МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
Пятигорский институт (филиал) СКФУ
Колледж Пятигорского института (филиал) СКФУ

ПМ.02 ПРОЕКТИРОВАНИЕ УПРАВЛЯЮЩИХ ПРОГРАММ КОМПЬЮТЕРНЫХ СИСТЕМ И КОМПЛЕКСОВ
МДК.02.03 РАЗРАБОТКА ПРИКЛАДНЫХ ПРИЛОЖЕНИЙ

МЕТОДИЧЕСКИЕ УКАЗАНИЯ ДЛЯ ЛАБОРАТОРНЫХ РАБОТ

Специальности СПО

09.02.01 Компьютерные системы и комплексы

Пятигорск 2026

Методические указания для лабораторных работ по дисциплине МДК.02.03 Разработка прикладных приложений составлены в соответствии с требованиями ФГОС СПО. Предназначены для студентов, обучающихся по специальности 09.02.01 Компьютерные системы и комплексы.

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

Программа дисциплины Разработка кода информационных систем предусматривает изучение алгоритмизации как базовой составляющей технологического процесса создания программного продукта, а так же изучение языка программирования и формирование навыков разработки ИС, объектно-ориентированного программирования у студентов. При изучении предмета следует соблюдать единство терминологии и обозначения в соответствии с действующими стандартами, Международной системной единицы (СИ).

В результате изучения дисциплины студенты *должны:*

знать:

- основные виды и процедуры обработки информации, модели и методы решения задач обработки информации;
- основные платформы для создания, исполнения и управления информационной системой;
- основные процессы управления проектом разработки;
- основные модели построения информационных систем, их структуру, особенности и области применения;
- методы и средства проектирования, разработки и тестирования информационных систем;
- систему стандартизации, сертификации и систему обеспечения качества продукции.

уметь:

- осуществлять постановку задач по обработке информации;
- проводить анализ предметной области;
- осуществлять выбор модели и средства построения информационной системы и программных средств;
- использовать алгоритмы обработки информации для различных приложений;
- решать прикладные вопросы программирования и языка сценариев для создания программ;
- разрабатывать графический интерфейс приложения;
- создавать и управлять проектом по разработке приложения;
- проектировать и разрабатывать систему по заданным требованиям и спецификациям. **иметь практический опыт в**:
- управлении процессом разработки приложений с использованием инструментальных средств;
- обеспечении сбора данных для анализа использования и функционирования информационной системы;
- программировании в соответствии с требованиями технического задания;
- использовании критериев оценки качества и надежности функционирования информационной системы;
- применении методики тестирования разрабатываемых приложений;

-определении состава оборудования и программных средств разработки информационной системы;

- разработке документации по эксплуатации информационной системы;
- проведении оценки качества и экономической эффективности информационной системы в рамках своей компетенции;
- модификации отдельных модулей информационной системы.

В результате освоения учебной дисциплины студент должен овладевать:

Общими компетенциями:

ОК 01. Выбирать способы решения задач профессиональной деятельности, применительно к различным контекстам.

ОК 02. Использовать современные средства поиска, анализа и интерпретации информации, и информационные технологии для выполнения задач профессиональной деятельности.

ОК 04. Эффективно взаимодействовать и работать в коллективе и команде.

ОК 09. Пользоваться профессиональной документацией на государственном и иностранном языках.

Профессиональными компетенциями:

ПК 2.1. Проектировать, разрабатывать и отлаживать программный код модулей управляющих программ.

ПК 2.2. Владеть методами командной разработки программных продуктов.

ПК 2.3. Выполнять интеграцию модулей в управляющую программу.

ПК 2.4. Тестировать и верифицировать выпуски управляющих программ.

ПК 2.5. Выполнять установку и обновление версий управляющих программ (с учетом миграции - при необходимости).

ОБЩИЕ МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ВЫПОЛНЕНИЮ ЛАБОРАТОРНЫХ РАБОТ

По дисциплине Разработка кода информационных систем лабораторные работы содержат задачи и теоретические вопросы. Варианты для каждого обучающегося - индивидуальные.

Задачи и ответы на вопросы, выполненные не по своему варианту, не засчитываются.

Лабораторная работа выполняется в отдельной тетради. Условия задачи и формулировки вопросов переписываются полностью. Формулы, расчеты, ответы на вопросы пишутся ручкой, а чертежи, схемы и рисунки выполняются карандашом, на графиках и диаграммах указывается масштаб. Вначале задача решается в общем виде, затем делаются расчёты по условию задания. Решение задач обязательно ведется в Международной системе единиц (СИ).

При выполнении лабораторной работы необходимо следовать методическим указаниям: повторить краткое содержание теории, запомнить основные формулы и законы, проанализировать пример выполнения аналогичного задания, затем преступить непосредственно к решению задачи. К зачету допускаются студенты, получившие положительные оценки по всем лабораторным работам.

Правила выполнения лабораторной работы.

1. Студент должен прийти на лабораторную работу подготовленным к выполнению лабораторной работы.
2. Каждый студент после проведения работы должен представить отчет о проделанной работе с анализом полученных результатов и выводом по работе.
3. Таблицы и рисунки следует выполнять с помощью чертежных инструментов (линейки, циркуля, и.т.д.) карандашом с соблюдением ЕСКД.
4. Расчет следует проводить с точностью до двух значащих цифр.

5. Исправления проводить на обратной стороне листа. При мелких исправлениях неправильное слово (буква, число и т.п.) аккуратно зачеркивается и над ним пишут правильное пропущенное слово (букву, число и т.п.).

6. Вспомогательные расчеты можно выполнять на отдельных листах, а при необходимости на листах отчета.

7. Если студент не выполнит лабораторную работу или часть работы, то он выполнит ее во внеурочное время, согласованное с преподавателем.

8. Оценку по лабораторной работе студент получает с учетом срока выполнения работы, если;

- расчеты выполнены правильно и в полном объеме;
- сделан анализ проделанной работы и вывод по результатам работы;
- студент может пояснить выполнение любого этапа работы;
- отчет выполнен в соответствии с требованиями к выполнению работы.

Лабораторная работа №1. Методы без параметров в учебном проекте

Цель: освоить создание и использование методов без параметров в Java, понять их роль в структурировании кода и повышении его читаемости.

Основные понятия: методы, параметры.

Теоретическая часть.

Метод в Java - это блок кода, который выполняет определённое действие. Его можно вызывать многократно.

Метод без параметров - метод, который не принимает никаких входных данных при вызове. Он выполняет действие, опираясь только на свои внутренние инструкции или на данные класса.

Синтаксис метода без параметров:

```
[модификатор доступа] [тип возвращаемого значения] [имя метода]() {
   // тело метода — код, который выполняется при вызове
}
```

Пример:

```
public void sayHello() {
   System.out.println("Привет, мир!");
}
```

Здесь:

- public — модификатор доступа (метод доступен из любого другого класса);
- void — тип возвращаемого значения (метод ничего не возвращает);
- sayHello — имя метода;
- () — скобки без параметров (метод без параметров);
- {...} — тело метода.

Зачем нужны методы без параметров?

- Повторное использование кода. Один и тот же код можно вызывать много раз.
- Упрощение структуры программы. Код разбивается на логические блоки.
- Повышение читаемости. По имени метода сразу понятно, что он делает.

Задание

Создайте простую Java- программу, состоящую из одного класса GreetingApp. Программа должна демонстрировать вызов трёх методов без параметров.

- Шаг 1. Создайте класс GreetingApp
  В вашей IDE создайте новый Java- класс с именем GreetingApp.

- Шаг 2. Определите три метода без параметров
  Внутри класса напишите следующие методы:

  1. printHello() — выводит в консоль фразу «Привет!».
  2. printGoodbye() — выводит в консоль фразу «Пока!».
  3. printProgramInfo() — выводит информацию о программе: «Программа демонстрации методов без параметров».

- Шаг 3. Напишите метод main
  В том же классе создайте метод main — точку входа в программу.
  Внутри него последовательно вызовите все три созданных метода.

- Шаг 4. Запустите и протестируйте программу

Контрольные вопросы

1. Что такое метод в Java?
2. В чём особенность метода *без параметров*?
3. Приведите пример такого метода из вашей программы.

Лабораторная работа №2. Методы с параметрами в учебном проекте.

Цель: Ознакомление с концепцией передачи аргументов методам в языке Java, приобретение навыков работы с передачей значений и ссылок, закрепление понимания областей видимости переменных внутри методов.

Теоретический часть

Метод в Java представляет собой фрагмент кода, выполняющий определённую операцию над объектом или классом. Одним из важнейших аспектов работы с методами является передача параметров — переменных или констант, используемых методом для выполнения вычислений.

Параметры передаются в метод либо по значению (primitive types), либо по ссылке (reference types). Для примитивных типов (например, int, double) изменения значения параметра внутри метода не влияют на исходное значение вне метода. Объекты же передаются по ссылке, следовательно, любые изменения атрибутов объекта сохраняются и после выхода из метода.

```
public class Example {
    public static void main(String[] args) {
        int a = 10;
        changePrimitive(a); // a останется равным 10

        StringBuilder sb = new StringBuilder("Hello");
        changeReference(sb); // теперь sb равно "World"

        System.out.println("a=" + a);
        System.out.println("sb=" + sb.toString());
    }

    private static void changePrimitive(int x) {
        x += 5;
    }

    private static void changeReference(StringBuilder str) {
        str.replace(0, str.length(), "World");
    }
}
```

Вывод:
a=10
sb=World

Передача параметров:

1. Передача по значению: Используется для примитивных типов (int, float, boolean и др.). Изменения внутри метода никак не повлияют на исходное значение переменной.
2. Передача по ссылке: Применяется для объектов (все классы являются объектами). Внутри метода возможно изменение состояния объекта, которое сохраняется и после возврата из метода.

Задание

Создайте класс Calculator, содержащий статический метод calculateSum, принимающий два целых числа и возвращающий их сумму.
Создайте второй метод calculateProduct, принимающий несколько чисел и возвращающий произведение элементов массива.

Контрольные вопросы

1. Что такое параметр метода?
2. Чем отличается передача параметров по значению от передачи по ссылке?
3. Как правильно передать массив в качестве параметра метода?
4. Может ли метод изменить значение локальной переменной, переданной ему в качестве аргумента?

Лабораторная работа №3. Оператор SWITCH,
цикл FOR, цикл WHILE в учебном проекте.

Цель: Ознакомление с оператором ветвления switch, конструкциями циклов for и while. Освоение практики написания конструкций ветвления и повторяющихся действий на языке Java.

Теоретическая часть.

Оператор switch позволяет выбрать одну из возможных ветвей исполнения кода исходя из значения целочисленного выражения. Каждый случай обозначается ключевым словом case, завершающимся оператором break.

Пример использования оператора switch:

```
int dayOfWeek = 3;
switch(dayOfWeek){
   case 1:
      System.out.println("Понедельник");
      break;
   case 2:
      System.out.println("Вторник");
      break;
   case 3:
      System.out.println("Среда");
      break;
   default:
      System.out.println("Ошибка!");
}
// Результат: "Среда"
```

Цикл for используется для выполнения последовательности инструкций фиксированное количество раз. Имеет следующую форму записи:

```
for (инициализация; условие продолжения; обновление) {
   тело_цикла;
}
```

Пример использования цикла for:

```
for (int i = 1; i <= 5; i++) {
   System.out.print(i + " ");
}
// Результат: "1 2 3 4 5 "
```

Цикл while выполняется пока истинно указанное выражение условия. Его синтаксис выглядит следующим образом:

```
while (условие) {
   тело_цикла;
}
```

Пример использования цикла while:

```
int count = 1;
while(count <= 5) {
   System.out.print(count + " ");
   count++;
}
// Результат: "1 2 3 4 5 "
```

Задание

Реализуйте программу, которая запрашивает у пользователя число месяца и выводит название соответствующего времени года используя оператор switch.

Контрольные вопросы

1. Какие типы выражений допускаются в условии оператора switch?
2. Чем отличаются друг от друга циклы for и while?
3. Почему важно указывать оператор break в каждом случае оператора switch?
4. Можно ли в цикле for пропустить инициализацию, проверку условия или инкрементацию?

Лабораторная работа №4. Объявление и обработка одномерного массива.

Цель: Освоение способов объявления и инициализации одномерных массивов в Java, практика выполнения стандартных операций с элементами массива, таких как вывод содержимого массива, поиск максимального элемента и суммирование элементов.

Теоретическая часть.

Одномерный массив — это последовательность однотипных элементов, хранящихся последовательно в памяти компьютера. Массивы объявляются следующим образом:

```
тип[] имяПеременной = new тип[размер];
```

или

```
тип[] имяПеременной = { элемент1, элемент2, ..., элементN };
```

Примеры объявлений:

```
int[] arrInt = new int[5]; // объявление массива из 5 элементов
char[] charArray = {'A', 'B', 'C'}; // инициализация массива литералом
```

Доступ к элементам массива осуществляется с помощью индекса, начиная с нуля:

```
arrInt[0] = 10; // присваиваем первому элементу значение 10
System.out.println(arrInt[2]); // вывод третьего элемента массива
```

Важные моменты:

- Индексация начинается с 0.
- Попытка обратиться к несуществующему элементу приведет к ошибке IndexOutOfBoundsException.
- Размер массива задаётся при объявлении и неизменяем.

Задание

Напишите программу на Java, которая создает одномерный массив целых чисел размера N (число вводится пользователем), заполняет этот массив случайными числами от 1 до 100 включительно, находит максимальный элемент массива и выводит его на экран вместе с суммой всех элементов массива.

Пример вывода:

Размер массива: 5
Массив: [45, 23, 89, 12, 67]
Максимальное значение: 89
Сумма элементов: 236

Контрольные вопросы

1. Как создать одномерный массив в Java?
2. Чем ограничены элементы массива?
3. Что произойдет, если попытаться обратиться к элементу массива с неправильным индексом?
4. Как получить длину массива в Java?

Лабораторная работа №5. Объявление и обработка двумерного массива.

Цель: Получение практических навыков по созданию, заполнению и обработке двумерных массивов в Java. Изучение особенностей доступа к элементам двумерного массива и проведение базовых операций с ним.

Теоретическая часть.

Двумерный массив — это массив массивов одинакового типа. Фактически, это таблица, состоящая из строк и столбцов. В Java двумерный массив создается следующим образом:

тип[][] имяПеременной = new тип[количествоСтрок][количествоСтолбцов];

Или с непосредственной инициализацией:

тип[][] имяПеременной = {{элемент11, элемент12}, {элемент21, элемент22}};

Например:

int[][] matrix = new int[3][4]; // создаем матрицу 3 строки × 4 столбца
Каждый элемент двумерного массива доступен по паре индексов [i][j],
где i — номер строки, а j — номер столбца.

Чтобы заполнить двумерный массив, можно воспользоваться вложенными циклами:

```
for (int i = 0; i < matrix.length; i++) {
   for (int j = 0; j < matrix[i].length; j++) {
      matrix[i][j] = someValue; // Присваиваем какое-то значение
   }
}
```

Также можно выводить содержимое массива:

```
for (int i = 0; i < matrix.length; i++) {
   for (int j = 0; j < matrix[i].length; j++) {
      System.out.print(matrix[i][j] + "\t");
   }
   System.out.println();
}
```

Задание

Напишите программу на Java, которая создаёт двумерный массив целых чисел размерности M×N (размерность задаёт пользователь), заполняет его случайными числами от 1 до 100 включительно и выводит максимальное значение среди всех элементов массива.

Контрольные вопросы

1. Как правильно объявляется двумерный массив в Java?
2. Сколько индексов используется для обращения к элементу двумерного массива?
3. Чем отличается длина двумерного массива по строке и длине отдельного внутреннего массива?
4. Как вывести двумерный массив на экран в удобочитаемом виде?

Лабораторная работа №6. Ввод массивов.

Цель: изучить способы ввода массивов в языке программирования Java, освоить работу с классом Scanner для получения данных от пользователя, научиться выводить массивы на экран.

Теоретическая часть.

В Java существует несколько способов ввода данных в массивы:

Через стандартный ввод (консоль):Используя класс Scanner, можно считывать вводимые пользователем данные непосредственно в массив.

```
import java.util.Scanner;

Scanner scanner = new Scanner(System.in);
int n = scanner.nextInt(); // Количество элементов массива
int[] array = new int[n];

for (int i = 0; i < n; i++) {
   array[i] = scanner.nextInt(); // Заполняем массив
}
```

Чтение из файла:Можно считать данные из внешнего файла с помощью класса BufferedReader или FileInputStream.

```
try (BufferedReader reader = new BufferedReader(new FileReader("data.txt")))
{
   String line = reader.readLine();
   if (line != null) {
      String[] values = line.split(",");
      double[] array =
Arrays.stream(values).mapToDouble(Double::parseDouble).toArray();
   }
} catch (IOException e) {
   e.printStackTrace();
}
```

Генерация случайных значений:Иногда удобно заполнить массив случайными значениями с помощью генератора случайных чисел.

```
Random random = new Random();
int[] array = new int[10];
for (int i = 0; i < array.length; i++) {
   array[i] = random.nextInt(100); // Генерируем случайные числа от 0 до 99
}
```

Задание

Напишите программу на Java, которая принимает размеры массива с клавиатуры, далее вводит сами элементы массива с клавиатуры и выводит массив на экран.

Контрольные вопросы

1. Какие способы ввода данных в массив существуют в Java?
2. Как организовать ввод элементов массива с клавиатуры?
3. Как заполнить массив случайными числами?
4. Возможны ли проблемы при чтении большого объема данных из файла? Если да, то какие меры предосторожности следует предпринять?

Лабораторная работа №7. Обработка строк: поиск, сравнение.

Цель: Изучение основных методов обработки строк в Java, таких как поиск подстрок, сравнение строк и преобразование регистров символов. Освоение работы с встроенными методами библиотеки Java для манипуляции строковыми данными.

Теоретическая часть.

Основные методы обработки строк:

1. Поиск подстроки:
   - indexOf(substring) — возвращает позицию первого появления указанной подстроки.
   - lastIndexOf(substring) — возвращает последнюю позицию указанной подстроки.
2. Сравнение строк:
   - equals(otherString) — сравнивает строки по символам (учитывая регистр).
   - compareTo(otherString) — лексикографическое сравнение двух строк.
   - contains(charSequence) — проверяет наличие указанной подстроки в строке.
3. Преобразование регистра:
   - toUpperCase() — преобразует строку в верхний регистр.
   - toLowerCase() — преобразует строку в нижний регистр.
4. Замена символов:
   - replace(oldChar, newChar) — заменяет символы в строке.
   - replaceAll(regex, replacement) — замена по регулярному выражению.
5. Длина строки:
   - length() — возвращает длину строки (количество символов).

Задание

Реализовать программу на Java, выполняющую следующие действия:

1. Запрашивает ввод двух строк у пользователя.
2. Определяет, содержится ли одна строка внутри другой.
3. Показывает позицию первой найденной подстроки.
4. Осуществляет точное сравнение введённых строк.

Контрольные вопросы

1. Чем отличаются методы indexOf() и lastIndexOf()?
2. Какое отличие между методами equals() и compareTo()?
3. Что означает возвращаемое значение метода indexOf(), равное "-1"?
4. Какие ещё полезные методы класса String существуют помимо указанных в работе?

Лабораторная работа №8. Обработка символов.

Цель: Ознакомление с основными способами работы с отдельными символами в Java, изучение различных функций для преобразования и анализа отдельных символов.

Теоретическая часть.

В Java символ представляется примитивным типом данных char. Он хранит один 16-битный символ Unicode.

Основные особенности типа char:

- диапазон значений: от \u0000 до \uFFFF (65 536 символов);
- символы заключаются в одинарные кавычки: 'A', '7', '@';
- можно задавать через Unicode-коды: '\u0041' (это буква A).

Класс Character — обёртка для типа char, содержит полезные статические методы для проверки и преобразования символов:

- Character.isLetter(ch) — проверяет, является ли символ буквой;
- Character.isDigit(ch) — проверяет, является ли символ цифрой;
- Character.isWhitespace(ch) — проверяет, является ли символ пробелом, табуляцией и т.д.;
- Character.toUpperCase(ch) — преобразует символ в верхний регистр;
- Character.toLowerCase(ch) — преобразует символ в нижний регистр.

Операции с символами:

- сравнение: ch1 == ch2;
- арифметические операции (работают с кодами символов): 'B' - 'A' == 1.

Важные моменты:

- тип char — примитивный, не объектный;
- символы Unicode позволяют работать с текстами на разных языках;
- класс Character предоставляет удобные методы для валидации и преобразования.

Задание

Напишите Java-программу, которая:

1. Запрашивает у пользователя ввод одного символа с клавиатуры.
2. Определяет тип введённого символа и выводит соответствующую информацию:
   - если это буква — сообщает «Это буква» и показывает её в верхнем и нижнем регистре;
   - если это цифра — сообщает «Это цифра» и вычисляет следующую цифру (например, для '5' выводит '6');
   - если это пробел — сообщает «Это пробел»;
   - для всех остальных символов — сообщает «Это специальный символ».
3. Завершает работу после обработки одного символа.

Контрольные вопросы

1. Как объявить переменную типа char и присвоить ей значение? Приведите пример объявления символа 'X'.
2. В чём разница между типами char и String в Java? Можно ли хранить один символ в String?
3. Для чего нужен класс Character? Назовите два метода этого класса и кратко опишите их назначение.
4. Как с помощью Scanner считать один символ с клавиатуры? Почему нельзя использовать nextChar() и какой метод применяется вместо него?

Лабораторная работа №9. Включение класса в учебный проект.

Цель: освоить процесс создания и включения пользовательского класса в Java- проект, понять принципы организации кода через классы, научиться создавать объекты и вызывать их методы.

Теоретическая часть.

Класс в Java – это шаблон (шаблон описания) для создания объектов, определяющий их структуру (поля) и поведение (методы).

Основные компоненты класса:

- Поля (переменные класса) — хранят данные объекта (например, name, age).
- Методы — определяют действия, которые может выполнять объект (например, introduce()).
- Конструктор — специальный метод для инициализации объекта при его создании (new).

Процесс включения класса в проект:

1. Создание файла с расширением .java (имя файла должно совпадать с имене м класса).
2. Объявление класса с помощью ключевого слова class.
3. Определение полей, методов и конструктора внутри класса.
4. Создание объекта класса в основном классе (Main) через оператор new.
5. Вызов методов объекта через точечную нотацию (object.method()).

Ключевые принципы:

- Каждый публичный класс должен находиться в отдельном файле с таким ж е именем.
- Для доступа к полям и методам объекта используется точечная нотация.
- Конструктор вызывается автоматически при создании объекта через new.

Задание

Создайте класс Student

В проекте создайте файл Student.java со следующим содержимым:

- Поля класса:
  - String name — имя студента;
  - int age — возраст студента;
  - String group — номер группы.
- Конструктор с тремя параметрами для инициализации полей.
- Метод introduce(), который выводит информацию о студенте в формате: «Я студент [имя], мне [возраст] лет, учусь в группе [группа].»

Создайте главный класс Main

- Создайте файл Main.java с методом main:
- В методе main создайте два объекта класса Student с разными данными.
- Для каждого объекта вызовите метод introduce().

Контрольные вопросы

1. Что такое класс в Java? Приведите пример класса из вашей программы.
2. Для чего нужен конструктор класса? Как он вызывается в программе?
3. Почему поля класса рекомендуется делать private? Как тогда получить доступ к их значения м?
4. Как создать объект класса в Java? Напишите синтаксис создания объекта и приведите приме р из вашей программы.

Лабораторная работа №10. Разработка приложения в соответствии с принципами объектно-ориентированного программирования.

Цель: закрепить понимание основных принципов объектно- ориентированного программирования (инкапсуляция, наследование, полиморфизм, абстракция) путём создания простого Java- приложения, демонстрирующего их применение.

Теоретическая часть.

Объектно-ориентированное программирование (ООП) —
подход к разработке программ, где основными элементами являются объекты, вза имодействующие друг с другом.

Основные принципы ООП:

1. Инкапсуляция —
   сокрытие внутренней реализации объекта и предоставление доступа к данн ым только через методы. Достигается с помощью модификаторов доступа (private, protected, public) и геттеров/сеттеров.
2. Наследование —
   возможность класса наследовать поля и методы другого класса с помощью ключевого слова extends. Позволяет переиспользовать код и строить иерарх ии классов.
3. Полиморфизм —
   способность объектов разных классов обрабатывать данные по-разному, ис пользуя один и тот же интерфейс. В Java реализуется через переопределени е методов (@Override) и использование интерфейсов.
4. Абстракция —
   выделение общих характеристик и скрытие ненужных деталей. Реализуется через абстрактные классы (с ключевым словом abstract) и интерфейсы (inte rface).

В Java эти принципы позволяют создавать модульный, гибкий и легко подд ерживаемый код.

Задание

Разработайте простое приложение для учёта транспортных средств, демонстриру ющее все четыре принципа ООП.

Шаг 1. Создайте абстрактный класс Vehicle

- Поля: String brand, int year.
- Конструктор с параметрами для инициализации полей.
- Абстрактный метод void startEngine().
- Метод void displayInfo(), выводящий информацию о транспортном средстве.

Шаг 2. Создайте два класса-наследника:

- Car —
  наследуется от Vehicle. Добавьте поле int numberOfDoors. Переопределите метод s tartEngine(), чтобы он выводил «Автомобиль [brand] заводит двигатель с характер ным звуком Vroom!».
- Motorcycle —
  наследуется от Vehicle. Добавьте поле boolean hasSidecar. Переопределите метод st artEngine(), чтобы он выводил «Мотоцикл [brand] заводит двигатель с резким звук ом Vroom-vroom!».

Шаг 3. Создайте главный класс Main

- В методе main создайте по одному объекту классов Car и Motorcycle.
- Для каждого объекта:
  - вызовите метод displayInfo();
  - вызовите метод startEngine().

Контрольные вопросы

1. Какой принцип ООП демонстрирует использование абстрактного класса Vehicle? Объясните , почему.
2. Как в данной программе реализуется принцип наследования? Приведите конкретный пример из кода.
3. В чём проявляется полиморфизм в вашем приложении? Укажите, какие методы демонстрир уют это свойство и как.
4. Как принцип инкапсуляции реализован в классах Car и Motorcycle? Опишите, какие модифи каторы доступа использованы и зачем.

Лабораторная работа №11. Обработка потоков в учебном проекте.

Цель: Изучить основы работы с потоками ввода-вывода в Java, научиться читать и записывать данные в файлы, освоить использование классов FileInputStream, FileOutputStream, BufferedReader и PrintWriter.

Теоретическая часть.

Потоки ввода-вывода используются для передачи данных между приложениями и внешними источниками данных, такими как файлы, сети и устройства. Основные классы для работы с файлами в Java находятся в пакете java.io.

Классы для чтения и записи файлов:

FileInputStream / FileOutputStream: Эти классы предназначены для низкоуровневой работы с байтами. Их удобно использовать для бинарных файлов.

BufferedReader / BufferedWriter: Позволяют эффективно считывать и записывать большие объемы данных. Для удобства часто используются совместно с PrintWriter и Scanner.

PrintWriter: Удобный способ вывода текстовых данных в файл или консоль.

Примеры использования некоторых классов:

```
// Чтение файла с использованием BufferedReader
try (BufferedReader reader = new BufferedReader(new FileReader("file.txt"))) {
    String line;
    while ((line = reader.readLine()) != null) {
        System.out.println(line);
    }
} catch (IOException e) {
    e.printStackTrace();
}

// Запись в файл с использованием PrintWriter
try (PrintWriter writer = new PrintWriter("output.txt")) {
    writer.println("Записываем данные в файл.");
} catch (FileNotFoundException e) {
    e.printStackTrace();
}
```

Задание

Создать приложение на Java, которое выполняет следующие шаги:

1. Читает содержимое существующего текстового файла.
2. Преобразует каждую строчку файла таким образом, чтобы первая буква каждого слова была заглавной.
3. Сохраняет полученный результат в новый файл.

Контрольные вопросы

1. В чём разница между классами FileInputStream и BufferedReader?
2. Когда лучше использовать класс PrintWriter, а когда FileOutputStream?
3. Зачем нужны буферизованные потоки (BufferedReader)?
4. Может ли случиться ошибка во время открытия файла? Если да, какая исключительная ситуация возникает?

Лабораторная работа №12. Обработка файлов в учебном проекте.

Цель: Освоить базовые приёмы работы с файлами в Java, включая создание, чтение, модификацию и удаление файлов, изучить стандартные инструменты библиотеки java.io для манипуляций с файлами.

Теоретическая часть.

Файлы являются важным элементом любого программного проекта, позволяющим сохранять и извлекать данные вне зависимости от жизненного цикла приложения. Работа с файлами в Java осуществляется посредством специальных классов пакета java.io.

Ключевые понятия и классы для работы с файлами:

- File: Класс представляет объект, соответствующий физическому файлу или каталогу на диске. Используется для управления файлом (создание, переименование, удаление), но не позволяет непосредственно читать или писать данные.
- FileInputStream и FileOutputStream: Классы для чтения и записи бинарных данных в виде потока байтов.
- BufferedReader и BufferedWriter: Используются для эффективного чтения и записи больших объемов текстовых данных. Применяются чаще всего с объектами Reader и Writer.
- PrintWriter: Простой способ записать текстовые данные в файл или вывести их на консоль.

Примеры использования классов:Создание нового файла:

```
File file = new File("example.txt");
if (!file.exists()){
   try{
      file.createNewFile(); // Создаем файл
   }catch(Exception ex){}
}
```

Чтение данных из файла:

```
BufferedReader br = new BufferedReader(new FileReader(file));
String line;
while((line=br.readLine())!=null){
   System.out.println(line);
}
br.close();
```

Запись данных в файл:

```
PrintWriter pw = new PrintWriter(file);
pw.println("Это новая строка в файле.");
pw.close();
```

Задание

Разработать программу на Java, которая делает следующее:

- Предлагает пользователю ввести название файла.
- Открывает указанный файл и отображает его содержание на экране.
- Запрашивает новую строку текста у пользователя и добавляет её в конец файла.

Контрольные вопросы

1. Чем отличается класс File от классов InputStream и OutputStream?
2. Какие типы ошибок могут возникнуть при попытке создать или прочитать файл?
3. Как проверить существование файла в Java?

Лабораторная работа №13. Доработка приложения с учетом обработки файлов

Цель: Научиться интегрировать обработку файлов в существующее приложение на Java, освоив возможности стандартных библиотек для работы с файлами, а также умение правильно организовывать сохранение и загрузку данных из файлов.

Теоретическая часть.

При разработке приложений важно уметь сохранить состояние программы или важные данные пользователя. Одним из распространённых способов является хранение информации в файлах

Ключевые концепции и классы для работы с файлами в Java:

- File: Представляет собой ссылку на реальный файл или директорию на устройстве. Не предназначен для непосредственного чтения или записи данных.
- FileInputStream / FileOutputStream: Потоки для низкоуровневого чтения и записи данных в виде последовательностей байтов.
- BufferedReader / BufferedWriter: Предоставляют возможность удобной и эффективной работы с большими объёмами текстовых данных.
- PrintWriter: Упростит процесс записи данных в текстовый файл.

Читаем данные из файла:

```
BufferedReader reader = new BufferedReader(new FileReader("data.txt"));
String line;
while((line = reader.readLine()) != null){
   System.out.println(line);
}
reader.close();
```

Записываем данные в файл:

```
PrintWriter writer = new PrintWriter("output.txt");
writer.println("Данные успешно сохранены.");
writer.close();
```

Удаляем файл:

```
File file = new File("old_file.txt");
if(file.delete()){
   System.out.println("Файл удалён успешно.");
}else{
   System.out.println("Ошибка удаления файла.");
}
```

Задание

Дополнить существующую программу, предназначенную для учёта книг, возможностью сохранения списка книг в файл и загрузки данных обратно. Реализуйте следующие пункты:

1. Добавьте меню опций: сохранить список книг в файл, загрузить список книг из файла.
2. При сохранении всех книг добавьте соответствующую информацию в файл.
3. После загрузки файла убедитесь, что загруженный список корректно отображается в программе.

Контрольные вопросы

1. Как проверить, существует ли файл в директории?
2. В чем различие между классами File и FileInputStream?
3. Почему рекомендуется использовать BufferedReader и BufferedWriter при работе с большим объемом данных?
4. Какие ошибки могут возникнуть при создании или открытии файла, и как их обработать?

Лабораторная работа №14. Использование коллекций в учебном проекте

Цель: Изучить особенности работы с коллекциями в Java, познакомиться с различными интерфейсами коллекций и особенностями их реализаций, применить полученные знания на практике путем доработки учебного проекта.

Теоретическая часть.

Коллекции в Java представляют собой набор объектов определенного типа, предоставляя удобный механизм работы с динамическими наборами элементов. Наиболее распространенные интерфейсы коллекций включают List, Set и Map. Интерфейс List обеспечивает упорядоченное хранение элементов с доступом по индексу, например ArrayList или LinkedList. Интерфейс Set запрещает дублирование элементов и поддерживает уникальные значения, например HashSet или TreeSet. Интерфейс Map связывает ключи с соответствующими значениями, обеспечивая быстрый доступ по ключу, примером служит HashMap или TreeMap.

Рассмотрим простейший пример использования коллекции ArrayList: создаем массив чисел и добавляем элементы с помощью метода add(), затем проходим по списку и выводим каждый элемент.

Задание

Улучшить учебный проект добавлением возможностей работы с коллекциями. Необходимо заменить обычные массивы на коллекции, обеспечив гибкость и удобство работы с элементами. Важно продемонстрировать применение разных видов коллекций (списки, множества, карты) в зависимости от требований конкретного сценария.

Контрольные вопросы

1. Чем отличается коллекция List от массива в Java?
2. Перечислите ключевые различия между ArrayList и LinkedList.
3. Какие ограничения накладывает использование коллекции Set и почему это полезно?
4. Расскажите о структуре и назначении карт (Map) в Java и приведите пример использования.

Лабораторная работа №15. Реализация параметризованного интерфейса в учебном проекте.

Цель: Научиться использовать параметризованные интерфейсы в Java, понимать их назначение и преимущества, а также грамотно внедрять обобщённые интерфейсы в структуру учебных проектов.

Теоретическая часть.

Параметризованный интерфейс (также называемый generic interface) позволяет разработчику абстрагироваться от конкретных типов данных, делая код универсальным и применимым ко множеству типов одновременно. Такие интерфейсы принимают формальные параметры типов, обозначаемые угловыми скобками <>. Например, стандартный интерфейс Collection<E> объявляется с параметром E, означающим произвольный тип элемента коллекции. При реализации такого интерфейса конкретные типы подставляются автоматически, что повышает безопасность и удобство разработки.

Простым примером использования параметризованных интерфейсов является реализация своего собственного интерфейса Stack<T>, где T обозначает тип хранимых элементов стека. Затем мы реализуем этот интерфейс классом MyStack, указывая желаемый тип данных, например MyStack<Integer>.

Задание

Разработать свою собственную реализацию параметризованного интерфейса для учебной задачи. Пусть это будет интерфейс SortableCollection<T>, представляющий коллекцию элементов с поддержкой сортировки. Требуется реализовать этот интерфейс собственным классом SortedArrayList<T>. Продемонстрируйте использование вашего класса на примерах сортировок различных типов данных.

Контрольные вопросы

1. В чем преимущество использования параметризованных интерфейсов перед обычными?
2. Могут ли параметры типов быть ограничены определёнными категориями классов? Если да, то каким образом это делается?
3. Как обрабатываются случаи несоответствия типов при компиляции и выполнении кода с параметрами типов?
4. Приведите пример правильного объявления и реализации параметризованного интерфейса, пояснив необходимость именно такого подхода.

Лабораторная работа №16. Создание форм. Добавление кнопок, меток, текстовых полей. Интерфейс формы и размещение компонентов.

Цель: Ознакомиться с основами построения графического интерфейса пользователя (GUI) в Java, научившись добавлять компоненты окна (кнопки, метки, поля ввода), размещать их на форме и обрабатывать события нажатия кнопок.

Теоретическая часть.

Графический интерфейс пользователя (Graphical User Interface, GUI) играет ключевую роль в современных приложениях, позволяя пользователям взаимодействовать с системой визуально удобным способом. В Java для построения GUI широко применяется библиотека Swing, предоставляющая разнообразные компоненты для окон, кнопок, полей ввода и прочего.

Компоненты интерфейса размещаются на контейнерах, например JPanel, JFrame, которые управляют их расположением с помощью менеджеров размещения LayoutManager. Например, FlowLayout располагает компоненты слева направо сверху вниз, GridLayout распределяет компоненты равномерно по сетке, а BorderLayout делит контейнер на пять областей (верхняя, нижняя, левая, правая и центральная).

Вот простой пример формы с двумя полями ввода имени и пароля, меткой и кнопкой входа:

Создаем окно с названием "Форма входа":

```
JFrame frame = new JFrame("Форма входа");
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setSize(300, 200);
```

Добавляем панель и компоненты:

```
JPanel panel = new JPanel();
JLabel labelName = new JLabel("Имя:");
JTextField fieldName = new JTextField(10);
JLabel labelPassword = new JLabel("Пароль:");
JPasswordField fieldPassword = new JPasswordField(10);
```

```
JButton buttonLogin = new JButton("Войти");
panel.add(labelName);
panel.add(fieldName);
panel.add(labelPassword);
panel.add(fieldPassword);
panel.add(buttonLogin);
```

Настройка менеджера расположения:

```
frame.getContentPane().add(panel);
frame.pack();
frame.setVisible(true);
```

Таким образом создается простая форма с базовым набором элементов.

Задание

Создайте форму с заголовком "Регистрация пользователя", содержащую поля для ввода имени, электронной почты и пароля, а также кнопки регистрации.
Используйте подходящий менеджер размещения для аккуратного распределения компонентов.
При нажатии на кнопку регистрации выведите сообщение с введенными данными.

Контрольные вопросы

1. Какие основные компоненты графического интерфейса доступны в библиотеке Swing?
2. В чем состоит задача менеджера размещения (layout manager)? Какие виды менеджеров вам известны?
3. Как зарегистрировать событие нажатия кнопки в Java?
4. Какие свойства компонента JTextField задают ширину поля ввода и режим отображения пароля?

Лабораторная работа №17. Разработка кода обработки событий в учебном проекте.

Цель: Изучить механизм обработки событий в Java, разобраться в особенностях модели делегирования событий и научиться разрабатывать обработчики событий для компонентов графического интерфейса пользователя (GUI).

Теоретическая часть.

В Java обработка событий основана на модели делегирования, согласно которой компонент генерирует событие, а специальный слушатель (listener) получает уведомление о событии и соответствующим образом реагирует на него. Существует большое количество предопределенных слушателей, таких как ActionListener, MouseListener, KeyListener и др., предназначенных для обработки действий мыши, клавиатуры и прочих событий.

Процесс обработки событий включает три шага:

- Компонент создает событие.
- Компонент уведомляет зарегистрированных слушателей.
- Листы получают событие и выполняют соответствующее действие.

Пример реализации обработчика события клика на кнопку с использованием анонимного внутреннего класса:

```
JButton myButton = new JButton("Нажмите меня");myButton.addActionListener(new ActionListener(){    public void actionPerformed(ActionEvent event){        JOptionPane.showMessageDialog(null,"Вы нажали кнопку!");    }});
```

Другой способ — использование лямбда-выражений (начиная с Java 8):

```
button.addActionListener(e -> JOptionPane.showMessageDialog(null, "Кликнули кнопку!"));
```

Задание

Реализовать простой калькулятор, содержащий четыре кнопки ("+", "-", "*", "/") и поле для ввода значений. По нажатию соответствующей кнопки вычислить результат выбранной арифметической операции над введенным числом и другим заранее определенным числом (например, 10).

Результат показать в отдельном окне.

Контрольные вопросы

1. Что такое модель делегирования событий в Java и зачем она нужна?
2. В чем разница между внутренним классом и лямбда-выражением при написании обработчиков событий?
3. Какие наиболее используемые типы событий в Java-приложении с графическим интерфейсом?
4. Как можно обеспечить повторное использование обработчика событий несколькими компонентами?

Лабораторная работа №18. Разработка приложения с графическим интерфейсом

Цель: Освоить основные техники проектирования и разработки приложений с графическим интерфейсом пользователя (GUI) в Java, ознакомиться с возможностями библиотеки Swing и научиться создавать интерактивные формы, содержащие элементы управления и реагировать на взаимодействие пользователя.

Теоретическая часть.

Библиотека Swing в Java предоставляет мощные инструменты для разработки кросс-платформенных графических интерфейсов. Основными компонентами Swing являются контейнеры (такие как JFrame, JPanel), управляющие виджетами (кнопки JButton, поля ввода JTextField, выпадающие списки JComboBox и т.п.) и менеджеры размещения (FlowLayout, GridLayout, BorderLayout и другие), определяющие расположение элементов на форме.

Простой пример создания окна с надписью и кнопкой:

```
JFrame frame = new JFrame("Просто
окно");frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

JPanel panel = new JPanel();JLabel label = new JLabel("Добро
пожаловать!");JButton button = new JButton("Закрыть");

panel.add(label);panel.add(button);

frame.add(panel);frame.pack();frame.setVisible(true);
```

Для обработки действий пользователя (нажатий кнопок, выбора пунктов меню и т.д.) используют слушателей событий (listeners). Например, регистрация обработчика для кнопки:

```
button.addActionListener(new ActionListener() {    public void
actionPerformed(ActionEvent ae){        frame.dispose();    }});
```

Задание

Напишите приложение для планирования мероприятий. Приложение должно иметь главное окно с тремя радиокнопками («Концерт», «Спортивное мероприятие», «Выставка») и кнопкой «Подтвердить». После подтверждения выбранного мероприятия откроется диалоговое окно с информацией о выбранном мероприятии (название, описание и ориентировочная стоимость билета).

Контрольные вопросы

1. Что такое библиотека Swing и какую роль она играет в разработке графических интерфейсов Java?
2. В чем заключаются основные отличия менеджеров размещения FlowLayout и GridLayout?
3. Каким образом реализуется реакция на клик по кнопке в приложении с графическим интерфейсом?
4. В чем состоят основные этапы разработки приложения с графическим интерфейсом в Java?

Лабораторная работа №19. Разработка учебного проекта в Android Studio (начальный этап).

Цель: Освоить начальные навыки работы с Android Studio, научиться создавать проекты, редактировать макеты и запускать эмуляторы устройств для тестирования приложений.

Теоретическая часть.

Android Studio — официальная среда разработки приложений для платформы Android. Она предоставляет мощный редактор кода, систему контроля версий, поддержку Gradle для сборки проектов и встроенные инструменты для тестирования и отладки приложений.

Для начала разработки Android-приложения необходимо установить Android Studio, настроить виртуальную машину для запуска эмулятора и создать новый проект. Проект Android Studio имеет определенную структуру папок, включающую ресурсы (res), манифест (AndroidManifest.xml), классы активности (Activity.java) и XML-файлы разметки (activity_main.xml).

Важнейшие составляющие проекта:

- activity_main.xml: Макет экрана приложения, где располагаются кнопки, текстовые поля и другие элементы интерфейса.
- MainActivity.java: Основной класс приложения, управляющий поведением активности и реакцией на действия пользователя.

Пример простого HelloWorld-приложения:Открываем activity_main.xml и добавляем текстовую надпись TextView:

```
<TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Привет, мир!" />
```

Затем открываем MainActivity.java и устанавливаем контент активити:

```
protected void onCreate(Bundle savedInstanceState)
{super.onCreate(savedInstanceState);setContentView(R.layout.activity_main);}
```

Задание

Создайте простое приложение для Android, которое отображает приветствие на главном экране. Приложение должно включать текстовую надпись "Здравствуйте, пользователи!" и кнопку "О программе", при нажатии на которую появится всплывающее окно с коротким описанием вашего приложения.

Контрольные вопросы

1. Что такое Android Studio и для чего она предназначена?
2. Какие основные папки структуры проекта создаются при создании нового проекта в Android Studio?
3. В чем разница между layout файлами и Activity классами в Android?
4. Как запустить приложение на эмуляторе устройства в Android Studio?

Лабораторная работа №20. Модификация учебного проекта в Android Studio.

Цель: Освоить процессы модификации существующих Android-проектов, внесения изменений в макеты, адаптацию поведения приложения и проверку функциональности с помощью эмуляторов или реальных устройств.

Теоретическая часть.

Android Studio предоставляет широкие возможности для быстрого изменения и расширения функционала мобильных приложений. Одной из ключевых особенностей среды разработки является поддержка различных вариантов дизайна и адаптивных макетов. Мы рассмотрим, как вносить изменения в существующие макеты и управлять ими программно.

Макеты приложения хранятся в ресурсах res/layout/, где определяются элементы интерфейса. За поведение отвечают классы активности (Activity), расположенные в src/main/java/. Для обновления внешнего вида или логики достаточно внести изменения в соответствующие файлы.

Пример: изменение надписи на главной странице приложения: Откройте файл activity_main.xml и замените старый текст на новый:

<TextView
android:id="@+id/textView"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Новая версия моего приложения" />

Далее обновите поведение приложения, добавив реакцию на нажатие кнопки в классе MainActivity.java:

public class MainActivity extends AppCompatActivity {@Overrideprotected void onCreate(Bundle savedInstanceState) {super.onCreate(savedInstanceState);setContentView(R.layout.activity_main);Button btnClickMe = findViewById(R.id.button);btnClickMe.setOnClickListener(v -> Toast.makeText(this, "Вы нажали кнопку!", Toast.LENGTH_SHORT).show());}}

Задание

Ваш учебный проект представляет собой простое приложение с одной кнопкой и надписью. Ваша задача — изменить текст надписи на "Добро пожаловать в мой первый проект!", сделать кнопку неактивной по умолчанию и включить её при первом касании надписи пальцем..

Контрольные вопросы

1. Как называется основной файл ресурсов, в котором хранится информация о дизайне и макете главного экрана приложения?
2. Какими инструментами Android Studio можно воспользоваться для адаптации макета под разные размеры экранов?
3. Как задать реакцию на нажатие кнопки в Android-приложении?
4. Как сделать кнопку неактивной по умолчанию и активировать её при определенных условиях?

Лабораторная работа №21. Разработка БД и подключение ее к учебному

Цель: Освоить проектирование и создание простой реляционной базы данных, изучить основные SQL-запросы, подключиться к базе данных из Java-приложения и осуществить выполнение запросов.

Теоретическая часть.

Реляционные базы данных предоставляют структурированный способ хранения и извлечения данных. Одна из популярных СУБД — SQLite, которая подходит для небольших проектов благодаря своей легкости и встраиваемости прямо в приложение.

Для подключения к базе данных в Java используется драйвер JDBC (Java Database Connectivity). Перед началом работы необходимо подключить нужный драйвер к вашему проекту.

Пример подключения к базе данных и выборки данных с использованием SQLite:

```
Connection conn = DriverManager.getConnection("jdbc:sqlite:sample.db");
Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery("SELECT * FROM users;");
while(rs.next()) {
   System.out.println("Пользователь: " + rs.getString("name"));
```

```
}
rs.close();
stmt.close();
conn.close();
```

Для создания таблицы можно использовать следующий SQL-код:

```
CREATE TABLE users (
   id INTEGER PRIMARY KEY AUTOINCREMENT,
   name TEXT NOT NULL,
   email TEXT UNIQUE
);
```

Задание

Разработать базу данных SQLite с таблицей товаров (product_id, product_name, price). Подключитесь к этой базе данных из вашего Java-проекта и напишите простой запрос для вставки новых товаров и последующего их просмотра.

Контрольные вопросы

1. Что такое реляционная база данных и какие преимущества она предоставляет?
2. Какие этапы необходимы для подключения к базе данных SQLite из Java-приложения?
3. Каково назначение драйвера JDBC и какой синтаксис команды для подключения к базе данных?
4. Какие операторы SQL используют для создания таблиц, вставки данных и их выборки?

Лабораторная работа №22. Подключение контент-провайдера.

Цель: Освоить принцип работы с Content Providers в Android-приложениях, научиться получать доступ к данным сторонних приложений через API провайдеров и использовать их в своем проекте.

Теоретическая часть.

Контент-провайдеры (Content Providers) — это слой инфраструктуры Android, позволяющий приложениям обмениваться данными друг с другом безопасным и стандартизированным способом. Через контент-провайдер приложение может делиться своими данными с другими приложениями, использующими тот же URI-доступ.

Основной задачей при работе с контент-провайдерами является получение данных с использованием стандартного формата URIs, начиная с префикса content:// и заканчивая именем поставщика данных и конкретной сущностью (таблицей или объектом).

Пример обращения к контактам телефона через контент-провайдер:

```
Cursor cursor = getContentResolver().query(
   ContactsContract.Contacts.CONTENT_URI, null, null, null, null);
while(cursor.moveToNext()) {
   long contactID =
cursor.getLong(cursor.getColumnIndex(ContactsContract.Contacts._ID));
   Log.d("MyApp", "Contact ID: " + contactID);
}
cursor.close();
```

Здесь CONTENT_URI — уникальный адрес, используемый для доступа к данным контактов.

Задание

Создайте Android-приложение, которое обращается к встроенному контент-провайдеру CalendarProvider для получения информации о ближайших событиях календаря. Выведите список событий на экран приложения.

Контрольные вопросы

1. Что такое контент-провайдер в Android и для чего он нужен?
2. Каким образом определяется уникальный URI ресурса в контент-провайдере?
3. Какие разрешения требуются для доступа к контент-провайдерам сторонних приложений?
4. Как можно обратиться к данным, предоставляемым другим приложением через контент-провайдер?

Лабораторная работа №23. Включение диалога в учебный проект.

Цель: Освоить работу с диалоговыми окнами в Android-приложениях, изучить их типы и сценарии использования, а также научиться интегрировать диалоги в существующий проект.

Теоретическая часть.

Диалоги в Android представляют собой небольшие окошки поверх основного экрана, предназначенные для уведомления пользователя или запроса дополнительной информации. Диалоги бывают нескольких типов:

- AlertDialog — самый распространенный тип, показывающий предупреждения, запросы подтверждения или сообщения.
- ProgressDialog — показывает прогресс выполнения задачи.
- DatePickerDialog и TimePickerDialog — позволяют выбрать дату или время.

Диалоги обычно настраиваются с помощью специального билдера DialogBuilder, который помогает сконструировать внешний вид и поведение диалога.

Пример создания простого AlertDialog:

```
AlertDialog.Builder builder = new AlertDialog.Builder(MainActivity.this);
builder.setTitle("Внимание!")
    .setMessage("Хотите продолжить?")
    .setPositiveButton("Да", (dialog, which) -> {
       // Действие при выборе Да
    })
    .setNegativeButton("Нет", (dialog, which) -> dialog.dismiss())
    .create()
    .show();
```

Задание

Интегрируйте в ваш учебный проект диалоговое окно, которое отображается при долгом нажатии на кнопку. Диалог должен предложить пользователю подтвердить выход из приложения с выбором "Да" или "Нет".

Контрольные вопросы

1. Что такое диалог в Android и для чего он предназначен?
2. Какие основные типы диалогов существуют в Android и каковы их назначения?
3. Как вызвать диалоговое окно при определенном действии пользователя (например, долгом нажатии)?
4. Как передать результат из диалога в основную активность приложения?

Лабораторная работа №24. Создание в учебном проекте потока для выхода в интернет.

Цель: Изучить организацию асинхронных процессов в Android, научиться устанавливать соединение с интернет-ресурсами, передавать и получать данные, а также обрабатывать возможные ошибки и проблемы безопасности.

Теоретическая часть.

В Android многопоточное программирование крайне важно, поскольку UI-поток не допускает блокирующих операций, таких как длительные HTTP-запросы. Поэтому необходимо создавать отдельные фоновые потоки для выполнения ресурсоемких задач, таких как отправка запросов и получение данных из интернета.

Одним из распространенных подходов является использование AsyncTask для организации фонового процесса. AsyncTask упрощает многопоточность, предлагая удобную схему параллельного выполнения операций без сложного ручного управления потоками.

Пример отправки GET-запроса с использованием AsyncTask:

```
class DownloadWebPageTask extends AsyncTask<String, Void, String> {
    @Override
    protected String doInBackground(String... urls) {
        HttpURLConnection connection = null;
        try {
            URL url = new URL(urls[0]);
            connection = (HttpURLConnection)url.openConnection();
            connection.connect();
            BufferedReader in = new BufferedReader(new
InputStreamReader(connection.getInputStream()));
            StringBuffer response = new StringBuffer();
            String inputLine;
            while ((inputLine = in.readLine()) != null) {
                response.append(inputLine);
            }
            return response.toString();
        } finally {
            if (connection != null) connection.disconnect();
        }
    }

    @Override
    protected void onPostExecute(String result) {
        // здесь можно обновить UI или обработать результат
    }
}
new DownloadWebPageTask().execute("http://example.com");
```

Задание

Организуйте асинхронный запрос к сайту www.example.com и выведите на экран результат (HTML-код страницы) в вашем учебном проекте. При возникновении проблем соединения с сервером предусмотрите обработку ошибок и информирование пользователя.

Контрольные вопросы

1. Зачем в Android необходим отдельный поток для выполнения сетевых запросов?
2. Какие классы и методы используются для организации HTTP-запросов в Android?
3. Что такое AsyncTask и как он облегчает создание асинхронных процессов?

Лабораторная работа №25. Подготовка тестового плана и тестовых пакетов и плана для тестирования модулей и/или классов учебного проекта.

Цель: Освоить процесс подготовки тестового плана и формирования набора тестов для автоматизированного тестирования модуля или класса в рамках учебного проекта на Java. Научиться создавать модульные тесты с применением инструмента JUnit..

Теоретическая часть.

Автоматизированное тестирование позволяет убедиться в правильности работы отдельных частей системы (модулей, классов) и значительно снижает вероятность ошибок. Один из распространенных инструментов для написания модульных тестов в Java — JUnit.

JUnit предоставляет аннотации для пометки методов теста (@Test), инициализации и очистки контекста (@BeforeEach, @AfterEach) и ожидания специфичных условий выполнения (@AssertEquals, @Fail и другие).

Пример простого теста с использованием JUnit:

```
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.assertEquals;

public class CalculatorTest {
   private Calculator calculator = new Calculator();

   @Test
   public void testAdd() {
      assertEquals(5, calculator.add(2, 3));
   }
}
```

Тест проверяет правильность работы метода сложения в классе Calculator.

Задание

Подготовьте пакет тестов для проверки работоспособности класса Person из вашего учебного проекта. Класс Person должен поддерживать методы set/get для полей Name и Age. Проверьте корректность установки имен и возраста с положительными и граничными значениями.

Контрольные вопросы

1. Что такое модульное тестирование и для чего оно применяется?
2. Какие аннотации используются в JUnit для обозначения методов тестов и настройки окружения?
3. В чем разница между утверждением assertTrue и assertEquals в JUnit?
4. Как построить эффективный тестовый план для комплексного тестирования модуля или класса?

Лабораторная работа №26. Функциональное тестирование интерфейса пользователя учебного проекта.

Цель: Освоить методики функционального тестирования пользовательского интерфейса мобильного приложения, подготовиться к созданию тестовых сценариев и проведению тестирования с использованием автоматизации..

Теоретическая часть.

Функциональное тестирование направлено на проверку соответствия поведения

приложения заявленным требованиям и спецификациям. Особенно важны тесты пользовательского интерфейса, так как именно через него большинство пользователей взаимодействуют с приложением.
Инструмент для автоматического тестирования интерфейса Android — Espresso, входящий в состав Android Testing Support Library. Эта библиотека позволяет имитировать реальные действия пользователя, такие как нажатие кнопок, прокрутка, ввод текста и ожидание отклика.

Пример простого теста на Espresso:

```
@RunWith(AndroidJUnit4.class)
public class LoginScreenTest {
   @Rule
   public ActivityScenarioRule<LoginActivity> mActivityRule =
         new ActivityScenarioRule<>(LoginActivity.class);

   @Test
   public void testLoginSuccess() {
      // Найти EditText для ввода логина и ввести текст
      onView(withId(R.id.username)).perform(typeText("testUser"));
      // Найти EditText для ввода пароля и ввести пароль
      onView(withId(R.id.password)).perform(typeText("password"));
      // Найти кнопку "Войти" и нажать её
      onView(withId(R.id.login_button)).perform(click());
      // Ожидание успешного перехода на страницу профиля
      onView(withId(R.id.profile_screen)).check(matches(isDisplayed()));
   }
}
```

Задача теста — пройти процедуру авторизации и убедиться, что приложение переходит на страницу профиля.

Задание

Разработайте автоматический тест для вашего учебного проекта, который проверяет успешное прохождение процедуры регистрации пользователя (или аналогичного процесса, зависящего от вашей предметной области): заполнение обязательных полей и переход на страницу успеха после успешной регистрации.

Контрольные вопросы

1. Что такое функциональное тестирование и какова его цель?
2. Какие основные инструменты и библиотеки используются для функционального тестирования интерфейса Android-приложений?
3. Как строится типичный тестовый сценарий на Espresso?
4. Какие общие ошибки возникают при проведении функциональных тестов интерфейса и как их избежать?

Лабораторная работа №27. Структурное тестирование программного кода обработки событий интерфейса пользователя.

Цель: Овладеть техникой структурного тестирования программного кода, работающего с событиями интерфейса пользователя, научиться анализировать покрытие тестами и выявлять потенциальные ошибки в обработке событий.

Теоретическая часть.

Структурное тестирование сосредоточено на проверке полного покрытия ветвей исполнения кода. Основная идея заключается в том, чтобы протестировать каждую ветвь алгоритма хотя бы раз. Инструменты, такие как JaCoCo, позволяют измерять степень покрытия кода тестами.

Для событий интерфейса пользователя особенно важна проверка следующих аспектов:

- обработка нажатий кнопок;
- реагирование на жесты (долгое нажатие, свайпы);
- обновление состояний интерфейса в результате действий пользователя.

Пример простого структурного теста с покрытием ветки условного оператора:

```
@Test
public void testButtonClick() {
    Button button = new Button(context);
    button.setOnClickListener(view -> {
        if (view.isEnabled()) {
            // Код при условии enabled
        } else {
            // Код при условии disabled
        }
    });

    // Тестируем обе ветви условия
    button.performClick(); // вызовет исполнение кода при условии enabled
    button.setEnabled(false);
    button.performClick(); // вызовет исполнение кода при условии disabled
}
```

В данном примере тестируется кнопка, которая ведет себя по-разному в зависимости от своего состояния (enabled/disabled).

Задание

Структурное тестирование сосредоточено на проверке полного покрытия ветвей исполнения кода. Основная идея заключается в том, чтобы протестировать каждую ветвь алгоритма хотя бы раз. Инструменты, такие как JaCoCo, позволяют измерять степень покрытия кода тестами.

Для событий интерфейса пользователя особенно важна проверка следующих аспектов:

- обработка нажатий кнопок;
- реагирование на жесты (долгое нажатие, свайпы);
- обновление состояний интерфейса в результате действий пользователя.

Пример простого структурного теста с покрытием ветки условного оператора:

```
@Test
public void testButtonClick() {
    Button button = new Button(context);
    button.setOnClickListener(view -> {
        if (view.isEnabled()) {
            // Код при условии enabled
        } else {
            // Код при условии disabled
        }
    });

    // Тестируем обе ветви условия
    button.performClick(); // вызовет исполнение кода при условии enabled
    button.setEnabled(false);
    button.performClick(); // вызовет исполнение кода при условии disabled
}
```

В данном примере тестируется кнопка, которая ведет себя по-разному в зависимости от своего состояния (enabled/disabled).

Контрольные вопросы

1. Что такое структурное тестирование и какие цели оно преследует?
2. Какие подходы используются для измерения степени покрытия кода тестами?
3. Какие сложности возникают при тестировании обработки событий интерфейса пользователя и как их преодолеть?
4. Каков правильный подход к созданию эффективных структурных тестов для событий интерфейса?

Лабораторная работа №28. Генерация тестовых данных для тестирования модулей/классов обработки данных Формирование отчета о тестировании проекта.

Цель: Освоить процесс генерации тестовых данных для полноценного тестирования модулей и классов обработки данных в Java, научиться формировать отчет о результатах тестирования.

Теоретическая часть.

Качественное тестирование требует большого количества разнообразных данных для проверок. Иногда сложно вручную подготовить наборы данных, поэтому применяют генераторы случайных данных или специально подготовленные фикстуры. Для генерации тестовых данных часто используются библиотеки, такие как Faker, RandomBean и DataFaker.

Отчет о тестировании фиксирует пройденные тесты, обнаруженные дефекты и общую статистику тестирования. Такой отчет важен для понимания качества продукта и дальнейшего улучшения тестирования.

Пример генерации тестовых данных с использованием Faker:

```
Faker faker = new Faker();
for(int i = 0; i < 10; i++) {
   String userName = faker.name().fullName();
   String userEmail = faker.internet().emailAddress();
   System.out.println(userName + ": " + userEmail);
}
```

Пример формирования простого отчета о тестировании:

```
int passedTests = 10;
int failedTests = 2;
double successRate = (passedTests * 100.0) / (passedTests + failedTests);
System.out.println("Отчёт о тестировании:\nПрошло тестов: " + passedTests +
            "\nПровалилось тестов: " + failedTests +
            "\nУспешность (%): " + successRate);
```

Задание

Подготовьте генератор тестовых данных для модуля аутентификации вашего учебного проекта. Сгенерируйте набор из 100 пользователей с уникальными именами и почтовыми адресами. Выполните серию тестов на модуль аутентификации и сформируйте отчёт о тестировании, включающий общее количество выполненных тестов, количество успешных и неудачных попыток, процент успешности.

Контрольные вопросы

1. Что такое генерация тестовых данных и для чего она необходима?
2. Какие существуют способы автоматической генерации тестовых данных?
3. Как сформировать полноценный отчет о тестировании и какие показатели должны присутствовать в отчёте?
4. Какие факторы влияют на качество тестируемых данных и какое влияние оказывает неправильная генерация данных на результаты тестирования?

Основная литература:

1. Алдан, А. Введение в генерацию программного кода / А. Алдан. - 2-е изд., испр. - М. : Национальный Открытый Университет «ИНТУИТ», 2016. - 189 с. : схем. - Библиогр. в кн. ; То же [Электронный ресурс]. - URL: http://biblioclub.ru/index.php?page=book&id=428825

2. Соколов В.П. Кодирование в системах защиты информации [Электронный ресурс]: учебное пособие/ Соколов В.П., Тарасова Н.П.— Электрон. текстовые данные.— М.: Московский технический университет связи и информатики, 2016.— 94 с.— Режим доступа: http://www.iprbookshop.ru/61485.html.— ЭБС «IPRbooks»

3. Антимиров, В. М. Проектирование аппаратуры систем автоматического управления. В 2 ч. Ч. 1 : учебное пособие для СПО / В. М. Антимиров. — 2-е изд. — Саратов, Екатеринбург : Профобразование, Уральский федеральный университет, 2019. — 92 c. — ISBN 978-5-4488-04014, 978-5-7996-2834-5. — Текст : электронный // Электронно-библиотечная система IPR BOOKS : [сайт]. — URL: http://www.iprbookshop.ru/87852.html. — Режим доступа: для авторизир.
пользователей

Дополнительная литература:

1. Федорова Г.И. Разработка, внедрение и адаптация программного обеспечения отраслевой направленности. Учебное пособие. Изд.: КУРС, Инфра-М. Среднее профессиональное образование. 2016 г. 336 стр.