Методические рекомендации по выполнению лабораторных работ по дисциплине «Информационные технологии и программирование»

для студентов направления
09.03.01 Информатика и вычислительная техника

Лабораторная работа №1

Когда мы начинаем изучать язык программирования, первой программой всегда является вывод Hello World. Установим все необходимое и запустим программу на Java, которая выводит приветствие.

Hello World на Java

JavaHelloWorldProgram.java

```
public class JavaHelloWorldProgram {

    public static void main(String args[]){
        System.out.println("Hello World");
    }
}
```

Сохраните приведенную выше программу под именем JavaHelloWorldProgram.java в каталоге для первой лабораторной работы.

Откройте командную строку и перейдите в каталог, в котором сохранен файл программы. Затем выполните следующие команды по порядку.

```
> java JavaHelloWorldProgram.java
```

Программа скомпилируется и запустится, отобразится результат.

Стоит запомнить, что:

- Исходный файл Java может содержать несколько классов, но допускается только один public (это модификатор доступа) класс.
- Имя исходного файла java должно совпадать с именем public класса. Поэтому имя файла нашей программы - JavaHelloWorldProgram.java.
- Когда мы компилируем код, он генерирует байт-код и сохраняет его как расширение Class_Name.class. Если посмотреть на каталог, в котором мы скомпилировали java-файл, то можно заметить новый созданный файл JavaHelloWorldProgram.class
- При выполнении файла класса нам не нужно указывать полное имя файла. Нам нужно использовать только публичное имя класса.
- Когда мы запускаем программу с помощью команды java, она загружает класс в JVM, ищет в классе метод main и запускает его. Синтаксис метода main должен быть таким же, как указан в примере, иначе программа не будет запущена и выбросит исключение Exception в потоке "main" java.lang.NoSuchMethodError: main.

Задания для выполнения лабораторной работы:

Задание 1 Создайте программу, которая находит и выводит все простые числа меньше 100.

Создайте файл с именем Primes.java, в этом файле опишите следующий класс:

```
public class Primes {
    public static void main(String[] args) {
    }
}
```

Внутри созданного класса, после метода main(), опишите метод IsPrime (int n), который определяет, является ли аргумент простым числом или нет. Можно предположить, что входное значение n всегда будет больше 2. Полное описание метода будет выглядеть так:

```
public static boolean isPrime(int n) {
}
```

Данный метод вы можете реализовать по вашему усмотрению, однако простой подход заключается в использовании цикла for. Данный цикл должен перебирать числа, начиная с 2 до (но не включая) n, проверяя существует ли какое-либо значение, делящееся на n без остатка. Для этого можно использовать оператора остатка «%».Если какая-либо переменная полностью делится на аргумент, сработает оператор return false. Если же значение не делится на аргумент без остатка, то это простое число, и сработает оператор return true.

После того, как это будет реализовано, приступайте к заполнению основного метода main() другим циклом, который перебирает числа в диапазоне от 2 до 100 включительно. Необходимо вывести на экран те значения, которые метод IsPrime () посчитал простыми.

После завершения написания программы скомпилируйте и протестируйте её. Убедитесь, что результаты правильные.

Задание 2 **Создайте программу, которая определяет, является ли введенная строка палиндромом.**

Для этой программы, создайте класс с именем Palindrome в файле под названием Palindrome.java. На этот раз вы можете воспользоваться следующим кодом:

```
public class Palindrome {
	public static void main(String[] args) {
		for (int i = 0; i < args.length; i++) {
			String s = args[i];
		}
	}
}
```

Ваша первая задача состоит в том, чтобы создать метод, позволяющий полностью изменить символы в строке. Заголовок метода должен быть следующим:

```
public static String reverseString(Strings)
```

Вы можете реализовать этот метод путем создания локальной переменной, которая является пустой строкой, а затем добавлять символы из входной строки в выходные данные, в обратном порядке. Используйте метод length(), который возвращает длину строки, и метод charAt(int index), который возвращает символ по указанному индексу. Индексы начинаются с 0 и увеличиваются на 1.

После того, как вы создали метод reverseString (), создайте еще один метод

```
public static boolean isPalindrome(String s)
```

Этот метод должен перевернуть слово s, а затем сравнить с первоначальными данными. Используйте метод equals (Object) для проверки значения равенства. Например:

```
String s1 = "hello";
```

String s2 = "Hello";
String s3 = "hello";
s1.equals(s2); // Истина
s1.equals(s3); // Ложь

Для сравнения строк нельзя использовать оператор «==».

Скомпилируйте и протестируйте программу. Для того чтобы передать аргументы в программу, необходимо указать их при запуске программы через командную строку:

java Palindrome madam racecar apple kayak song noon

Лабораторная работа №2

Основными концепциями ООП являются:

- Абстракция
- Инкапсуляция
- Полиморфизм
- Наследование

Java позволяет описывать объекты (как и любой ООП язык). В данной лабораторной работе необходимо использовать классы, чтобы описать, как эти объекты работают. Вот код для простого класса, который представляет двумерную точку:

Point2d.java

```
public class Point2d {
	/ ** координата X **/
	private double xCoord;
	/ ** координата Y **/
	private double yCoord;
	/ ** Конструктор инициализации **/
	public Point2d (double x, double y) {
		xCoord = x;
		yCoord = y;
	}
	/ ** Конструктор по умолчанию. **/
	public Point2d () {
		this(0, 0);
	}
	/ ** Возвращение координаты X **/
	public double getX () {
		return xCoord;
	}
	/ ** Возвращение координаты Y **/
	public double getY () {
		return yCoord;
	}
	/ ** Установка значения координаты X. **/
	public void setX ( double val) {
		xCoord = val;
	}
	/ ** Установка значения координаты Y. **/
	public void setY ( double val) {
		yCoord = val;
	}
}
```

Экземпляр класса можно также создать, вызвав любой из реализованных конструкторов, например:

Point2d myPoint = new Point2d (); //создает точку (0,0)
Point2d myOtherPoint = new Point2d (5,3); //создает точку (5,3)
Point2d aThirdPoint = new Point2d ();

При реализации класса трехмерной точки можно использовать описанный выше класс двумерной точки. Для этого необходимо указать

что класс Point3d наследует класс Point2d с помощью ключевого слова extends, и далее реализовать функционал, который присущ классу Point3d, но отличается от функционала класса Point2d.

Реализация класса Point3D будет выглядеть следующим образом:

Point3d.java

```
public class Point3d {
	/ ** координата Z **/
	private double zCoord;
	/ ** Конструктор инициализации **/
	public Point3d (double x, double y, double z){
		super (x, y);
		zCoord=z;
	}
	/ ** Конструктор по умолчанию. **/
	public Point3d() {
		this (0,0,0);
	}
	/ ** Возвращение координаты Z **/
	public double getZ () {
		return zCoord;
	}
	/ ** Установка значения координаты Z. **/
	public void setZ (double val) {
		zCoord = val;
	}
}
```

Весь функционал класса Point2d присутствует в классе Point3d, поэтому необходимо добавить только взаимодействие с 3 координатой. Попробуйте запустить код и создать экземпляры классов Point2d и Point3d, а также установить/получить значения всех координат.

Стоит запомнить, что:

- Важнейшим преимуществом наследования является повторное использование кода, поскольку подклассы наследуют переменные и методы суперкласса.
- Приватные члены суперкласса недоступны для подкласса напрямую, но могут быть косвенно доступна через методы (геттеры и сеттеры).
- Члены суперкласса с модификатором доступа по умолчанию доступны подклассу только в том случае, если они находятся в одном пакете.
- Конструкторы суперкласса не наследуются подклассом.
- Если суперкласс не имеет конструктора по умолчанию, то в подклассе также должен быть определен явный конструктор. В противном случае будет выброшено исключение времени компиляции. В конструкторе подкласса вызов конструктора суперкласса в этом случае обязателен и должен быть первым утверждением в конструкторе подкласса.

- Java не поддерживает множественное наследование, подкласс может расширять только один класс.
- Мы можем создать экземпляр подкласса и затем присвоить его переменной суперкласса, это называется upcasting (повышающее преобразование).
- Когда экземпляр суперкласса присваивается переменной подкласса, это называется downcasting (понижающее преобразование). Нам необходимо явно привести этот экземпляр к подклассу.
- Мы можем переопределить метод суперкласса в классе наследнике. Однако мы всегда должны аннотировать переопределенный метод аннотацией @Override. Компилятор будет знать, что мы переопределяем метод, и если что-то изменится в методе суперкласса, то мы получим ошибку компиляции, а не нежелательные результаты во время выполнения.
- Мы можем вызывать методы суперкласса и обращаться к переменным суперкласса, используя ключевое слово super. Это удобно, когда у нас есть одноименная переменная/метод в подклассе, но мы хотим получить доступ к переменной/методу суперкласса. Это также используется, когда в суперклассе и подклассе определены конструкторы и нам необходимо явно вызвать конструктор суперкласса.
- Мы можем использовать инструкцию instanceof для проверки наследования между объектами.
- В Java мы не можем расширять final классы.
- Если вы не собираетесь использовать суперкласс в коде, т.е. ваш суперкласс является просто базой для сохранения многократно используемого кода, то вы можете сделать его абстрактным классом, чтобы избежать ненужного инстанцирования клиентскими классами. Это также ограничит создание экземпляров базового класса.

Задания для выполнения лабораторной работы:

Задание 1 Создайте иерархию классов в соответствии с вариантом. Ваша иерархия должна содержать:

- абстрактный класс
- 2 уровня наследуемых классов (классы должны содержать в себе минимум 3 поля и 2 метода, описывающих поведение объекта)
- демонстрацию реализации всех принципов ООП (абстракция, модификаторы доступа, перегрузка, переопределение)
- наличие конструкторов (в том числе по умолчанию)
- наличие геттеров и сеттеров
- ввод/вывод информации о создаваемых объектах
- предусмотрите в одном из классов создание счетчика созданных объектов с использованием статической переменной, продемонстрируйте работу

Варианты для лабораторной работы №2

1 Базовый класс: Животные Дочерние классы: Кошка, Попугай, Рыбка	2 Базовый класс: Сотрудник Дочерние классы: Администратор, Программист, Менеджер	3 Базовый класс: Человек Дочерние классы: Студент, Преподаватель, Ассистент преподавателя
4 Базовый класс: Транспортное средство Дочерние классы: Легковой автомобиль, Грузовой автомобиль, Мотоцикл	5 Базовый класс Велосипед Дочерние классы: Горный велосипед, Детский велосипед, BMX	6 Базовый класс: Геометрическая фигура Дочерние классы: Шар, Параллелепипед, Цилиндр
7 Базовый класс: Книга Дочерние классы: Аудиокнига, Фильм, Мюзикл	8 Базовый класс: Мебель Дочерние классы: Стол, Стул, Кровать	9 Базовый класс: Монстр Дочерние классы: Гоблин, Русалка, Дракон
10 Базовый класс: Гаджеты Дочерние классы: Часы, Смартфон, Ноутбук	11 Базовый класс: Бытовая техника Дочерние классы: Холодильник, Посудомоечная машина, Пылесос	12 Базовый класс: Приложение Дочерние классы: Социальная сеть, Игра, Погода
13 Базовый класс: Оружие Дочерние классы: Меч, Лук, Волшебная палочка	14 Базовый класс: Заведение Дочерние классы: Кафе, Магазин, Библиотека	15 Базовый класс: Компьютерная периферия Дочерние классы: Клавиатура, Наушники, Графический планшет

Предметная область может быть выбрана другая по согласованию с преподавателем.

Лабораторная работа №3

Хэш-таблица – это структура данных, которая используется для хранения пар «ключ-значение». Она основана на идее хэш-функции, которая преобразует ключ в индекс массива, где хранится значение.

Например, если мы хотим хранить и получать значения по ключу для следующих пар: («apple», 5), («banana», 3), («orange», 7), то мы можем использовать хэш-таблицу следующим образом:

Hash table:
Index 0:
Index 1:
Index 2: («banana», 3)
Index 3: («orange», 7)
Index 4:
Index 5: («apple», 5)
Index 6:

Здесь ключи «apple», «banana» и «orange» были преобразованы в индексы массива с помощью хэш-функции. Значения хранятся в связанных списках, которые могут быть добавлены к соответствующему индексу.

Хэш-функция должна быть быстрой и однозначной, то есть каждому ключу должен соответствовать уникальный индекс. Если два разных ключа преобразуются в один и тот же индекс, то это называется коллизией.

Коллизии могут быть разрешены различными способами, например, с помощью метода цепочек или открытой адресации.

В методе цепочек каждый индекс массива содержит связанный список пар «ключ-значение». Если происходит коллизия, то новая пара добавляется в конец списка. При поиске значения по ключу нужно просмотреть все элементы списка, начиная с первого.

Например, если мы добавим еще одну пару («pear», 2), то наша хэш-таблица будет выглядеть следующим образом:

Hash table:
Index 0:
Index 1:
Index 2: («banana», 3) -> («pear», 2)
Index 3: («orange», 7)
Index 4:
Index 5: («apple», 5)
Index 6:

Здесь ключ «pear» также преобразован в индекс 2, где уже находится пара «banana» – поэтому новая пара добавляется в конец списка.

В Java хэш-таблицы реализованы в виде классов HashMap и Hashtable. Они имеют похожий интерфейс, но различаются в том, что HashMap не является потокобезопасным, а Hashtable является.

Задания для выполнения лабораторной работы:

Задание 1:

1. Создайте класс HashTable, который будет реализовывать хэш-таблицу с помощью метода цепочек.

2. Реализуйте методы put(key, value), get(key) и remove(key), которые добавляют, получают и удаляют пары «ключ-значение» соответственно.

3. Добавьте методы size() и isEmpty(), которые возвращают количество элементов в таблице и проверяют, пуста ли она.

Пример реализации метода put(key, value):

```
public void put(K key, V value) {
    int index = hash(key);
    if (table[index] == null) {
        table[index] = new LinkedList<Entry<K, V>>();
    }
    for (Entry<K, V> entry : table[index]) {
        if (entry.getKey().equals(key)) {
            entry.setValue(value);
            return;
        }
    }
    table[index].add(new Entry<K, V>(key, value));
    size++;
}
```

Задание 2:

Вариант 1: Реализация хэш-таблицы для хранения информации о студентах. Ключом является номер зачетной книжки, а значением - объект класса Student, содержащий поля имя, фамилия, возраст и средний балл. Необходимо реализовать операции вставки, поиска и удаления студента по номеру зачетки.

Вариант 2: Реализация хэш-таблицы для хранения информации о товарах в интернет-магазине. Ключом является артикул товара, а значением - объект класса Product, содержащий поля наименование, описание, цена и количество на складе. Необходимо реализовать операции вставки, поиска и удаления товара по артикулу.

Вариант 3: Реализация хэш-таблицы для хранения информации о заказах в интернет-магазине. Ключом является номер заказа, а значением - объект класса Order, содержащий поля дата заказа, список товаров и статус заказа. Необходимо реализовать операции вставки, поиска и удаления заказа по номеру. Также необходимо реализовать метод для изменения статуса заказа.

Вариант 4: Реализация хэш-таблицы для хранения информации о книгах в библиотеке. Ключом будет ISBN книги, а значением - объект класса Book, содержащий информацию о названии, авторе и количестве копий. Необходимо реализовать операции вставки, поиска и удаления книги по ISBN.

Вариант 5: Реализация хэш-таблицы для учета продуктов на складе. Ключом будет штрихкод товара, а значением - объект класса Product, содержащий информацию о названии, цене и доступном количестве. Необходимо реализовать операции вставки, поиска и удаления продукта по штрихкоду.

Вариант 6: Реализация хэш-таблицы для хранения контактов в телефонной книге. Ключом будет номер телефона, а значением - объект

класса Contact, содержащий имя, адрес электронной почты и дополнительные контактные данные. Необходимо реализовать операции вставки, поиска и удаления контактов по номеру телефона.

Вариант 7: Реализация хэш-таблицы для учета заказов в интернет-магазине. Ключом будет номер заказа, а значением - объект класса Order, содержащий информацию о товарах, адресе доставки и стоимости заказа. Необходимо реализовать операции вставки, поиска и удаления заказа по номеру заказа.

Вариант 8: Реализация хэш-таблицы для хранения информации о сотрудниках в компании. Ключом будет идентификационный номер сотрудника, а значением - объект класса Employee, содержащий данные о имени, должности и заработной плате. Необходимо реализовать операции вставки, поиска и удаления сотрудника по ID.

Вариант 9: Реализация хэш-таблицы для учета автомобилей в автопарке. Ключом будет номерной знак автомобиля, а значением - объект класса Car, содержащий информацию о марке, модели и годе выпуска. Необходимо реализовать операции вставки, поиска и удаления автомобиля по знаку.

Вариант 10: Реализация хэш-таблицы для хранения данных о заказах в ресторане. Ключом будет номер столика или заказа, а значением - объект класса Order, содержащий информацию о блюдах, стоимости и времени заказа. Необходимо реализовать операции вставки, поиска и удаления заказа по номеру столика.

Лабораторная работа №4

Исключения в Java являются механизмом обработки ошибок и нестандартных ситуаций во время выполнения программы. Они позволяют изолировать ошибки и обрабатывать их в специальном блоке кода, что делает программу более надежной и устойчивой к ошибкам.

В Java есть множество типов исключений, которые могут возникнуть в различных ситуациях. Например, классы IOException и FileNotFoundException используются для обработки ошибок ввода-вывода, а классы NullPointerException и ArrayIndexOutOfBoundsException используются для обработки ошибок, связанных с работой с объектами и массивами.

Кроме того, в Java можно создавать свои собственные исключения, наследуясь от класса Exception или его подклассов. Это позволяет создавать более гибкую и точечную обработку ошибок в своих программах.

При работе с исключениями важно учитывать, что блок try-catch не должен заменять проверку условий и предотвращение возможных ошибок на этапе написания кода. Он должен использоваться только для обработки неожиданных ошибок, которые могут возникнуть в процессе выполнения программы. Код, который может вызвать исключение, помещается в блок try. Если исключение возникает внутри блока try, то управление передается в блок catch, который обрабатывает исключение. Блок catch может содержать несколько разделов, каждый из которых обрабатывает определенный тип исключения.

```
try {
   // Код, который может вызвать исключение
} catch (ExceptionType1 e) {
   // Обработка исключения типа ExceptionType1
} catch (ExceptionType2 e) {
   // Обработка исключения типа ExceptionType2
} finally {
   // Код, который выполняется всегда, независимо от того, возникло ли исключение или нет
}
```

Блок finally необязателен, но если он присутствует, то код внутри него будет выполняться всегда, независимо от того, возникло ли исключение или нет.

Задания для выполнения лабораторной работы:

Задание 1:

Необходимо написать программу, которая будет находить среднее арифметическое элементов массива. При этом программа должна обрабатывать ошибки, связанные с выходом за границы массива и неверными данными (например, если элемент массива не является числом).

Пример реализации метода:

```
public class ArrayAverage {
```

```
        public static void main(String[] args) {
            int[] arr = {1, 2, 3, 4, 5};
            int sum = 0;
            try {
                //Обработка массива
                }
                //Вывод массива
            } catch (*исключение*) {
                //Вывод исключения
            } catch (*исключение*) {
                //Вывод исключения
            }
        }
    }
```

Задание 2:

Необходимо написать программу, которая будет копировать содержимое одного файла в другой. При этом программа должна обрабатывать возможные ошибки, связанные с:

Вариант 1	Вариант 2
Открытием и закрытием файлов	Чтение и записью файлов

Задание 3:

Создайте Java-проект для работы с исключениями. Для каждой из восьми задач, напишите свой собственный класс для обработки исключений. *Создайте обработчик исключений, который логирует информацию о каждом выброшенном исключении в текстовый файл.* Реализуйте следующие варианты задач:

Вариант 1: Создайте класс CustomDivisionException, который будет использоваться для обработки исключений при делении на ноль. Напишите программу, которая делит два числа, и, если происходит деление на ноль, выбрасывайте исключение CustomDivisionException.

Вариант 2: Создайте класс CustomAgeException, который будет использоваться для обработки недопустимых возрастов. Реализуйте программу, которая проверяет возраст пользователя с использованием этого класса, и, если возраст меньше 0 или больше 120, выбрасывайте исключение CustomAgeException.

Вариант 3: Создайте класс CustomFileNotFoundException, который будет использоваться для обработки исключения FileNotFoundException. Напишите программу для чтения файла, и, если файл не существует, выбрасывайте исключение CustomFileNotFoundException.

Вариант 4: Создайте класс CustomNumberFormatException, который будет использоваться для обработки исключения NumberFormatException. Реализуйте программу, которая пытается преобразовать строку в число (Integer.parseInt()), и, если строка не является числом, выбрасывайте исключение CustomNumberFormatException.

Вариант 5: Создайте класс CustomEmptyStackException, который будет использоваться для обработки исключения EmptyStackException. Напишите класс CustomStack, имитирующий стек, и, если происходит

попытка извлечь элемент из пустого стека, выбрасывайте исключение CustomEmptyStackException.

Вариант 6: Создайте класс CustomInputMismatchException, который будет использоваться для обработки исключения InputMismatchException. Напишите программу, которая считывает целое число с клавиатуры, и, если ввод пользователя не является числом, выбрасывайте исключение CustomInputMismatchException.

Вариант 7: Создайте класс CustomEmailFormatException, который будет использоваться для обработки недопустимого формата email-адреса. Реализуйте программу, которая проверяет формат email-адреса с использованием этого класса, и, если адрес не соответствует формату, выбрасывайте исключение CustomEmailFormatException.

Вариант 8: Создайте класс CustomUnsupportedOperationException, который будет использоваться для обработки исключения UnsupportedOperationException. Реализуйте программу, которая выполняет математические операции (сложение, вычитание, умножение, деление) с помощью собственного класса и выбрасывайте исключение CustomUnsupportedOperationException, если операция не поддерживается.

Лабораторная работа №5

Регулярные выражения (Regular expressions) — это специальный язык для поиска и обработки текстовой информации, который широко используется в программировании и компьютерных науках. В Java регулярные выражения реализованы в классе java.util.regex.Pattern.

Синтаксис регулярных выражений в Java состоит из специальных символов, которые обозначают определенные шаблоны текста. Например:

- "." обозначает любой символ
- "*" обозначает повторение предыдущего символа или группы символов
- "+" обозначает повторение предыдущего символа или группы символов один или более раз
- "?" обозначает, что предыдущий символ или группа символов может быть присутствовать или отсутствовать
- "^" обозначает начало строки
- "$" обозначает конец строки
- "[]" обозначает набор символов, которые могут быть использованы в данной позиции
- "[^]" обозначает набор символов, которые не могут быть использованы в данной позиции
- "\\b" обозначает границу слова
- "\\d" обозначает любую цифру
- "\\s" обозначает любой пробельный символ
- "\\w" обозначает любую букву или цифру

Кроме того, можно использовать скобки для группировки символов и задания приоритета операций. Например:

- "(abc)+" означает повторение группы символов "abc" один или более раз
- "(abc|def)" означает, что символы "abc" или "def" могут быть использованы в данной позиции

Также в Java можно использовать специальные символы для задания квантификаторов, которые определяют количество повторений символов или групп символов. Например:

- "{n}" означает точное количество повторений (например, "\\d{3}" означает три цифры подряд)
- "{n,}" означает, что символ или группа символов должны повторяться не менее n раз

В Java регулярные выражения реализованы в классе java.util.regex.Pattern, который имеет методы для компиляции регулярного выражения и поиска совпадений в тексте. Класс Matcher используется для поиска совпадений в тексте и замены найденных совпадений.

<u>Примеры использования регулярных выражений в Java:</u>

1. Поиск слова в тексте:

```
String text = "The quick brown fox jumps over the lazy dog";
Pattern pattern = Pattern.compile("fox");
```

```
Matcher matcher = pattern.matcher(text);
if (matcher.find()) {
   System.out.println("Match found!");
}
```

2. Замена слова в тексте:

```
String text = "The quick brown fox jumps over the lazy dog";
Pattern pattern = Pattern.compile("fox");
Matcher matcher = pattern.matcher(text);
String result = matcher.replaceAll("cat");
System.out.println(result);
```

3. Проверка корректности email-адреса:

```
String email = "example@mail.com";
Pattern pattern = Pattern.compile("\\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\\.[A-Z]{2,}\\b");
Matcher matcher = pattern.matcher(email);
if (matcher.matches()) {
   System.out.println("Valid email address!");
}
```

Задания для выполнения лабораторной работы:

Задание 1: Поиск всех чисел в тексте

Необходимо написать программу, которая будет искать все числа в заданном тексте и выводить их на экран. При этом программа должна использовать регулярные выражения для поиска чисел и обрабатывать возможные ошибки.

Пример реализации метода:

```
import java.util.regex.*;

public class NumberFinder {
   public static void main(String[] args) {
      String text = "The price of the product is $19.99";
      Pattern pattern = Pattern.compile("\\d+\\.\\d+");
      Matcher matcher = pattern.matcher(text);
      while (matcher.find()) {
         System.out.println(matcher.group());
      }
   }
}
```

Задание 2: Проверка корректности ввода пароля

Необходимо написать программу, которая будет проверять корректность ввода пароля. Пароль должен состоять из латинских букв и цифр, быть длиной от 8 до 16 символов и содержать хотя бы одну заглавную букву и одну цифру. При этом программа должна использовать регулярные выражения для проверки пароля и обрабатывать возможные ошибки.

Задание 3: Замена всех ссылок на гиперссылки
Необходимо написать программу, которая будет заменять все ссылки в заданном тексте на гиперссылки. При этом программа должна использовать регулярные выражения для поиска ссылок и замены и обрабатывать возможные ошибки.
Задание 4: Проверка корректности ввода IP-адреса
Необходимо написать программу, которая будет проверять корректность ввода IP-адреса. IP-адрес должен состоять из 4 чисел, разделенных точками, и каждое число должно быть в диапазоне от 0 до 255. При этом программа должна использовать регулярные выражения для проверки IP-адреса и обрабатывать возможные ошибки.
Задание 5: Поиск всех слов, начинающихся с заданной буквы
Необходимо написать программу, которая будет искать все слова в заданном тексте, начинающиеся с заданной буквы, и выводить их на экран. При этом программа должна использовать регулярные выражения для поиска слов и обрабатывать возможные ошибки.

Лабораторная работа №6

Коллекции в Java представляют собой классы и интерфейсы, которые служат для хранения и управления группами объектов. Они позволяют удобно и эффективно работать с данными, осуществлять поиск, добавление, удаление и изменение элементов коллекции. В Java существует несколько основных интерфейсов коллекций:

1. Collection - базовый интерфейс коллекций, который содержит методы для работы с элементами коллекции.
2. List - интерфейс, который представляет упорядоченную коллекцию элементов, которые могут содержать дубликаты.
3. Set - интерфейс, который представляет неупорядоченную коллекцию элементов, которые не могут содержать дубликаты.
4. Map - интерфейс, который представляет отображение ключ-значение.

ArrayList и LinkedList — это две коллекции в Java, которые используются для доступа по индексу элемента.

ArrayList — это список, реализованный на основе массива, а LinkedList — это классический связный список, основанный на объектах с ссылками между ними.

Различия между ArrayList и LinkedList заключаются в том, что ArrayList основан на массиве, а LinkedList на связном списке. ArrayList обеспечивает быстрый доступ к элементам по индексу, но медленно работает при добавлении или удалении элементов из середины списка. LinkedList обеспечивает быстрое добавление и удаление элементов из середины списка, но медленно работает при доступе к элементам по индексу.

Примеры использования коллекций в Java:

1. Создание списка и добавление элементов:

```
List<String> list = new ArrayList<> ();
list.add("apple");
list.add("banana");
list.add("orange");
```

2. Использование цикла для обхода элементов списка:

```
for (String fruit : list) {
   System.out.println(fruit);
}
```

3. Удаление элемента из списка:

```
list.remove("banana");
```

Дженерики в Java позволяют создавать обобщенные классы и методы, которые могут работать с различными типами данных. Они позволяют повысить безопасность и удобство программирования, так как компилятор проверяет типы данных на этапе компиляции. Примеры использования дженериков в коллекциях и методах:

1. Объявление списка с использованием дженериков:

```
List<Integer> numbers = new ArrayList<>();
```

2. Использование дженериков в методе:

```
public static <T> T getFirst(List<T> list) {
    return list.get(0);
}
```

3. Использование дженериков в интерфейсе:

```
public interface Pair<K, V> {
    K getKey();
    V getValue();
}
```

Итераторы в Java представляют собой объекты, которые позволяют последовательно обходить элементы коллекции. Они позволяют безопасно и эффективно перебирать элементы коллекции, не заботясь о внутреннем устройстве коллекции. Примеры использования итераторов в коллекциях:

1. Использование итератора для обхода элементов списка:

```
Iterator<String> iterator = list.iterator();
while (iterator.hasNext()) {
    String fruit = iterator.next();
    System.out.println(fruit);
}
```

2. Удаление элемента из списка с помощью итератора:

```
Iterator<String> iterator = list.iterator();
while (iterator.hasNext()) {
    String fruit = iterator.next();
    if (fruit.equals("banana")) {
        iterator.remove();
    }
}
```

Задания для выполнения лабораторной работы:

Задание 1:

Написать программу, которая считывает текстовый файл и выводит на экран топ-10 самых часто встречающихся слов в этом файле. Для решения задачи использовать коллекцию Map, где ключом будет слово, а значением - количество его повторений в файле.

Пример реализации:

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.*;
```

```
public class TopWords {
    public static void main(String[] args) {
        // указываем путь к файлу
        String filePath = "C:\\text.txt";
        // создаем объект File
        File file = new File(filePath);
        // создаем объект Scanner для чтения файла
        Scanner scanner = null;
        try {
            scanner = new Scanner(file);
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }
        // создаем объект Map для хранения слов и их количества
        ******
        // читаем файл по словам и добавляем их в Map
        ******
        // закрываем Scanner
        ******
        // создаем список из элементов Map
        ******
        // сортируем список по убыванию количества повторений
        Collections.sort(list, new Comparator<Map.Entry<String, Integer>>() {
            @Override
            public int compare(Map.Entry<String, Integer> o1, Map.Entry<String,
Integer> o2) {
                ******
            }
        });
        // выводим топ-10 слов
        ******
            }
             //выводим результат
            *******
        }
    }
}
```

Задание 2:

Написать обобщенный класс Stack<T>, который реализует стек на основе массива. Класс должен иметь методы push для добавления элемента в стек, pop для удаления элемента из стека и peek для получения верхнего элемента стека без его удаления.

Пример реализации:

```
public class Stack<T> {
    private T[] data;
    private int size;
```

```
    public Stack(int capacity) {
        data = (T[]) new Object[capacity];
        size = 0;
    }

    public void push(T element) {
        ******
    }

    public T pop() {
        ******
    }

    public T peek() {
        ******
    }
}
```

Пример использования:

```
    Stack<Integer> stack = new Stack<>(10);
    stack.push(1);
    stack.push(2);
    stack.push(3);
    System.out.println(stack.pop());
    System.out.println(stack.peek());
    stack.push(4);
    System.out.println(stack.pop());
```

Задание 3:

Необходимо разработать программу для учета продаж в магазине. Программа должна позволять добавлять проданные товары в коллекцию, выводить список проданных товаров, а также считать общую сумму продаж и наиболее популярный товар.

Варианты выполнения задания:

Вариант	*Задача*
1	Использовать ArrayList для хранения списка проданных товаров.
2	Использовать LinkedList для хранения списка проданных товаров.
3	Использовать HashSet для хранения списка проданных товаров.
4	Использовать TreeSet для хранения списка проданных товаров.
5	Использовать HashMap для хранения пар "товар-количество продаж".
6	Использовать TreeMap для хранения пар "товар-количество продаж".

7	Использовать LinkedHashMap для хранения пар "товар-количество продаж".
8	Использовать ConcurrentHashMap для хранения пар "товар-количество продаж".
9	Использовать ConcurrentHashMap и AtomicInteger для счетчика количества продаж каждого товара.
10	Использовать CopyOnWriteArrayList для хранения списка проданных товаров с возможностью одновременного чтения и записи.

Лабораторная работа №7

Многопоточность в Java позволяет выполнять несколько задач одновременно в рамках одного приложения. Это позволяет увеличить производительность и сократить время выполнения программы.

Пример создания потока в Java:

```
public class MyThread extends Thread {
    public void run() {
        System.out.println("Hello from thread!");
    }
}

public class Main {
    public static void main(String[] args) {
        MyThread thread = new MyThread();
        thread.start();
    }
}
```

В данном примере создается класс MyThread, который наследуется от класса Thread. В методе run() задается код, который будет выполняться в потоке. В классе Main создается объект класса MyThread и запускается метод start(), который запускает новый поток.

Пример использования интерфейса Runnable:

```
public class MyRunnable implements Runnable {
    public void run() {
        System.out.println("Hello from runnable!");
    }
}

public class Main {
    public static void main(String[] args) {
        MyRunnable runnable = new MyRunnable();
        Thread thread = new Thread(runnable);
        thread.start();
    }
}
```

В данном примере создается класс MyRunnable, который реализует интерфейс Runnable. В методе run() задается код, который будет выполняться в потоке. В классе Main создается объект класса MyRunnable и передается в конструктор класса Thread. Затем запускается метод start(), который запускает новый поток.

Пример синхронизации доступа к общим ресурсам:

```
public class Counter {
    private int count;
    public synchronized void increment() {
        count++;
    }

    public int getCount() {
        return count;
```

```
    }
}

public class MyThread extends Thread {
    private Counter counter;

    public MyThread(Counter counter) {
        this.counter = counter;
    }

    public void run() {
        for (int i = 0; i < 1000; i++) {
            counter.increment();
        }
    }
}

public class Main {
    public static void main(String[] args) {
        Counter counter = new Counter();
        MyThread thread1 = new MyThread(counter);
        MyThread thread2 = new MyThread(counter);
        thread1.start();
        thread2.start();
        try {
            thread1.join();
            thread2.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println("Count: " + counter.getCount());
    }
}
```

В данном примере создается класс Counter, который имеет метод increment(), который увеличивает счетчик на 1. Метод increment() объявлен как synchronized, что позволяет синхронизировать доступ к общему ресурсу. В классе MyThread создаются два потока, которые используют один и тот же объект класса Counter. В методе run() каждый поток вызывает метод increment() 1000 раз. В классе Main создается объект класса Counter и два объекта класса MyThread, которым передается этот объект. Затем запускаются потоки и ожидается их завершение с помощью метода join(). После завершения потоков выводится значение счетчика.

Задание 1.

Реализация многопоточной программы для вычисления суммы элементов массива.

Вариант 1. Создать два потока, которые будут вычислять сумму элементов массива по половинкам, после чего результаты будут складываться в главном потоке.

Вариант 2. Создать пул потоков с помощью класса ExecutorService и разделить массив на равные части, каждую из которых будет обрабатывать отдельный поток. После завершения работы всех потоков результаты будут складываться в главном потоке.

Задание 2.

Реализация многопоточной программы для поиска наибольшего элемента в матрице.

Вариант 1. Создать несколько потоков, каждый из которых будет обрабатывать свою строку матрицы. После завершения работы всех потоков результаты будут сравниваться в главном потоке для нахождения наибольшего элемента.

Вариант 2. Создать пул потоков с помощью класса ExecutorService и разделить матрицу на равные части, каждую из которых будет обрабатывать отдельный поток. После завершения работы всех потоков результаты будут сравниваться в главном потоке для нахождения наибольшего элемента.

Задание 3:

У вас есть склад с товарами, которые нужно перенести на другой склад. У каждого товара есть свой вес. На складе работают 3 грузчика. Грузчики могут переносить товары одновременно, но суммарный вес товаров, которые они переносят, не может превышать 150 кг. Как только грузчики соберут 150 кг товаров, они отправятся на другой склад и начнут разгружать товары.

Напишите программу на Java, используя многопоточность, которая реализует данную ситуацию.

Варианты:

Вариант	*Задание*
1	Использование Thread: Создайте классы Товар, Склад, и Грузчик. Каждый грузчик должен быть представлен в виде отдельного потока.
2	Использование Runnable: Создайте интерфейс Грузчик и реализуйте его в классе LoaderRealization. Используйте ExecutorService для управления потоками.
3	Использование Lock и Condition: Используйте блокировки и условия для синхронизации работы грузчиков.
4	Использование Semaphore: Используйте семафоры для ограничения доступа к складу и контроля над весом товаров.
5	Использование CompletableFuture: Используйте CompletableFuture для асинхронного выполнения задачи переноса товаров.
6	Использование ForkJoinPool: Разделите склад на подзадачи и используйте Fork-Join пул для обработки этих подзадач.
7	Использование ReentrantLock и Condition: Используйте реентерабельные блокировки для управления доступом к ресурсам.

8	Использование CountDownLatch: Используйте CountDownLatch для синхронизации начала и завершения переноса товаров.
9	Использование CyclicBarrier: Используйте барьеры для синхронизации грузчиков перед отправлением на другой склад.
10	Использование Executor и CompletionService: Используйте Executor для управления потоками и CompletionService для получения результатов выполнения.

Лабораторная работа №8

Аннотации (Annotations)

Аннотации — это метаданные, которые могут быть присоединены к классам, методам, полям и другим элементам программы в Java. Аннотации позволяют добавлять дополнительную информацию к коду и могут использоваться для анализа и автоматизации процесса компиляции.

Аннотации используются для различных целей, таких как:

- Пометка кода для анализа сторонними инструментами.
- Отображение информации во время выполнения программы (аннотации с RetentionPolicy.RUNTIME).
- Управление поведением компилятора.

Пример аннотации:

```
public @interface MyAnnotation {
    String value() default "";
}
```

Stream API

Stream API предоставляет богатый набор операций для обработки и манипуляции данными в коллекциях в функциональном стиле. Он упрощает многие типичные задачи обработки данных, такие как фильтрация, отображение, сортировка и агрегирование.

Пример Stream API:

```
List<Person> people = // Инициализация списка людей
List<Person> olderThan30 = people.stream()
    .filter(person -> person.getAge() > 30)
    .collect(Collectors.toList());
```

Пример сортировки:

```
List<String> fruits = Arrays.asList("apple", "banana", "cherry", "date", "elderberry");
List<String> sortedFruits = fruits.stream()
        .sorted()
        .collect(Collectors.toList());
System.out.println(sortedFruits);  // Output: [apple, banana, cherry, date, elderberry]
```

java.util.concurrent

java.util.concurrent предоставляет множество классов и интерфейсов для работы с многопоточностью. Он включает в себя средства для создания потоков, управления потоками, а также синхронизации и координации между потоками.

Пример java.util.concurrent:

```
ConcurrentCounter counter = new ConcurrentCounter();
ExecutorService executorService = Executors.newFixedThreadPool(5);
```

```
for (int i = 0; i < 10; i++) {
    executorService.submit(() -> {
        counter.increment();
    });
}
executorService.shutdown();
executorService.awaitTermination(10, TimeUnit.SECONDS);
System.out.println("Counter value: " + counter.getValue());
```

Создание потоков с помощью ExecutorService:

```
ExecutorService executorService = Executors.newFixedThreadPool(3);
Runnable task = () -> {
    System.out.println("Task running in thread: " +
Thread.currentThread().getName());
};
executorService.execute(task);
```

Задание для выполнения лабораторной работы:

Вам необходимо разработать приложение, которое считывает данные из исходного источника (например, файл, база данных или сетевой ресурс), применяет к ним различные операции с использованием Stream API, и сохраняет результаты в новый источник данных.

1. Создайте аннотацию @DataProcessor, которая будет использоваться для пометки методов обработки данных.
2. Создайте класс DataManager, который будет отвечать за многопоточную обработку данных. Этот класс должен иметь методы:
    - registerDataProcessor(Object processor): Регистрирует объект-обработчик данных с аннотацией @DataProcessor.
    - loadData(String source): Загружает данные из исходного источника.
    - processData(): Запускает многопоточную обработку данных, применяя методы с аннотацией @DataProcessor с использованием Stream API.
    - saveData(String destination): Сохраняет обработанные данные в новый источник.
3. Создайте несколько классов, представляющих различные обработчики данных, и пометьте их аннотацией @DataProcessor. Например, можно создать классы для фильтрации, трансформации и агрегации данных.
4. Используйте многопоточность из java.util.concurrent для эффективной обработки данных параллельно.
5. Протестируйте ваше приложение, загрузив данные из исходного источника, применив различные обработчики с помощью Stream API, и сохраните результаты в новый источник.