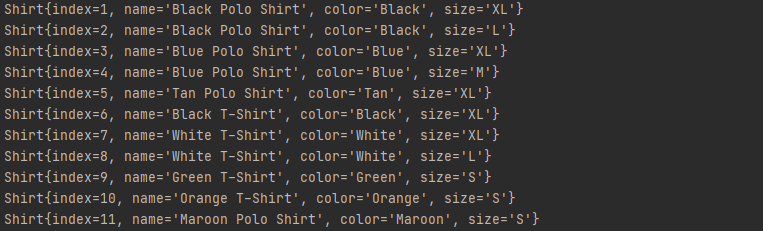
****

**Задача 3**

public class Shirt {  
  
 private int index;  
 private String name;  
 private String color;  
 private String size;  
  
  
 public Shirt(int index, String name, String color, String size) {  
 this.index = index;  
 this.name = name;  
 this.color = color;  
 this.size = size;  
 }  
  
  
 public static Shirt fromString(String src) throws IllegalArgumentException {  
 String[] fields = src.split(",");  
  
 if (fields.length != 4)  
 throw new IllegalArgumentException();  
  
 fields[0] = fields[0].replace('S', '0');  
  
 return new Shirt(  
 Integer.*parseInt*(fields[0]),  
 fields[1],  
 fields[2],  
 fields[3]  
 );  
 }  
  
 public String toString() {  
 return "Shirt{" +  
 "index=" + index +  
 ", name='" + name + '\'' +  
 ", color='" + color + '\'' +  
 ", size='" + size + '\'' +  
 '}';  
 }  
}

public class TestShirt {  
  
 private static final String[] *shirts* = new String[]{  
 "S001,Black Polo Shirt,Black,XL",  
 "S002,Black Polo Shirt,Black,L",  
 "S003,Blue Polo Shirt,Blue,XL",  
 "S004,Blue Polo Shirt,Blue,M",  
 "S005,Tan Polo Shirt,Tan,XL",  
 "S006,Black T-Shirt,Black,XL",  
 "S007,White T-Shirt,White,XL",  
 "S008,White T-Shirt,White,L",  
 "S009,Green T-Shirt,Green,S",  
 "S010,Orange T-Shirt,Orange,S",  
 "S011,Maroon Polo Shirt,Maroon,S"  
 };  
  
  
 public static void main(String[] args) {  
 Shirt[] shirtsArr = new Shirt[*shirts*.length];  
 for (int i = 0; i < *shirts*.length; ++i)  
 shirtsArr[i] = Shirt.fromString(*shirts*[i]);  
  
 for (Shirt shirt : shirtsArr)  
 System.*out*.println(shirt);  
 }  
}

**Вывод программы:**

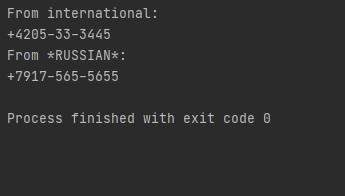
****

**Задача 4**

public class Numbers {  
 private final int countryCode;  
 private final int firstThree;  
 private final int secondThree;  
 private final int four;  
  
 private static final String *INTERNATIONAL\_REGEX* = "^[+][0**-**9][/0**-**9]{0,15}$";  
 private static final String *RUSSIAN\_REGEX* = "^[8][/0**-**9]{10}$";  
  
  
 private Numbers(int countryCode, int firstThree, int secondThree, int four) {  
 this.countryCode = countryCode;  
 this.firstThree = firstThree;  
 this.secondThree = secondThree;  
 this.four = four;  
 }  
  
  
 public String toUnique() {  
 return new StringBuffer().append("+").append(countryCode).append(firstThree)  
 .append("-").append(secondThree).append("-").append(four).toString();  
 }  
  
  
 public static Numbers fromInternational(String src) throws IllegalArgumentException {  
 Matcher matcher = Pattern.*compile*(*INTERNATIONAL\_REGEX*).matcher(src);  
 if (!matcher.matches())  
 throw new IllegalArgumentException("Wrong international phone number format.");  
  
 return new Numbers(  
 Integer.*parseInt*(String.*valueOf*(src.charAt(1))),  
 Integer.*parseInt*(src.substring(2, 5)),  
 Integer.*parseInt*(src.substring(5, 8)),  
 Integer.*parseInt*(src.substring(8, 12))  
 );  
 }  
  
  
 public static Numbers fromRussian(String src) throws IllegalArgumentException {  
 Matcher matcher = Pattern.*compile*(*RUSSIAN\_REGEX*).matcher(src);  
 if (!matcher.matches())  
 throw new IllegalArgumentException("Wrong russian phone number format.");  
  
 return new Numbers(  
 7,  
 Integer.*parseInt*(src.substring(1, 4)),  
 Integer.*parseInt*(src.substring(4, 7)),  
 Integer.*parseInt*(src.substring(7, 11))  
 );  
 }  
}

public class TestNumbers {  
  
 public static void main(String[] args) {  
 System.*out*.println("From international: ");  
 System.*out*.println(Numbers.fromInternational("+420503334455").toUnique());  
  
 System.*out*.println("From \*RUSSIAN\*:");  
 System.*out*.println(Numbers.fromRussian("89175655655").toUnique());  
 }  
}

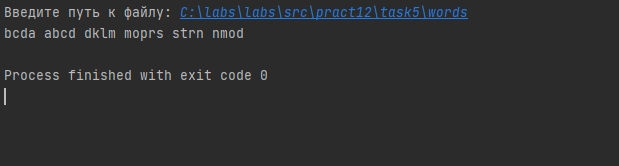
**Вывод программы:**

****

**Задача 5**

public class Words {  
  
 // Путь к файлу в виде: C:\Users\User\IdeaProjects\labs\src\pract14\task5\words  
 public static void main(String[] args) {  
 System.*out*.print("Введите путь к файлу: ");  
  
 String fileName;  
 String content;  
  
 try (BufferedReader reader = new BufferedReader(new InputStreamReader(System.*in*))) {  
 fileName = reader.readLine();  
 } catch (IOException e) {  
 System.*out*.println("Error. Invalid file name.");  
 return;  
 }  
 try {  
 content = Files.*readString*(Paths.*get*(fileName));  
 } catch (IOException e) {  
 System.*out*.println("Error. Invalid file.");  
 return;  
 }  
 String[] words = content.split(" ");  
  
 String result = *getLine*(words);  
 System.*out*.println(result);  
 }  
  
  
 public static String getLine(String... words) {  
 if (words == null || words.length == 0 || words[0].equals(""))  
 return "";  
 if (words.length == 1)  
 return words[0];  
 ArrayList<String> wordsList = new ArrayList<>(Arrays.*asList*(words));  
 while (wordsList.contains(""))  
 wordsList.remove("");  
 while (!*isContinues*(wordsList)) {  
 Collections.*shuffle*(wordsList);  
 }  
 StringBuilder result = new StringBuilder();  
 for (String word : wordsList)  
 result.append(word).append(" ");  
 result.deleteCharAt(result.length() - 1);  
 return result.toString();  
 }  
  
  
 public static boolean isContinues(ArrayList<String> wordsList) {  
 for (int i = 0; i < wordsList.size() - 1; ++i) {  
 String firstWord = wordsList.get(i).toLowerCase();  
 String secondWord = wordsList.get(i + 1).toLowerCase();  
 if (firstWord.toLowerCase().charAt(firstWord.length() - 1) != secondWord.toLowerCase().charAt(0))  
 return false;  
 }  
 return true;  
 }  
}

**Вывод программы:**

****

**Шестнадцатая практика**

**Задача:**

Задание 1

Создайте класс Drink – напитка. Класс описывает сущность – напиток и характеризуется следующими свойствами - стоимостью, названием и описанием. Класс должен быть определен как неизменяемый (Immutable class).

Задание 2

Создайте интерфейс Item – для работы с позициями заказа.

Задание 3

Создайте класс InternetOrder, который моделирует сущность интернет заказ в ресторане или кафе. Класс основан на циклическом двусвязном списке с выделенной головой и может хранить как блюда, так и напитки. Внимание: список реализуется самостоятельно.

public class Main {  
  
 public static void main(String[] args) {  
 TableOrder tableOrder = new TableOrder(20);  
  
 MenuItem someDrink = new Drink(40, DrinkTypeEnum.valueOf("VODKA"),  
 300, "Столичная водка", "RUSSIAN VODKA");  
 MenuItem someDish = new Dish(50000, "Фритатта", "Фритатта с лобстером и икрой");  
 MenuItem yetAnotherDrink = new Drink(0, DrinkTypeEnum.valueOf("JUICE"),  
 150, "Сок", "Сок апельсиновый 200мл.");  
 MenuItem yetAnotherDish = new Dish(30, "Доширак", "Лапша быстрого приготовления");  
  
  
  
 tableOrder.add(someDrink);  
 tableOrder.add(someDish);  
 tableOrder.add(yetAnotherDrink);  
 tableOrder.add(yetAnotherDish);  
 tableOrder.add(yetAnotherDish);  
  
 tableOrder.remove(yetAnotherDish);  
 //tableOrder.removeAll("Доширак");  
 //tableOrder.removeAll(yetAnotherDish);  
 tableOrder.sortedItemsByCostDesc();  
  
  
 System.*out*.println("Цена всех блюд и напитков: " + tableOrder.costTotal());  
  
 String[] itemsNames = tableOrder.itemsNames();  
 for (String name : itemsNames)  
 System.*out*.println(name);  
  
  
 }  
}

public final class Customer {  
  
 private final String firstName;  
 private final String secondName;  
 private final int age;  
 private final Address address;  
 private final Customer MATURE\_UNKNOWN\_CUSTOMER = new Customer(null, null, 18, Address.EMPTY\_ADDRESS);  
 private final Customer NOT\_MATURE\_UNKNOWN\_CUSTOMER = new Customer( null, null, 0, Address.EMPTY\_ADDRESS);  
  
 public Customer(String firstName, String secondName, int age, Address address) {  
 this.firstName = firstName;  
 this.secondName = secondName;  
 this.age = age;  
 this.address = address;  
 }  
  
 public String getFirstName() {  
 return firstName;  
 }  
 public String getSecondName() {  
 return secondName;  
 }  
 public int getAge() {  
 return age;  
 }  
 public Address getAddress() {  
 return address;  
 }  
}

public class Address {  
  
 private final String cityName;  
 private final int zipCode;  
 private final String streetName;  
 private final int buildingNumber;  
 private final char buildingLetter;  
 private final int apartmentNumber;  
 public static Address *EMPTY\_ADDRESS* =  
 new Address(null, 0, null,  
 0, '0', 0);  
  
  
 public Address(String cityName, int zipCode, String streetName, int buildingNumber, char buildingLetter, int apartmentNumber) {  
 this.cityName = cityName;  
 this.zipCode = zipCode;  
 this.streetName = streetName;  
 this.buildingNumber = buildingNumber;  
 this.buildingLetter = buildingLetter;  
 this.apartmentNumber = apartmentNumber;  
 }  
  
 public String getCityName() { return cityName; }  
 public int getZipCode() { return zipCode; }  
 public String getStreetName() { return streetName; }  
 public int getBuildingNumber() { return buildingNumber; }  
 public char getBuildingLetter() { return buildingLetter; }  
 public int getApartmentNumber() { return apartmentNumber; }  
}

public class MyArrayList<E> implements List<E> {  
  
 private int capacity;  
 private int count;  
 private Object[] arr;  
  
 private static final int *DEFAULT\_CAPACITY* = 10;  
  
 public MyArrayList() {  
 arr = new Object[*DEFAULT\_CAPACITY*];  
 capacity = *DEFAULT\_CAPACITY*;  
 count = 0;  
 }  
 public MyArrayList(int size) {  
 arr = new Object[0];  
 reserve(size);  
 }  
  
 private boolean indexInBounds(int index) {  
 return index >= 0 && index < count;  
 }  
  
 private void reserve(int newCapacity) {  
 var newArr = new Object[newCapacity];  
 System.*arraycopy*(arr, 0, newArr, 0, capacity);  
  
 arr = newArr;  
 capacity = newCapacity;  
 }  
  
 public int size() {  
 return count;  
 }  
 public boolean isEmpty() {  
 return count > 0;  
 }  
 public boolean contains(Object o) {  
 for (var elem : arr)  
 if (o.equals(elem))  
 return true;  
  
 return false;  
 }  
  
 public Iterator<E> iterator() {  
 return new Iterator<>() {  
 private int currentIndex = 0;  
 public boolean hasNext() {  
 return currentIndex + 1 <= count;  
 }  
 public E next() throws IndexOutOfBoundsException {  
 return (E)arr[currentIndex++];  
 }  
 };  
 }  
  
 public Object[] toArray() {  
 var result = new Object[count];  
 System.*arraycopy*(arr, 0, result, 0, count);  
  
 return result;  
 }  
  
 public <T> T[] toArray(T[] a) {  
 return (T[])toArray();  
 }  
  
 public boolean add(E t) {  
 if (capacity <= count)  
 reserve(capacity << 1);  
  
 arr[count++] = t;  
 return true;  
 }  
  
 public boolean remove(Object o) {  
  
 for (int i = 0; i < count; ++i) {  
 if (arr[i].equals(o)) {  
 // array shift to left  
 if (count - i + 1 >= 0)  
 System.*arraycopy*(arr, i + 1, arr, i + 1 - 1, count - i + 1);  
 --count;  
 return true;  
 }  
 }  
  
 return false;  
 }  
  
 public boolean containsAll(Collection<?> c) {  
 for (var cElem : c)  
 if (!contains(cElem))  
 return false;  
 return true;  
 }  
  
 public boolean addAll(Collection<? extends E> c) {  
 for (var cElem : c)  
 add(cElem);  
 return true;  
 }  
  
 public boolean addAll(int index, Collection<? extends E> c) {  
 for (var e : c)  
 add(index++, e);  
 return true;  
 }  
  
 public boolean removeAll(Collection<?> c) {  
 for (var e : c)  
 if (contains(e))  
 remove(e);  
 return true;  
 }  
  
 public boolean retainAll(Collection<?> c) {  
 for (int i = 0; i < count; ++i)  
 if (!c.contains(arr[i]))  
 remove(arr[i]);  
 return true;  
 }  
  
 public void clear() {  
 count = 0;  
 capacity = 0;  
 }  
  
 public E get(int index) {  
  
 if (!indexInBounds(index))  
 throw new IndexOutOfBoundsException();  
  
 return (E) arr[index];  
 }  
  
 public E set(int index, E element) {  
  
 if (!indexInBounds(index))  
 throw new IndexOutOfBoundsException();  
  
 var result = arr[index];  
 arr[index] = element;  
  
 return (E) result;  
 }  
  
 public void add(int index, E element) {  
 if (capacity < size() + 1)  
 reserve(capacity << 1);  
  
 // array shift to right  
 if (count - index >= 0)  
 System.*arraycopy*(arr, index, arr, index + 1, count - index);  
  
 arr[index] = element;  
 ++count;  
 }  
  
  
 @SuppressWarnings("unchecked")  
 public E remove(int index) {  
 if (!indexInBounds(index))  
 throw new IndexOutOfBoundsException();  
  
 var result = arr[index];  
 // array shift to left  
 if (count - index + 1 >= 0)  
 System.*arraycopy*(arr, index + 1, arr, index + 1 - 1, count - index + 1);  
 --count;  
  
 return (E) result;  
 }  
  
 public int indexOf(Object o) {  
 for (int i = 0; i < count; ++i)  
 if (arr[i].equals(o))  
 return i;  
 return -1;  
 }  
  
 public int lastIndexOf(Object o) {  
 for (int i = count - 1; i >= 0; --i)  
 if (arr[i].equals(o))  
 return i;  
 return -1;  
 }  
  
  
 public ListIterator<E> listIterator() {  
 return new ListIterator<E>();  
 }  
 public ListIterator<E> listIterator(int index) {  
 return new ListIterator<>(index);  
 }  
  
  
 public List<E> subList(int fromIndex, int toIndex) {  
 var result = new Object[count];  
 System.*arraycopy*(arr, 0, result, 0, count);  
  
 return (List<E>)Arrays.*asList*(result).subList(fromIndex, toIndex);  
 }  
  
  
 public E at(int index) throws IndexOutOfBoundsException {  
 if (indexInBounds(index))  
 return (E)arr[index];  
  
 throw new IndexOutOfBoundsException();  
 }  
  
  
 private class ListIterator<T> implements java.util.ListIterator<T> {  
  
 private int currentIndex = 0;  
  
  
 private ListIterator() {  
 currentIndex = 0;  
 }  
 private ListIterator(int currentIndex) {  
 this.currentIndex = currentIndex;  
 }  
 public boolean hasNext() {  
 return currentIndex + 1 <= count;  
 }  
 public T next() {  
 return (T)arr[currentIndex++];  
 }  
 public boolean hasPrevious() {  
 return count < currentIndex - 1;  
 }  
 public T previous() {  
 return (T)arr[--currentIndex];  
 }  
 public int nextIndex() {  
 return currentIndex;  
 }  
 public int previousIndex() {  
 return currentIndex - 1;  
 }  
 public void remove() {  
 MyArrayList.this.remove(currentIndex);  
 }  
 public void set(T e) {  
 MyArrayList.this.set(currentIndex, (E)e);  
 }  
 public void add(T e) {  
 MyArrayList.this.add((E)e);  
 }  
 }  
}

public class LinkedListNode<T> {  
 private T data;  
 private LinkedListNode<T> prev;  
 private LinkedListNode<T> next;  
  
  
 public LinkedListNode(T data) {  
 this.data = data;  
 }  
  
 public LinkedListNode(T data, LinkedListNode<T> prev, LinkedListNode<T> next) {  
 this.data = data;  
 this.prev = prev;  
 this.next = next;  
 }  
  
  
 public T getData() {  
 return data;  
 }  
 public LinkedListNode<T> getPrev() {  
 return prev;  
 }  
 public LinkedListNode<T> getNext() {  
 return next;  
 }  
  
 public void setData(T data) {  
 this.data = data;  
 }  
 public void setPrev(LinkedListNode<T> prev) {  
 this.prev = prev;  
 }  
 public void setNext(LinkedListNode<T> next) {  
 this.next = next;  
 }  
}

public class ItemsSorter {  
  
 public static void sort(Item[] items) {  
 int size = 0;  
 while (items[size] != null)  
 size++;  
  
 for (int i = 0; i < size - 1; ++i)  
 for (int j = i + 1; j < size; ++j)  
 if (items[i].getCost() < items[j].getCost()) {  
 var temp = items[i];  
 items[i] = items[j];  
 items[j] = temp;  
 }  
 }  
}

public class CycledLinkedList<E> {  
 protected LinkedListNode<E> current;  
 protected int size;  
  
 public static final int *INIT\_SIZE* = 0;  
  
 public static void main(String[] args) {  
 var test = new CycledLinkedList<Dish>();  
 var first = new Dish(1, "first", "123");  
 test.add(first);  
 test.add(new Dish(2, "second", "123"));  
 test.add(new Dish(3, "third", "123"));  
 test.add(new Dish(4, "second", "321"));  
  
 for (int i = 0; i < test.size; ++i)  
 System.*out*.println(test.prev());  
  
 test.next();  
 test.remove((item) -> item.getName().equals("second"));  
 test.remove((item) -> item.getName().equals("first"));  
 System.*out*.println("###");  
  
 for (int i = 0; i < test.size; ++i)  
 System.*out*.println(test.next());  
 }  
  
 public CycledLinkedList() {  
 this.current = null;  
 this.size = *INIT\_SIZE*;  
 }  
  
 public CycledLinkedList(E[] items) {  
 this.current = null;  
 this.size = *INIT\_SIZE*;  
  
 if (items.length == 0)  
 return;  
 for (E item : items)  
 add(item);  
 }  
  
 public boolean add(E item) {  
 LinkedListNode<E> newNode = new LinkedListNode<>(item);  
  
 if (current == null) {  
 current = newNode;  
 current.setPrev(current);  
 current.setNext(current);  
 }  
 else {  
 newNode.setNext(current.getNext());  
 newNode.setPrev(current);  
 current.getNext().setPrev(newNode);  
 current.setNext(newNode);  
 current = newNode;  
 }  
  
 ++size;  
 return true;  
 }  
  
  
 public boolean remove(IItemSelector<E> selector) {  
 for (int i = 0; i < size; ++i)  
 if (selector.selectIt(prev())) {  
 LinkedListNode<E> newCurrent = current.getNext();  
 current.getPrev().setNext(newCurrent);  
 newCurrent.setPrev(current.getPrev());  
 --size;  
 current = newCurrent;  
  
 return true;  
 }  
  
 return false;  
 }  
  
  
 public int removeAll(IItemSelector<E> selector) {  
 int count = 0;  
  
 for (int i = 0; i < size; ++i) {  
 if (selector.selectIt(next())) {  
 LinkedListNode<E> newCurrent = current.getNext();  
 current.getPrev().setNext(newCurrent);  
 newCurrent.setPrev(current.getPrev());  
 --size;  
 current = newCurrent;  
  
 ++count;  
 }  
 }  
  
 return count;  
 }  
  
  
 public E next() {  
 E result = null;  
  
 if (current != null) {  
 result = current.getNext().getData();  
 current = current.getNext();  
 }  
 return result;  
 }  
  
  
 public E prev() {  
 E result = null;  
  
 if (current != null) {  
 result = current.getPrev().getData();  
 current = current.getPrev();  
 }  
 return result;  
 }  
  
  
 public E[] toArray() {  
 var result = new Object[size];  
  
 prev();  
 for (int i = 0; i < size; ++i)  
 result[i] = next();  
 return (E[])result;  
 }  
  
 public int getSize() {  
 return size;  
 }  
}

public class TableOrdersManager implements OrdersManager {  
  
 private Order[] orders;  
 public void add(Order order, int tableNumber) { }  
 public void addItem(MenuItem item, int tableNumber) { }  
 public int freeTableNumber() {  
 return 0;  
 }  
 public int[] freeTableNumbers() {  
 return new int[0];  
 }  
 public Order getOrder(int tableNumber) {  
 return null;  
 }  
 public void remove(int tableNumber) { }  
 public int remove(Order order) {  
 return 0;  
 }  
 public int removeAll(Order order) {  
 return 0;  
 }  
 public int itemsQuantity(String itemName) {  
 return 0;  
 }  
  
 public int itemsQuantity(MenuItem item) {  
 return 0;  
 }  
  
 public Order[] getOrders() {  
 return new Order[0];  
 }  
  
 public int ordersCostSummary() {  
 return 0;  
 }  
  
 public int ordersQuantity() {  
 return 0;  
 }  
}

public class TableOrder implements Order {  
 private final int DEFAULT\_SIZE\_OF\_ARRAY = 10;  
 private int size;  
 private MenuItem[] items;  
  
 public TableOrder() {  
 this.items = new MenuItem[DEFAULT\_SIZE\_OF\_ARRAY];  
 this.size = 0;  
 }  
  
 public TableOrder(int arraySize)  
 {  
 this.items = new MenuItem[arraySize];  
 this.size = 0;  
 }  
  
 public boolean add(MenuItem item) {  
 if (size < items.length) {  
 items[size] = item;  
 size++;  
 return true;  
 }  
 return false;  
 }  
  
 public String[] itemsNames() {  
 if (size > 0)  
 {  
 String[] names = new String[size];  
 int i = 0;  
 while (i < size) {  
 names[i] = this.items[i].getName();  
 i++;  
 }  
 return names;  
 }  
 else  
 return null;  
 }  
  
 public int itemsQuantity() {  
 return this.size;  
 }  
 public int itemQuantity(String itemName) {  
 int countItem = 0;  
 int i = 0;  
  
 while (i < size) {  
 if (this.items[i].getName().equals(itemName))  
 countItem++;  
 i++;  
 }  
 return countItem;  
 }  
 public int itemQuantity(MenuItem item) {  
 int countItem = 0;  
 int i = 0;  
  
 while (i < size)  
 {  
 if (this.items[i] == item)  
 countItem++;  
 i++;  
 }  
 return countItem;  
 }  
  
 public MenuItem[] getItems() { return this.items; }  
 public boolean remove(String itemName) {  
 MenuItem[] new\_items = new MenuItem[this.items.length];  
 int i = 0;  
 int k = 0;  
 boolean isDelete = false;  
  
 if (itemQuantity(itemName) != 0)  
 {  
 while (k < size) {  
 if (!this.items[k].getName().equals(itemName) || isDelete) {  
 new\_items[i] = this.items[k];  
 i++;  
 k++;  
 }  
 else if (!isDelete)  
 {  
 isDelete = true;  
 k++;  
 }  
 }  
 this.items = new\_items;  
 this.size = i;  
 return true;  
 }  
 else  
 return false;  
 }  
 public boolean remove(MenuItem item) {  
 MenuItem[] new\_items = new MenuItem[this.items.length];  
 int i = 0;  
 int k = 0;  
 boolean isDelete = false;  
  
 if (itemQuantity(item) != 0)  
 {  
 while (k < size) {  
 if (this.items[k] != item || isDelete) {  
 new\_items[i] = this.items[k];  
 i++;  
 k++;  
 }  
 else if (!isDelete)  
 {  
 isDelete = true;  
 k++;  
 }  
 }  
 this.items = new\_items;  
 this.size = i;  
 return true;  
 }  
 else  
 return false;  
 }  
  
 public boolean removeAll(String itemName) {  
 MenuItem[] new\_items = new MenuItem[this.items.length];  
 int i = 0;  
 int k = 0;  
  
 if (itemQuantity(itemName) != 0)  
 {  
 while (k < size) {  
 if (!this.items[k].getName().equals(itemName)) {  
 new\_items[i] = this.items[k];  
 i++;  
 }  
 k++;  
 }  
 this.items = new\_items;  
 this.size = i;  
 return true;  
 }  
 else  
 return false;  
 }  
  
  
 public boolean removeAll(MenuItem item) {  
 MenuItem[] new\_items = new MenuItem[this.items.length];  
 int i = 0;  
 int k = 0;  
  
 if (itemQuantity(item) != 0)  
 {  
 while (k < size) {  
 if (this.items[k] != item) {  
 new\_items[i] = this.items[k];  
 i++;  
 }  
 k++;  
 }  
 this.items = new\_items;  
 this.size = i;  
 return true;  
 }  
 else  
 return false;  
 }  
 public MenuItem[] sortedItemsByCostDesc() {  
 ItemsSorter.sort(this.items);  
 return items;  
 }  
 public int costTotal() {  
 int total = 0;  
 int i = 0;  
  
 while (i < size){  
 if (items[i] != null)  
 total += items[i].getCost();  
 i++;  
 }  
 return total;  
 }  
 public Customer getCustomer() { return null; }  
 public void setCustomer(Customer customer) { }  
}

public class QueueNode {  
 private QueueNode next;  
 private QueueNode prev;  
 private Order value;  
}

public class InternetOrdersManager implements OrdersManager {  
 private QueueNode head;  
 private QueueNode tail;  
 private int size;  
  
  
 public boolean add(Order order) {  
 return false;  
 }  
  
 //public Order remove() { }  
 //public Order order() { }  
  
 public int itemsQuantity(String itemName) {  
 return 0;  
 }  
  
 public int itemsQuantity(MenuItem item) {  
 return 0;  
 }  
  
 public Order[] getOrders() {  
 return new Order[0];  
 }  
  
 public int ordersCostSummary() {  
 return 0;  
 }  
  
 public int ordersQuantity() {  
 return 0;  
 }  
}

public class InternetOrder extends CycledLinkedList<Item> implements Order {  
  
 private IItemSelector<Item> getSelectorByName(String name) {  
 return (item) -> item.getName().equals(name);  
 }  
  
 public Item[] select(IItemSelector<Item> selector) {  
 var result = new MyArrayList<Item>();  
  
 for (int i = 0; i < size; ++i) {  
 if (selector.selectIt(next()))  
 result.add(current.getData());  
 }  
 return (Item[])result.toArray();  
 }  
  
 public int countOf(IItemSelector<Item> selector) {  
 int count = 0;  
  
 for (int i = 0; i < size; ++i)  
 if (selector.selectIt(next()))  
 ++count;  
 return count;  
 }  
  
 public int getOrderPrice() {  
 int result = 0;  
  
 prev();  
 for (int i = 0; i < size; ++i)  
 result += next().getCost();  
 return result;  
 }  
  
 public String[] allNames() {  
 Item[] items = toArray();  
 MyArrayList<String> result = new MyArrayList<>(items.length);  
  
 for (Item item : items)  
 if (!result.contains(item.getName()))  
 result.add(item.getName());  
 return (String[])result.toArray();  
 }  
  
 public Item[] toSortedArray() {  
 Item[] items = toArray();  
 ItemsSorter.sort(items);  
 return items;  
 }  
  
 public boolean removeByName(String name) {  
 return remove(getSelectorByName(name));  
 }  
 public int removeAllByName(String name) {  
 return removeAll(getSelectorByName(name));  
 }  
 public int getCount() {  
 return size;  
 }  
 public int countOf(String name) {  
 return countOf(getSelectorByName(name));  
 }  
  
 public boolean add(MenuItem item) {  
 return false;  
 }  
  
 public String[] itemsNames() {  
 return new String[0];  
 }  
  
 public int itemsQuantity() {  
 return 0;  
 }  
  
 public int itemQuantity(String itemName) {  
 return 0;  
 }  
  
 public int itemQuantity(MenuItem item) {  
 return 0;  
 }  
  
 public MenuItem[] getItems() {  
 return new MenuItem[0];  
 }  
  
 public boolean remove(String itemName) {  
 return false;  
 }  
  
 public boolean remove(MenuItem item) {  
 return false;  
 }  
  
 public boolean removeAll(String itemName) {  
 return false;  
 }  
  
 public boolean removeAll(MenuItem item) {  
 return false;  
 }  
  
 public MenuItem[] sortedItemsByCostDesc() {  
 return new MenuItem[0];  
 }  
  
 public int costTotal() {  
 return 0;  
 }  
  
 public Customer getCustomer() {  
 return null;  
 }  
  
 public void setCustomer(Customer customer) {  
  
 }  
}

public class MenuItem implements Item, Comparable<MenuItem>{  
 protected int cost;  
 protected String name;  
 protected String description;  
  
 public int getCost() {  
 return cost;  
 }  
 public String getName() {  
 return name;  
 }  
 public String getDescription() {  
 return description;  
 }  
  
 @Override  
 public int compareTo(MenuItem o) {  
 return getName().compareTo(o.getName());  
 }  
}

public enum DrinkTypeEnum {  
 *BEER*,  
 *WINE*,  
 *VODKA*,  
 *BRANDY*,  
 *CHAMPAGNE*,  
 *WHISKEY*,  
 *TEQUILA*,  
 *RUM*,  
 *VERMUTH*,  
 *LIQUOR*,  
 *JAGERMEISTER*,  
 *JUICE*,  
 *COFFEE*,  
 *GREEN\_TEE*,  
 *BLACK\_TEA*,  
 *MILK*,  
 *WATER*,  
 *SODA*}

public final class Drink extends MenuItem implements Alcoholable {  
 private final double alcoholVol;  
 private final DrinkTypeEnum type;  
 private static final int *INIT\_COST* = 0;  
  
 public Drink(double alcoholVol, DrinkTypeEnum type, String name, String description) throws IllegalArgumentException {  
 if (name.equals("") || name.isEmpty() || description.equals("") || description.isEmpty())  
 throw new IllegalArgumentException();  
 this.type = type;  
 this.alcoholVol = alcoholVol;  
 this.cost = *INIT\_COST*;  
 this.name = name;  
 this.description = description;  
 }  
  
 public Drink(double alcoholVol, DrinkTypeEnum type, int cost, String name, String description) {  
 if (name.equals("") || name.isEmpty() || description.equals("") || description.isEmpty() || cost < 0)  
 throw new IllegalArgumentException();  
 this.type = type;  
 this.alcoholVol = alcoholVol;  
 this.cost = cost;  
 this.name = name;  
 this.description = description;  
 }  
  
 @Override  
 public boolean isAlcoholicDrink() {  
 return (this.alcoholVol > 0);  
 }  
  
 @Override  
 public double getAlcoholVol() {  
 return this.alcoholVol;  
 }  
}

public final class Dish extends MenuItem {  
 private static final int *INIT\_COST* = 0;  
  
 public Dish(String name, String description) throws IllegalArgumentException {  
 if (name.equals("") || name.isEmpty() || description.equals("") || description.isEmpty())  
 throw new IllegalArgumentException();  
  
 this.cost = *INIT\_COST*;  
 this.name = name;  
 this.description = description;  
 }  
  
 public Dish(int cost, String name, String description) throws IllegalArgumentException {  
 if (name.equals("") || name.isEmpty() || description.equals("") || description.isEmpty() || cost < 0)  
 throw new IllegalArgumentException();  
 this.cost = cost;  
 this.name = name;  
 this.description = description;  
 }  
}

public interface OrdersManager {  
  
 int itemsQuantity(String itemName);  
 int itemsQuantity(MenuItem item);  
 Order[] getOrders();  
 int ordersCostSummary();  
 int ordersQuantity();  
}

public interface Order {  
 boolean add(MenuItem item);  
 String[] itemsNames();  
 int itemsQuantity();  
 int itemQuantity(String itemName);  
 int itemQuantity(MenuItem item);  
 MenuItem[] getItems();  
 boolean remove(String itemName);  
 boolean remove(MenuItem item);  
 boolean removeAll(String itemName);  
 boolean removeAll(MenuItem item);  
 MenuItem[] sortedItemsByCostDesc();  
 int costTotal();  
 Customer getCustomer();  
 void setCustomer(Customer customer);  
}

public interface Item {  
 int getCost();  
 String getName();  
 String getDescription();  
}

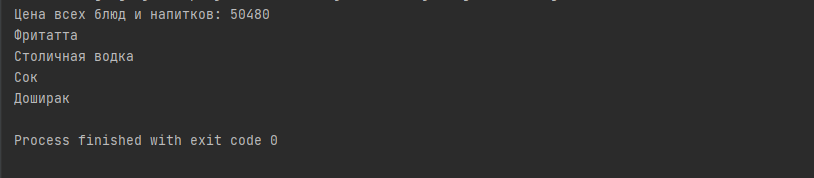
@FunctionalInterface  
public interface IItemSelector<E> {  
 boolean selectIt(E item);  
}

public interface Alcoholable {  
 boolean isAlcoholicDrink();  
 double getAlcoholVol();  
}

public class OrderAlreadyAddedException extends Exception {  
 private final Order addedOrder;  
 public Order getAddedOrder() {  
 return addedOrder;  
 }  
  
 public OrderAlreadyAddedException(String message) {  
 super(message);  
 addedOrder = null;  
 }  
  
 public OrderAlreadyAddedException(Order addedOrder) {  
 super("Order already added.");  
 this.addedOrder = addedOrder;  
 }  
  
 public OrderAlreadyAddedException(String message, Order addedOrder) {  
 super(message);  
 this.addedOrder = addedOrder;  
 }  
}

public class IllegalTableNumber extends RuntimeException {  
 public IllegalTableNumber(String message) {  
 super(message);  
 }  
 public IllegalTableNumber(int n) {  
 super("Illegal table number: " + n + ".");  
 }  
}

**Вывод программы:**

****