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

И.О. Директора
Пятигорского института
(филиал) СКФУ
В.А. Фурсов

ПМ.02 РАЗРАБОТКА И ИНТЕГРАЦИЯ МОДУЛЕЙ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ
02.01 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

Специальности СПО
09.02.11 Разработка и управление программным обеспечением

Пятигорск 2026 г.

Методические указания к выполнению лабораторных работ по дисциплине ПМ.02 Разработка и интеграция модулей программного обеспечения составлены в соответствии с требованиями ФГОС СПО. Предназначены для студентов, обучающихся по специальности 09.02.11 Разработка и управление программным обеспечением.

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

Дисциплина «Разработка программных модулей» направлена на формирование у студентов компетенций в области проектирования, разработки, тестирования и документирования программных модулей на всех этапах жизненного цикла программного обеспечения.

В результате освоения учебной дисциплины обучающийся должен:

знать:

основные этапы жизненного цикла программного обеспечения и модели разработки;

принципы алгоритмизации, структуры данных и базовые алгоритмические конструкции;

синтаксис и семантику языка программирования высокого уровня;

принципы работы систем контроля версий (Git);

основные концепции объектно-ориентированного программирования;

механизмы обработки исключительных ситуаций;

паттерны проектирования и их классификацию;

основы работы с файловой системой и потоками ввода-вывода;

принципы многопоточного и сетевого программирования;

основы разработки пользовательских интерфейсов и работы с базами данных;

методы оптимизации, профилирования и документирования кода.

уметь:

разрабатывать программные модули в соответствии с техническим заданием;

применять базовые алгоритмические конструкции и структуры данных;

использовать системы контроля версий для управления кодом;

реализовывать объектно-ориентированные концепции в коде;

обрабатывать исключительные ситуации и создавать пользовательские исключения;

применять паттерны проектирования для решения типовых задач;

выполнять чтение и запись данных в файлы;

создавать многопоточные и сетевые приложения;

разрабатывать пользовательские интерфейсы и подключаться к базам данных;

оптимизировать и профилировать код, создавать документацию.

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 03. Планировать и реализовывать собственное профессиональное и личностное развитие, предпринимательскую деятельность в профессиональной сфере, использовать знания по правовой и финансовой грамотности в различных жизненных ситуациях

ОК 04. Эффективно взаимодействовать и работать в коллективе и команде

ОК 05. Осуществлять устную и письменную коммуникацию на государственном языке Российской Федерации с учетом особенностей социального и культурного контекста

ОК 06. Проявлять гражданско-патриотическую позицию, демонстрировать осознанное поведение на основе традиционных российских духовно-нравственных ценностей, в том числе с учетом гармонизации межнациональных и межрелигиозных отношений, применять стандарты антикоррупционного поведения

ОК 07.Содействовать сохранению окружающей среды, ресурсосбережению, применять знания об изменении климата, принципы бережливого производства, эффективно действовать в чрезвычайных ситуациях

ОК 08.Использовать средства физической культуры для сохранения и укрепления здоровья в процессе профессиональной деятельности и поддержания необходимого уровня физической подготовленности

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

Профессиональными компетенциями:

ПК 2.1. Проектировать модули программного обеспечения

ПК 2.2. Разрабатывать модули программного обеспечения

ПК 2.3. Выполнять интеграцию модулей и компонентов программного обеспечения

ПК 2.4. Выполнять тестирование и отладку программного обеспечения

ПК 2.5. Осуществлять документирование программных модулей программного обеспечения

ОБЩИЕ МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ВЫПОЛНЕНИЮ ЛАБОРАТОРНЫХ РАБОТ

Варианты лабораторных работ по дисциплине «Безопасность программного обеспечения» 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 лабораторных работ.

Студент должен прийти на лабораторную работу подготовленным к выполнению лабораторной работы.

Каждый студент после проведения работы должен представить отчет о проделанной работе с анализом полученных результатов и выводом по работе.

Таблицы и рисунки следует выполнять с помощью чертежных инструментов (линейки, циркуля, и т.д.) карандашом с соблюдением ЕСКД.

Расчет следует проводить с точностью до двух значащих цифр.

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.).

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

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

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
Установка и настройка среды разработки. Создание первого проекта

Цель: освоить процесс установки и настройки интегрированной среды разработки (IDE), создать первый программный проект, изучить структуру проекта и базовые принципы компиляции/интерпретации кода.

Краткие теоретические сведения

Интегрированная среда разработки (IDE) — программное обеспечение, предоставляющее комплекс инструментов для разработки: редактор кода, компилятор/интерпретатор, отладчик, средства сборки.

Основные этапы создания проекта:

Выбор языка программирования и соответствующей IDE.

Установка необходимого компилятора или интерпретатора.

Создание нового проекта (выбор типа приложения: консольное, оконное, библиотека).

Написание исходного кода.

Компиляция/интерпретация и запуск.

Типовые элементы проекта:

исходные файлы (.c, .cpp, .py, .java);

заголовочные файлы (для C/C++);

конфигурационные файлы;

файлы ресурсов;

файлы сборки (Makefile, CMakeLists.txt, pom.xml).

Задание:

Выбрать язык программирования (C/C++, Python, Java, C#).

Установить и настроить среду разработки:

для C/C++: установить компилятор GCC/MinGW, IDE Code::Blocks или Visual Studio Code;

для Python: установить интерпретатор Python, IDE PyCharm Community или VS Code;

для Java: установить JDK, IDE IntelliJ IDEA Community или Eclipse;

для C#: установить .NET SDK, IDE Visual Studio Community.

Создать консольное приложение «Hello, World!».

Изучить структуру созданного проекта (какие файлы созданы, их назначение).

Модифицировать программу: запросить у пользователя имя и вывести персонализированное приветствие.

Научиться запускать программу в режиме отладки и без отладки.

Составить отчет, содержащий:

название и версию установленной IDE;

описание шагов создания проекта;

листинг программы с комментариями;

скриншот результата выполнения;

вывод о проделанной работе.

Контрольные вопросы:

Что такое IDE и для чего она нужна?

Чем компилятор отличается от интерпретатора?

Какие типы проектов могут быть созданы в выбранной IDE?

Что такое отладка (debugging) и как её запустить?

Какие расширения файлов используются для исходного кода в выбранном языке?

Лабораторная работа №2
Реализация основных структур данных на выбранном языке программирования

Цель: изучить основные структуры данных (массивы, списки, стеки, очереди) и реализовать их программно.

Краткие теоретические сведения

Массив — структура данных, хранящая фиксированное количество элементов одного типа, расположенных в памяти последовательно. Доступ к элементам осуществляется по индексу за O(1).

Динамический массив (список) — структура, автоматически изменяющая свой размер при добавлении/удалении элементов (например, std::vector в C++, list в Python, ArrayList в Java).

Стек — структура, работающая по принципу LIFO (Last In — First Out). Операции: push (добавление), pop (удаление), top (просмотр верхнего элемента).

Очередь — структура, работающая по принципу FIFO (First In — First Out). Операции: enqueue (добавление в конец), dequeue (удаление из начала).

Задание:

Реализовать следующие структуры данных (без использования встроенных контейнеров, кроме массивов):

статический массив целых чисел (сортировка, поиск min/max, сумма элементов);

динамический массив (список) с операциями добавления, удаления, вставки по индексу;

стек на основе массива (методы push, pop, peek, isEmpty, size);

очередь на основе массива (методы enqueue, dequeue, front, isEmpty, size).

Для каждой структуры написать демонстрационную программу, показывающую работу всех методов.

Реализовать задачу «Проверка сбалансированности скобок» с использованием стека.

Реализовать задачу «Обработка очереди заявок» с использованием очереди.

Сравнить эффективность (по времени выполнения) операций вставки и удаления для массива и списка.

Составить отчет с листингами всех реализаций, примерами работы и выводами.

Контрольные вопросы:

В чем отличие статического массива от динамического?

Какова временная сложность доступа к элементу в массиве? В связном списке?

Какие операции поддерживает стек? Очередь?

Где на практике применяется стек? Очередь?

Что произойдет при попытке добавить элемент в переполненный стек (на массиве фиксированного размера)?

Лабораторная работа №3

Написание программ с использованием условных операторов и циклов

Цель: освоить управляющие конструкции языка программирования: условные операторы и циклы различных типов.

Краткие теоретические сведения

Условный оператор if-else — выполняет блок кода в зависимости от истинности условия. Может быть вложенным.

Оператор выбора switch-case — позволяет выбрать один из нескольких вариантов выполнения на основе значения выражения (поддерживает целые типы, символы, строки в некоторых языках).

Цикл while — выполняет блок кода, пока условие истинно. Проверка условия перед каждой итерацией.

Цикл do-while — выполняет блок кода хотя бы один раз, затем проверяет условие.

Цикл for — используется для перебора диапазона значений (инициализация; условие; инкремент).

Задание:

Написать программу, определяющую тип треугольника по длинам трех сторон (равносторонний, равнобедренный, разносторонний, не существует).

Написать программу «Калькулятор» с использованием switch-case (операции +, -, *, /, ^).

Написать программу для вычисления факториала числа n с использованием цикла for.

Написать программу для вывода таблицы умножения от 1 до 10 с использованием вложенных циклов.

Написать программу для поиска всех простых чисел в диапазоне от 2 до N с использованием цикла while.

Написать программу, выводящую числа Фибоначчи до N-го элемента (с использованием цикла).

Для каждой программы составить блок-схему алгоритма.

Составить отчет с листингами программ, блок-схемами и результатами выполнения.

Контрольные вопросы:

В чем разница между if-else и switch-case?

Какие типы данных можно использовать в switch в выбранном языке?

В чем отличие цикла while от do-while?

Что произойдет, если условие цикла изначально ложно в while? В do-while?

Как прервать выполнение цикла досрочно? Как перейти к следующей итерации?

Лабораторная работа №4

Разработка функций для обработки данных, использование рекурсии

Цель: освоить механизмы создания и использования функций, изучить рекурсивный подход к решению задач.

Краткие теоретические сведения

Функция — именованный блок кода, выполняющий определенную задачу. Может принимать параметры и возвращать значение.

Сигнатура функции — имя функции, типы и количество параметров, тип возвращаемого значения.

Рекурсия — метод решения задачи, при котором функция вызывает саму себя. Обязательно наличие базового случая (условия выхода) для предотвращения бесконечной рекурсии.

Рекурсия vs итерация:

рекурсия проще для задач с естественной рекурсивной структурой (деревья, факториал, Ханойские башни);

итерация эффективнее по памяти (нет переполнения стека).

Задание:

Реализовать следующие функции (без использования встроенных функций языка, кроме математических):

вычисление факториала числа (итеративно и рекурсивно);

вычисление числа Фибоначчи (итеративно и рекурсивно);

вычисление наибольшего общего делителя (НОД) алгоритмом Евклида (рекурсивно);

возведение числа в степень (рекурсивно).

Написать программу, демонстрирующую работу всех функций.

Сравнить время выполнения итеративной и рекурсивной версий для разных входных данных.

Реализовать рекурсивный обход файловой системы (вывод всех файлов и папок).

Реализовать рекурсивное решение задачи «Ханойские башни» для n дисков.

Составить отчет с листингами функций, анализом рекурсии (глубина, количество вызовов) и выводами.

Контрольные вопросы:

Что такое рекурсия? Что такое базовый случай?

Что произойдет, если в рекурсивной функции отсутствует условие выхода?

Чем рекурсивная функция отличается от итеративной по использованию памяти?

Какие задачи удобнее решать рекурсивно?

Что такое переполнение стека (stack overflow) и как его избежать?

Лабораторная работа №5

Работа с локальным репозиторием Git: создание коммитов, просмотр истории

Цель: освоить базовые операции работы с системой контроля версий Git.

Краткие теоретические сведения

Git — распределенная система контроля версий, позволяющая отслеживать изменения в файлах и координировать работу нескольких разработчиков.

Основные понятия:

Репозиторий — хранилище всех версий файлов и истории изменений.

Коммит — сохранение текущего состояния файлов с описанием изменений.

Ветка — независимая линия разработки.

Индекс (staging area) — область, где собираются изменения перед коммитом.

Базовые команды:

git init — создание нового репозитория;

git status — просмотр состояния файлов;

git add <файл> — добавление файла в индекс;

git commit -m "сообщение" — создание коммита;

git log — просмотр истории коммитов;

git diff — просмотр изменений.

Задание:

Установить Git (если не установлен).

Создать локальный репозиторий в отдельной папке.

Создать файл main.py (или другой язык) с программой «Hello, World!».

Выполнить первый коммит.

Модифицировать файл: добавить функцию приветствия пользователя.

Просмотреть изменения через git diff.

Добавить изменения в индекс и выполнить второй коммит.

Просмотреть историю коммитов (git log, git log --oneline, git log --graph).

Создать файл .gitignore и добавить в него временные файлы.

Продемонстрировать игнорирование файлов.

Составить отчет с описанием всех выполненных команд и скриншотами результатов.

Контрольные вопросы:

Что такое система контроля версий? Для чего она нужна?

Чем отличается git add от git commit?

Как просмотреть историю коммитов в сокращенном виде?

Для чего нужен файл .gitignore?

Как отменить добавление файла в индекс (перед коммитом)?

Лабораторная работа №6

Работа с ветками в Git: создание, слияние, разрешение конфликтов

Цель: освоить работу с ветками в Git, научиться создавать, сливать ветки и разрешать конфликты.

Краткие теоретические сведения

Ветка — указатель на определенный коммит. Позволяет разрабатывать новые функции изолированно от основной линии разработки.

Команды для работы с ветками:

git branch — просмотр веток;

git branch <имя> — создание ветки;

git checkout <имя> — переключение на ветку;

git checkout -b <имя> — создание и переключение;

git merge <имя> — слияние ветки с текущей;

git branch -d <имя> — удаление ветки.

Конфликт (merge conflict) — ситуация, когда две ветки изменяют одну и ту же строку в файле по-разному. Требует ручного разрешения.

Задание:

Создать репозиторий и файл README.md с описанием проекта.

Создать ветку feature/add-function от main.

В ветке feature/add-function добавить новую функцию (например, вычисление суммы чисел).

Сделать коммит в ветке.

Вернуться в ветку main и добавить другую функцию (например, вычисление разности).

Сделать коммит в main.

Выполнить слияние ветки feature/add-function в main.

Создать искусственный конфликт:

в ветке main изменить строку в файле;

создать новую ветку feature/conflict от main до изменения;

в новой ветке изменить ту же строку иначе;

выполнить слияние и разрешить конфликт.

Просмотреть историю слияний через git log --graph --oneline.

Удалить использованные ветки.

Составить отчет со всеми выполненными командами и скриншотами.

Контрольные вопросы:

Зачем нужны ветки в Git?

Как создать новую ветку и переключиться на неё?

Что такое merge conflict и как он возникает?

Как разрешить конфликт слияния?

Что произойдет при слиянии ветки, которая уже была слита ранее?

Лабораторная работа №7

Создание классов и объектов, реализация инкапсуляции

Цель: освоить основные принципы объектно-ориентированного программирования (классы, объекты, инкапсуляция).

Краткие теоретические сведения

Класс — шаблон для создания объектов, определяющий их состояние (поля) и поведение (методы).

Объект — экземпляр класса, имеющий конкретные значения полей.

Инкапсуляция — механизм, скрывающий внутреннее состояние объекта и предоставляющий доступ к нему только через методы.

Модификаторы доступа:

private — доступен только внутри класса;

protected — доступен внутри класса и наследникам;

public — доступен из любого места.

Геттеры и сеттеры — методы для чтения и изменения приватных полей с возможностью валидации.

Задание:

Создать класс Person с приватными полями:

name (строка);

age (целое число);

email (строка).

Реализовать конструктор с параметрами.

Реализовать геттеры и сеттеры для всех полей (в сеттере возраста добавить проверку: возраст от 0 до 150).

Добавить метод displayInfo(), выводящий информацию о человеке.

Создать класс Student, наследующий Person, с дополнительным полем studentId (строка) и методом study().

В главной программе создать массив из 3-5 объектов Student, вывести их информацию.

Продемонстрировать попытку установить некорректный возраст (через сеттер) и обработать ошибку.

Составить отчет с листингами классов, демонстрацией работы и выводами.

Контрольные вопросы:

Что такое класс? Что такое объект?

Для чего нужны модификаторы доступа?

Что такое инкапсуляция и как она реализуется?

Зачем нужны геттеры и сеттеры?

Может ли приватное поле быть доступно из другого класса?

Лабораторная работа №8

Построение иерархии классов, использование наследования и полиморфизма

Цель: освоить механизмы наследования и полиморфизма в объектно-ориентированном программировании.

Краткие теоретические сведения

Наследование — механизм, позволяющий создавать новый класс на основе существующего, добавляя или переопределяя его функциональность.

Полиморфизм — способность объектов разных классов реагировать на одни и те же методы по-разному.

Виртуальные методы (в некоторых языках) — методы, которые могут быть переопределены в классах-наследниках.

Абстрактные методы — методы без реализации, которые обязательно должны быть переопределены в наследниках.

Задание:

Создать базовый класс Shape с:

полем color (строка);

конструктором;

виртуальным методом getArea() (возвращает 0);

методом displayInfo(), выводящим цвет и площадь.

Создать классы-наследники:

Rectangle (прямоугольник) с полями width, height;

Circle (круг) с полем radius;

Triangle (треугольник) с полями base, height.

В каждом наследнике переопределить метод getArea().

В главной программе создать массив из объектов разных типов (Shape* в C++, List<Shape> в C#, ArrayList в Java).

Продемонстрировать полиморфизм: вызвать getArea() для каждого объекта без проверки типа.

Добавить класс Square (квадрат), наследующий Rectangle.

Продемонстрировать многоуровневое наследование.

Составить отчет с UML-диаграммой иерархии классов, листингами и результатами работы.

Контрольные вопросы:

Что такое наследование? Приведите пример.

Что такое полиморфизм? Как он реализуется?

Чем виртуальный метод отличается от обычного?

Можно ли вызвать метод базового класса из наследника?

Что такое множественное наследование? Поддерживается ли оно в выбранном языке?

Лабораторная работа №9

Обработка встроенных исключений при работе с файлами и вводом

Цель: изучить механизм обработки исключительных ситуаций при работе с файлами и пользовательским вводом.

Краткие теоретические сведения

Исключение — событие, нарушающее нормальное выполнение программы (ошибка ввода-вывода, деление на ноль, выход за границы массива).

Конструкция try-catch-finally:

try — блок, где может возникнуть исключение;

catch — блок обработки исключения;

finally — блок, выполняемый всегда (закрытие ресурсов).

Типовые исключения:

FileNotFoundException — файл не найден;

IOException — ошибка ввода-вывода;

NumberFormatException — ошибка преобразования строки в число;

ArrayIndexOutOfBoundsException — выход за границы массива.

Задание:

Написать программу, запрашивающую у пользователя имя файла и выводящую его содержимое на экран.

Обработать следующие исключения:

файл не существует;

нет прав на чтение файла;

файл пуст.

Написать программу, запрашивающую у пользователя целое число и вычисляющую 100 / введенное число.

Обработать:

ввод нечислового значения;

деление на ноль.

Реализовать запись данных в файл с проверкой:

корректности пути;

доступности дискового пространства.

В блоке finally обеспечить закрытие всех открытых файловых потоков.

Составить отчет с листингами программ и примерами возникновения и обработки исключений.

Контрольные вопросы:

Что такое исключение? Приведите примеры.

Как работает конструкция try-catch-finally?

Чем отличается catch от finally?

Можно ли иметь несколько блоков catch?

Что произойдет, если исключение не обработано?

Лабораторная работа №10

Создание и использование пользовательских исключений

Цель: научиться создавать собственные классы исключений и использовать их в приложениях.

Краткие теоретические сведения

Пользовательские исключения создаются путем наследования от базового класса исключений (Exception, RuntimeException и т.д.).

Зачем нужны пользовательские исключения:

семантическая точность (отражают специфику приложения);

детализированная обработка ошибок;

передача дополнительной информации об ошибке.

Рекомендации:

имя класса исключения должно заканчиваться на Exception;

предоставлять несколько конструкторов (по умолчанию, с сообщением, с причиной);

наследовать от Exception (проверяемое) или RuntimeException (непроверяемое).

Задание:

Создать пользовательское исключение InvalidAgeException для ситуации, когда возраст не входит в допустимый диапазон.

Создать пользовательское исключение InsufficientFundsException для банковского приложения (недостаточно средств).

Создать класс BankAccount с полями balance и методами deposit(), withdraw().

В методе withdraw() выбрасывать InsufficientFundsException, если запрашиваемая сумма превышает баланс.

В главной программе обработать исключения с выводом информативных сообщений.

Создать класс User с полем age и методом setAge(), выбрасывающим InvalidAgeException.

Продемонстрировать цепочку исключений (exception chaining).

Составить отчет с листингами классов исключений, их использованием и выводами.

Контрольные вопросы:

Для чего нужны пользовательские исключения?

От какого класса следует наследовать пользовательское исключение?

Чем отличается проверяемое исключение от непроверяемого?

Как передать дополнительную информацию в исключении?

Что такое цепочка исключений (exception chaining)?

Лабораторная работа №11

Реализация абстрактных классов и интерфейсов

Цель: освоить использование абстрактных классов и интерфейсов для проектирования гибких архитектур.

Краткие теоретические сведения

Абстрактный класс — класс, который не может быть инстанциирован. Содержит абстрактные методы (без реализации) и/или реализованные методы.

Интерфейс — контракт, определяющий набор методов, которые класс-реализатор должен реализовать. Не содержит реализации методов (в классическом понимании).

Отличия абстрактного класса от интерфейса:

абстрактный класс может содержать поля и реализованные методы;

класс может реализовать несколько интерфейсов, но наследовать только один абстрактный класс;

интерфейс определяет поведение, абстрактный класс — общую сущность.

Задание:

Создать абстрактный класс Animal с:

полем name;

конструктором;

абстрактным методом makeSound();

реализованным методом sleep() (вывод "Zzz...").

Создать интерфейс Movable с методами move() и stop().

Создать интерфейс Eatable с методом eat().

Создать классы Dog, Cat, Bird, наследующие Animal и реализующие необходимые интерфейсы.

В каждом классе реализовать метод makeSound() (гав, мяу, чирик).

Продемонстрировать полиморфное поведение: создать массив Animal и вызвать makeSound() для каждого.

Продемонстрировать работу интерфейсов: создать список Movable и вызвать move().

Составить отчет с диаграммой классов, листингами и результатами.

Контрольные вопросы:

Что такое абстрактный класс? Чем он отличается от обычного?

Можно ли создать экземпляр абстрактного класса?

Что такое интерфейс? Чем он отличается от абстрактного класса?

Может ли класс реализовать несколько интерфейсов?

Когда следует использовать абстрактный класс, а когда интерфейс?

Лабораторная работа №12

Реализация паттернов Singleton и Factory Method

Цель: изучить порождающие паттерны проектирования Singleton и Factory Method, реализовать их программно.

Краткие теоретические сведения

Singleton — гарантирует, что класс имеет только один экземпляр, и предоставляет глобальную точку доступа к нему.

Реализация Singleton:

приватный конструктор;

статическое поле с единственным экземпляром;

статический метод для получения экземпляра.

Factory Method — определяет интерфейс для создания объекта, но позволяет подклассам решать, какой класс инстанциировать.

Задание:

Реализовать Singleton для класса Logger с методами log(message) и getLogs().

Продемонстрировать, что все обращения возвращают один и тот же экземпляр.

Реализовать потокобезопасную версию Singleton (double-checked locking или использование статической инициализации).

Реализовать паттерн Factory Method для создания транспортных средств:

абстрактный класс Transport с методом deliver();

классы Truck, Ship, Airplane, наследующие Transport;

абстрактный класс Logistics с фабричным методом createTransport();

конкретные фабрики RoadLogistics, SeaLogistics, AirLogistics.

В главной программе создать фабрику и использовать её для доставки.

Составить отчет с диаграммами классов, листингами и демонстрацией работы.

Контрольные вопросы:

Для чего нужен паттерн Singleton? Какие у него недостатки?

Как реализовать потокобезопасный Singleton?

Для чего нужен паттерн Factory Method?

Чем Factory Method отличается от простого конструктора?

В каких случаях следует использовать Factory Method?

Лабораторная работа №13

Применение паттернов Observer и Strategy

Цель: изучить поведенческие паттерны проектирования Observer и Strategy, реализовать их на практике.

Краткие теоретические сведения

Observer — определяет механизм подписки, позволяющий объектам следить за изменениями состояния другого объекта (издатель-подписчик).

Strategy — определяет семейство алгоритмов, инкапсулирует каждый из них и делает их взаимозаменяемыми.

Задание:

Реализовать паттерн Observer для системы уведомлений о погоде:

класс WeatherStation (издатель) с методами setTemperature(), registerObserver(), removeObserver(), notifyObservers();

интерфейс Observer с методом update(temperature);

классы PhoneDisplay, TVDisplay, WebDisplay, реализующие Observer.

Продемонстрировать, что при изменении температуры все подписчики получают уведомление.

Реализовать паттерн Strategy для системы оплаты:

интерфейс PaymentStrategy с методом pay(amount);

классы CreditCardPayment, PayPalPayment, CryptoPayment, реализующие PaymentStrategy;

класс ShoppingCart с методом setPaymentStrategy() и checkout().

Продемонстрировать смену стратегии оплаты во время выполнения.

Составить отчет с диаграммами классов, листингами и демонстрацией работы.

Контрольные вопросы:

Для чего нужен паттерн Observer? Где он применяется?

В чем разница между паттерном Observer и Publisher-Subscriber?

Для чего нужен паттерн Strategy?

Чем Strategy отличается от наследования?

Как связаны паттерны Strategy и Factory Method?

Лабораторная работа №14

Разработка приложения для чтения/записи данных в текстовый файл

Цель: освоить работу с файловой системой, чтение и запись текстовых файлов.

Краткие теоретические сведения

Поток (stream) — абстракция, представляющая последовательность данных. Может быть входным (чтение) или выходным (запись).

Режимы открытия файлов:

r — чтение;

w — запись (перезапись);

a — добавление в конец;

r+ — чтение и запись.

Классы для работы с файлами:

C++: ifstream, ofstream, fstream;

Python: open(), with open(...) as file;

Java: FileReader, FileWriter, BufferedReader, BufferedWriter;

C#: StreamReader, StreamWriter.

Задание:

Написать программу «Записная книжка», поддерживающую операции:

добавление новой записи (имя, телефон, email);

просмотр всех записей;

поиск записи по имени;

удаление записи по имени.

Данные хранить в текстовом файле (формат CSV или JSON).

При запуске программы загружать данные из файла, при завершении — сохранять.

Реализовать обработку исключений (файл не найден, повреждённый формат).

Добавить функцию экспорта всех записей в другой файл.

Добавить функцию импорта записей из другого файла (с проверкой дубликатов).

Составить отчет с листингом программы, примерами работы и выводами.

Контрольные вопросы:

Какие режимы открытия файлов существуют?

В чем разница между текстовым и бинарным файлом?

Что происходит с данными при перезаписи файла?

Как правильно закрыть файл после работы с ним?

Для чего используется конструкция with open() в Python (или using в C#)?

Лабораторная работа №15

Создание многопоточного приложения, решение проблемы синхронизации

Цель: изучить создание потоков, механизмы синхронизации и решение проблемы состояния гонки.

Краткие теоретические сведения

Поток (thread) — легковесный процесс, выполняющийся в рамках одного процесса и разделяющий его ресурсы.

Создание потоков:

C++11: std::thread;

Python: threading.Thread;

Java: Thread или Runnable;

C#: Task или Thread.

Проблемы многопоточности:

Состояние гонки (race condition) — результат зависит от порядка выполнения потоков.

Взаимоблокировка (deadlock) — два потока ожидают освобождения ресурсов друг другом.

Синхронизация:

Мьютекс (mutex) — обеспечивает исключительный доступ к ресурсу.

Семафор — ограничивает количество потоков, имеющих доступ.

Блокировки (lock) — автоматическое управление мьютексом.

Задание:

Написать программу, создающую 5 потоков, каждый из которых инкрементирует общий счётчик 100000 раз.

Продемонстрировать состояние гонки (неправильный результат).

Исправить проблему, используя мьютекс.

Реализовать программу «Производитель-Потребитель»:

общая очередь ограниченного размера;

поток-производитель добавляет числа в очередь;

поток-потребитель извлекает и обрабатывает их;

использовать условные переменные для уведомления.

Продемонстрировать взаимоблокировку (создать два потока, захватывающих два мьютекса в разном порядке).

Показать способ избежания deadlock (единый порядок захвата).

Составить отчет с листингами программ, анализом результатов и выводами.

Контрольные вопросы:

Что такое поток? Чем он отличается от процесса?

Что такое состояние гонки (race condition)?

Как мьютекс решает проблему состояния гонки?

Что такое взаимоблокировка (deadlock) и как её избежать?

Зачем нужны условные переменные (condition variables)?

Лабораторная работа №16

Разработка клиент-серверного приложения на основе TCP

Цель: изучить основы сетевого программирования, разработку TCP-клиента и TCP-сервера.

Краткие теоретические сведения

Модель OSI — 7 уровней (физический, канальный, сетевой, транспортный, сеансовый, представления, прикладной).

Стек TCP/IP — основа интернета, включает протоколы IP (маршрутизация), TCP (надёжная доставка), UDP (ненадёжная доставка).

Сокет (socket) — программный интерфейс для сетевого взаимодействия.

TCP-сервер (последовательный):

socket() — создание сокета;

bind() — привязка к порту;

listen() — ожидание подключений;

accept() — принятие подключения;

recv()/send() — обмен данными;

close() — закрытие сокета.

TCP-клиент:

socket() — создание сокета;

connect() — подключение к серверу;

send()/recv() — обмен данными;

close() — закрытие сокета.

Задание:

Разработать TCP-сервер «Чат»:

сервер принимает подключения от нескольких клиентов;

каждое сообщение от клиента пересылается всем остальным клиентам (broadcast);

сервер логирует все сообщения в файл.

Разработать TCP-клиента:

подключается к серверу;

отправляет сообщения;

в отдельном потоке принимает сообщения от сервера.

Реализовать команды клиента:

/nick <имя> — смена ника;

/quit — выход;

/help — справка.

Добавить обработку разрыва соединения.

Составить отчет с архитектурой приложения, листингами сервера и клиента, примерами работы.

Контрольные вопросы:

Чем отличается TCP от UDP?

Что такое сокет?

Для чего нужен bind()? Для чего listen()?

Как сервер обрабатывает несколько клиентов одновременно?

Какие порты считаются зарезервированными (0-1023)?

Лабораторная работа №17

Разработка приложения с графическим интерфейсом

Цель: освоить основы разработки графического пользовательского интерфейса (GUI) и событийного программирования.

Краткие теоретические сведения

Типы интерфейсов:

CLI (Command Line Interface) — командная строка;

GUI (Graphical User Interface) — графический;

NUI (Natural User Interface) — жестовый, голосовой;

TUI (Text User Interface) — псевдографический.

Событийное программирование — парадигма, при которой выполнение кода определяется наступлением событий (клик, ввод текста, таймер).

Основные GUI-фреймворки:

Python: Tkinter, PyQt, Kivy;

Java: Swing, JavaFX;

C#: Windows Forms, WPF;

C++: Qt, wxWidgets.

Задание:

Выбрать GUI-фреймворк в соответствии с языком программирования.

Разработать приложение «Калькулятор» с графическим интерфейсом:

кнопки цифр (0-9);

кнопки операций (+, -, *, /);

кнопка очистки (C);

кнопка вычисления (=);

поле для отображения ввода и результата.

Реализовать обработку событий для всех кнопок.

Добавить обработку нажатия клавиш с клавиатуры.

Добавить функцию сохранения истории вычислений в файл.

Оформить интерфейс с использованием стилей (темы, цвета).

Составить отчет с описанием элементов интерфейса, листингом программы и скриншотами.

Контрольные вопросы:

Что такое событийное программирование?

Какие типы пользовательских интерфейсов существуют?

Что такое callback-функция (обработчик события)?

В чем отличие GUI от CLI?

Как организован цикл обработки событий (event loop)?

Лабораторная работа №18

Написание приложения для работы с БД, выполнение SQL-запросов

Цель: изучить подключение к базам данных и выполнение SQL-запросов из приложения.

Краткие теоретические сведения

Реляционная база данных — данные организованы в виде таблиц со связями между ними.

SQL (Structured Query Language):
SELECT — выборка данных;
INSERT — вставка данных;
UPDATE — обновление данных;
DELETE — удаление данных.
Подключение к БД:
Python: sqlite3, psycopg2 (PostgreSQL), mysql-connector;
Java: JDBC;
C#: ADO.NET.

Задание:

Спроектировать базу данных для библиотеки:
таблица Authors (id, name, birth_year);
таблица Books (id, title, author_id, year, isbn);
таблица Readers (id, name, email);
таблица Loans (id, book_id, reader_id, loan_date, return_date).
Написать программу с консольным меню, поддерживающую:
добавление, редактирование, удаление книг;
добавление, редактирование, удаление читателей;
выдачу книги читателю;
возврат книги;
поиск книг по автору, названию, году;
вывод списка должников.
Все операции реализовать через SQL-запросы.
Использовать параметризованные запросы для защиты от SQL-инъекций.
Обработать исключения (нарушение внешнего ключа, дубликаты).
Составить отчет с ER-диаграммой, листингом программы и примерами работы.

Контрольные вопросы:

Что такое первичный и внешний ключи?
Для чего нужны параметризованные запросы?
Что такое SQL-инъекция и как от неё защититься?
Какие типы JOIN существуют в SQL?
Как обработать нарушение ссылочной целостности в приложении?

Лабораторная работа №19

Реализация CRUD-операций с использованием ORM

Цель: освоить использование ORM (Object-Relational Mapping) для работы с базами данных.

Краткие теоретические сведения

ORM — технология, связывающая объекты программы с таблицами базы данных.

Преимущества ORM:

работа с объектами, а не с SQL;

защита от SQL-инъекций;

кроссплатформенность (легкая смена СУБД);

автоматическое управление связями.

Популярные ORM:

Python: SQLAlchemy, Django ORM, Peewee;

Java: Hibernate, JPA;

C#: Entity Framework;

C++: ODB, Qt SQL.

Задание:

Выбрать ORM в соответствии с языком.

Создать модели (классы) для БД библиотеки из предыдущей работы.

Описать связи между моделями (один-ко-многим, многие-ко-многим).

Реализовать те же CRUD-операции, что и в ЛР №18, но с использованием ORM.

Продемонстрировать:

создание объектов и сохранение в БД;

выборку объектов с фильтрацией;

обновление объектов;

удаление объектов;

каскадное удаление.

Сравнить количество кода с версией на чистом SQL.

Составить отчет с описанием ORM, листингами моделей и операций.

Контрольные вопросы:

Что такое ORM и для чего он нужен?

Какие преимущества даёт использование ORM?

Какие недостатки есть у ORM?

Как в выбранном ORM описываются связи между таблицами?

Как ORM защищает от SQL-инъекций?

Лабораторная работа №20

Разработка простого REST-сервиса

Цель: освоить разработку REST API, реализовать основные эндпоинты.

Краткие теоретические сведения

REST (Representational State Transfer) — архитектурный стиль для построения веб-сервисов.

Принципы REST:

клиент-серверная архитектура;

отсутствие состояния (stateless);

единообразие интерфейса;

кэширование.

HTTP-методы в REST:

GET — получение ресурса;

POST — создание ресурса;

PUT — полное обновление;

PATCH — частичное обновление;

DELETE — удаление.

Коды состояния:

200 OK — успех;

201 Created — создано;

400 Bad Request — ошибка запроса;

401 Unauthorized — не авторизован;

403 Forbidden — запрещено;

404 Not Found — не найдено;

500 Internal Server Error — ошибка сервера.

Задание:

Выбрать фреймворк для создания REST API (Flask/Django REST для Python, Spring Boot для Java, Express для Node.js).

Создать REST-сервис для управления списком задач (To-Do list):

GET /tasks — получение всех задач;

GET /tasks/{id} — получение задачи по ID;

POST /tasks — создание задачи (в теле: title, description);

PUT /tasks/{id} — полное обновление задачи;

PATCH /tasks/{id} — частичное обновление (например, изменение статуса);

DELETE /tasks/{id} — удаление задачи.

Данные хранить в памяти (список/словарь) или в файле.

Добавить валидацию входных данных.

Добавить обработку ошибок (404 при отсутствии задачи).

Протестировать API с помощью Postman или cURL.

Составить отчет с описанием эндпоинтов, примерами запросов и ответов.

Контрольные вопросы:

Что такое REST API?

Какие методы HTTP используются в REST?

В чем разница между PUT и PATCH?

Какой код состояния возвращается при успешном создании ресурса?

Что такое stateless в контексте REST?

Лабораторная работа №21

Создание клиента для взаимодействия с REST API

Цель: разработать клиентское приложение для взаимодействия с REST API.

Краткие теоретические сведения

Клиент API — программа, отправляющая HTTP-запросы к API и обрабатывающая ответы.

Способы взаимодействия:

библиотека requests (Python);

HttpClient (Java, C#);

fetch/axios (JavaScript).

Форматы данных: JSON (чаще всего), XML, Protobuf.

Задание:

Использовать REST-сервис из предыдущей лабораторной работы (или другой учебный API).

Разработать клиентское приложение с консольным или графическим интерфейсом, поддерживающее:

просмотр всех задач;

просмотр задачи по ID;

добавление новой задачи;

редактирование задачи;

удаление задачи;

изменение статуса задачи.

Реализовать обработку ошибок API (неправильный ID, сервер недоступен).

Добавить вывод информации в удобном для чтения формате.

Реализовать автоматическую повторную отправку запроса при временной недоступности сервера.

Составить отчет с листингом клиента, примерами запросов и ответов.

Контрольные вопросы:

Какие библиотеки используются для HTTP-запросов в выбранном языке?

Как обработать ошибку соединения с сервером?

Что такое таймаут запроса и зачем он нужен?

Как в клиенте обрабатываются различные коды состояния HTTP?

Что такое маршаллинг и демаршаллинг JSON?

Лабораторная работа №22

Профилирование приложения, выявление узких мест и оптимизация кода

Цель: освоить инструменты профилирования, научиться выявлять узкие места и проводить оптимизацию кода.

Краткие теоретические сведения

Профилирование — измерение характеристик программы (время выполнения, использование памяти, частота вызовов функций).

Инструменты профилирования:

Python: cProfile, line_profiler, memory_profiler;

Java: VisualVM, JProfiler, YourKit;

C#: dotTrace, Visual Studio Diagnostic Tools;

C++: gprof, Valgrind, perf.

Этапы оптимизации:

Профилирование → выявление узкого места.

Анализ причины.

Оптимизация (алгоритмическая, структурная, микрооптимизации).

Повторное профилирование для проверки.

Задание:

Написать программу, выполняющую следующие операции:

поиск элемента в большом списке (100000+ элементов);

сортировка списка;

рекурсивный обход файловой системы;

частое создание объектов в цикле.

Замерить время выполнения каждой операции.

Использовать профилировщик для выявления функций, занимающих больше всего времени.

Оптимизировать:

заменить линейный поиск на бинарный (после сортировки);

заменить рекурсию на итерацию;

заменить создание объектов в цикле на переиспользование (object pooling).

Повторно замерить время выполнения и сравнить результаты.

Составить отчет с графиками (до/после), листингами оптимизированного кода и выводами.

Контрольные вопросы:

Что такое профилирование и зачем оно нужно?

Какие метрики можно измерить с помощью профилировщика?

Чем профилирование отличается от отладки?

Какие виды оптимизации кода существуют?

Почему преждевременная оптимизация считается вредной?

Лабораторная работа №23

Создание документации для разработанного модуля с помощью инструментов автоматической генерации

Цель: освоить стандарты оформления документации и инструменты для её автоматической генерации.

Краткие теоретические сведения

Виды документации:

API-документация (описание функций, классов, методов);

пользовательская документация (инструкции);

архитектурная документация;

техническое задание.

Стандарты оформления комментариев:

Javadoc (Java);

Doxygen (C++, C, C#, Java, Python);

Sphinx (Python);

JSDoc (JavaScript);

XML Documentation Comments (C#).

Инструменты генерации:

Doxygen — поддерживает множество языков;

Javadoc — встроен в JDK;

Sphinx + autodoc — для Python;

TypeDoc — для TypeScript.

Задание:

Выбрать один из предыдущих лабораторных проектов (например, REST-сервис или библиотеку).

Оформить комментарии в коде в соответствии со стандартом выбранного инструмента:

для всех классов;

для всех публичных методов;

для всех полей (если необходимо);

для модулей/файлов.

Настроить инструмент генерации документации.

Сгенерировать HTML-документацию.

Создать README.md для проекта, содержащее:

описание проекта;

инструкцию по установке и запуску;

примеры использования;

ссылку на сгенерированную документацию.

Составить отчет с описанием использованного стандарта, примерами комментариев и скриншотами сгенерированной документации.

Контрольные вопросы:

Какие виды документации существуют?

Для чего нужна автоматическая генерация документации?

Какие теги Javadoc/Doxygen вы знаете?

Чем хорош стандарт оформления комментариев в коде?

Как часто следует обновлять документацию при изменении кода?

Основная литература

Практическая подготовка обучающихся специальности 10.05.01 Компьютерная безопасность : методические указания / составители И. В. Влацкая, С. А. Герасименко. — Оренбург : ОГУ, 2024. — 70 с. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/503273 .

Череватова, Т. Ф. Нормативное обеспечение в сфере информационных технологий и систем : учебное пособие для СПО / Т. Ф. Череватова. — 2-е изд., стер. — Санкт-Петербург : Лань, 2024. — 84 с. — ISBN 978-5-507-47632-9. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/398513 .

Баланов, А. Н. Защита информационных систем. Кибербезопасность : учебное пособие для вузов / А. Н. Баланов. — 3-е изд., стер. — Санкт-Петербург : Лань, 2026. — 280 с. — ISBN 978-5-507-56255-8. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/514704 .

Дополнительная литература

Федорова Г.И. Разработка, внедрение и адаптация программного обеспечения отраслевой направленности. Учебное пособие. Изд.: КУРС, Инфра-М. Среднее профессиональное образование. 2022 г. 336 стр.