

Федеральное государственное образовательное бюджетное учреждение
высшего образования

**«ФИНАНСОВЫЙ УНИВЕРСИТЕТ ПРИ ПРАВИТЕЛЬСТВЕ
РОССИЙСКОЙ ФЕДЕРАЦИИ»
(Финансовый университет)**

**Кафедра информационных технологий
Факультет информационных технологий и анализа больших данных**

Документ подписан усиленной неквалифицированной электронной подписью
Организация: Финансовый университет при Правительстве РФ
Утверждено: Проректор по учебной и методической работе Е.А. Каменева
Сертификат: cYmpTsMZNDLZgQsOQy0zkhev0seTi2WM
Дата: 26.11.2025 г.

С.А. Румянцев

Современные технологии объектно-ориентированного программирования

Рабочая программа дисциплины

для студентов, обучающихся по направлению подготовки:

09.03.03 - Прикладная информатика,

Образовательная программа «Прикладные информационные системы в экономике
и финансах»

Рекомендовано

*Факультет информационных технологий и анализа больших данных
(протокол № 03 от 16.12.2025 г.)*

Одобрено

*Кафедра информационных технологий
(протокол № 12 от 03.12.2025 г.)*

© Москва 2026

СОДЕРЖАНИЕ

1.	Наименование дисциплины	3
2.	Перечень планируемых результатов освоения образовательной программы (перечень компетенций) с указанием индикаторов их достижения и планируемых результатов обучения по дисциплине	3
3.	Место дисциплины в структуре образовательной программы	6
4.	Объем дисциплины (модуля) в зачетных единицах и в академических часах с выделением объема аудиторной (лекции, семинары) и самостоятельной работы обучающихся	6
5.	Содержание дисциплины, структурированное по темам (разделам) дисциплины с указанием их объемов (в академических часах) и видов учебных занятий	7
5.1.	Содержание дисциплины	7
5.2.	Учебно-тематический план	11
5.3.	Содержание семинаров	12
6.	Перечень учебно-методического обеспечения для самостоятельной работы обучающихся по дисциплине	15
6.1.	Перечень вопросов, отводимых на самостоятельное освоение дисциплины, формы внеаудиторной самостоятельной работы	15
6.2.	Перечень вопросов, заданий, тем для подготовки к текущему контролю	20
7.	Фонд оценочных средств для проведения промежуточной аттестации обучающихся по дисциплине	23
8.	Перечень основной и дополнительной учебной литературы, необходимой для освоения дисциплины	47
9.	Перечень ресурсов информационно-телекоммуникационной сети «Интернет», необходимых для освоения дисциплины	48
10.	Методические указания для обучающихся по освоению дисциплины	49
11.	Перечень информационных технологий, используемых при осуществлении образовательного процесса по дисциплине, включая перечень необходимого программного обеспечения и информационных справочных систем	52
12.	Описание материально-технической базы, необходимой для осуществления образовательного процесса по дисциплине	52

1. Наименование дисциплины

«Современные технологии объектно-ориентированного программирования».

2. Перечень планируемых результатов освоения образовательной программы (перечень компетенций) с указанием индикаторов их достижения и планируемых результатов обучения по дисциплине

Код компетенции	Наименование компетенции	Индикаторы достижения компетенции	Результаты обучения (умения и знания), соотнесенные с индикаторами достижения компетенции
ПКН-1	Способность применять общенаучные, общетехнические знания, математические методы в сфере ИТ	Демонстрирует знания о современных естественнонаучных концепциях, общетехнических подходах, методах математического анализа и моделирования	знать: основные разделы математики, используемые в программировании и ИТ (элементы дискретной математики, линейной алгебры, математического анализа, теории вероятностей и математической статистики – на прикладном уровне); базовые понятия и методы математического моделирования (модель, параметр, допущение, калибровка, проверка адекватности модели); роль количественных оценок и формальных моделей при разработке и анализе программных и информационных систем. уметь: интерпретировать математические понятия применительно к задачам программирования (например, связывать структуры данных с объектами дискретной математики: графы, множества, отношения); использовать простые математические модели для описания поведения систем (например, линейные зависимости, вероятностные модели, агрегированные показатели эффективности); объяснять, какие исходные допущения и ограничения заложены в выбранную модель.
		Применяет знания для теоретического и экспериментального	знать: основные приемы оценивания и сравнения алгоритмов (асимптотическая сложность,

		исследования в сфере разработки программного обеспечения	порядки роста функций);базовые статистические методы (средние, дисперсия, корреляция, элементы анализа данных) для оценки характеристик программ и систем;принципы проведения вычислительных экспериментов (план эксперимента, сбор данных, анализ результатов). уметь: оценивать порядок сложности алгоритмов, применяемых в разрабатываемом программном обеспечении;использовать простые статистические и расчетные методы для анализа результатов работы программ (замеры времени, числа операций, объемов памяти);на основе полученных оценок делать выводы и предложения по оптимизации программных решений.
ПКН-2	Способность разрабатывать алгоритмы и программы с использованием современных технологий программирования	Владеет объектно-ориентированным языком программирования на уровне знания синтаксиса и семантики, основ стандартной библиотеки.	знать: синтаксис и семантику языка Java;основные встроенные типы данных, операции, управляющие конструкции;понятия класс, объект, интерфейс, пакет;базовые классы стандартной библиотеки Java (java.lang, java.util и др.). уметь: писать простые и составные программы на Java с использованием классов и методов;реализовывать основные конструкции ООП (инкапсуляция, наследование, полиморфизм) в Java;использовать базовые классы стандартной библиотеки (строки, коллекции, даты и т.п.).
		Использует инструментальные средства программирования (IDE, SDK, API, популярные фреймворки и библиотеки).	знать: назначение и основные функции IDE (IntelliJ IDEA / Eclipse и др.);принципы работы с JDK/SDK, сборки и запуска Java-приложений;общие принципы использования библиотек и фреймворков (подключение, документация, лицензии). уметь: настраивать и использовать IDE для разработки Java-проекта;подключать внешние

			библиотеки и фреймворки (через Maven/Gradle или вручную);использовать отладчик, просмотр стека вызовов, точки останова.
		Организовывает кодовую базу, ориентируется в существующем коде, демонстрирует знание общепринятых соглашений и политик в области оформления кода.	знать: основные соглашения по оформлению кода на Java (Java Code Conventions);принципы структурирования проекта (пакеты, модули, слои);понятия «читаемость кода», «рефакторинг», «code style». уметь: организовывать классы по пакетам в соответствии с логикой предметной области;читать и понимать чужой код на Java, выделять ключевые элементы;применять базовые приёмы рефакторинга (выделение методов, переименование, удаление дублирования).
		Проектирует текстовый, программный или графический интерфейс программной системы исходя из ее назначения.	знать: виды пользовательских интерфейсов (CLI, GUI, web-интерфейс) и их особенности;базовые принципы удобства и эргономики интерфейса;основные библиотеки/фреймворки для создания интерфейсов на Java (Swing, JavaFX и др. – на уровне обзора). уметь: проектировать структуру текстового меню/CLI для прикладной задачи;реализовывать простой интерфейс взаимодействия с пользователем на Java (консольный или графический);связывать интерфейс с объектной моделью приложения.

3. Место дисциплины в структуре образовательной программы

Дисциплина «Современные технологии объектно-ориентированного программирования» относится к «Общепрофессиональному циклу».

4. Объем дисциплины (модуля) в зачетных единицах и в академических часах с выделением объема аудиторной (лекции, семинары) и самостоятельной работы обучающихся

Очная форма обучения

Вид учебной работы по дисциплине	Всего (в з/е и часах)	Семестр 3 (в часах)
Общая трудоёмкость дисциплины	3/108	108
Контактная работа-Аудиторные занятия	50	50
Лекции	16	16
Семинары, практические занятия	34	34
Самостоятельная работа	58	58
Вид текущего контроля	Проектная работа	Проектная работа
Вид промежуточной аттестации	Экзамен	Экзамен

5. Содержание дисциплины, структурированное по темам (разделам) дисциплины с указанием их объемов (в академических часах) и видов учебных занятий

5.1. Содержание дисциплины

Тема 1. Введение в объектно-ориентированное программирование и Java

Парадигмы программирования: процедурная vs объектно-ориентированная. Основные принципы ООП: абстракция, инкапсуляция, наследование, полиморфизм. Обзор платформы Java: JVM, байт-код, JDK. Структура простой Java-программы. Синтаксис языка Java: типы данных, переменные, операторы, управляющие конструкции. Среда разработки (IDE) и инструменты сборки.

Тема 2. Классы и объекты в Java. Инкапсуляция

Определение классов, полей и методов. Создание объектов (оператор new). Конструкторы и перегрузка методов. Модификаторы доступа (public, private, protected) и организацию encapsulation. Пакеты и организационное разделение кода на модули. Примеры реализации классов для экономических задач (например, класс BankAccount). Практика проектирования классов: принцип единственной ответственности.

Тема 3. Наследование и полиморфизм

Иерархия классов, ключевое слово extends. Базовый и производный классы, переопределение методов (@Override). Виртуальные методы и позднее связывание (полиморфизм во время выполнения). Класс Object и методы toString(), equals() и др. Абстрактные классы и методы, интерфейсы (ключевое слово implements): различия и использование. Пример: модель «Сотрудник–Менеджер–Директор» с общим базовым классом. Интерфейсы Java API (Comparable, Serializable и др.). Использование полиморфизма для расширяемости программ.

Тема 4. Обработка исключений

Механизм исключений в Java: try-catch-finally, множество catch и иерархия исключений (Exception, RuntimeException, Error). Проверяемые и непроверяемые исключения. Оператор throws и проброс исключений. Создание собственных классов исключений (напр., InsufficientFundsException). Логирование и диагностика ошибок. Практические примеры: обработка ошибок ввода пользователя, работа с исключениями при вычислениях в финансах (деление на ноль, неверный формат числа).

Тема 5. Коллекции данных и обобщения (Generics)

Коллекции в Java (Java Collections Framework): списки (List, реализация ArrayList/LinkedList), множества (Set, HashSet/TreeSet), очереди, словари (Map, HashMap/TreeMap). Основные операции над коллекциями: добавление, удаление, поиск, сортировка (Comparator, Comparable). Итераторы. Обобщения (generic types): параметризация классов и методов типами, <> синтаксис. Ограничения и особенности дженериков (type erasure). Пример: хранение и обработка списка финансовых транзакций. Эффективность различных коллекций.

Тема 6. Ввод-вывод в Java. Потоки ввода-вывода

Байтовые и символьные потоки, классы InputStream/OutputStream и Reader/Writer. Чтение и запись текстовых файлов (FileReader/Writer, BufferedReader/Writer). Работа с файловой системой (класс File, Paths, Files). Обработка исключений при вводе-выводе (IOException). Кодировки символов. Пример: чтение CSV-файла с данными и формирование отчета. Кратко о NIO (New I/O) и Channels/Buffers.

Тема 7. Функциональное программирование в Java: лямбда-выражения и Stream API.

Понятие лямбда-выражения, синтаксис и области применения. Функциональные интерфейсы (Predicate, Function, Consumer и др.) и аннотация @FunctionalInterface. Использование лямбд для сортировки, фильтрации (Comparator на базе лямбда). Потоки (streams) в Java 8+: создание потока из коллекции, промежуточные операции (filter, map, sorted) и терминальные операции (forEach, collect, reduce). Понятие неизменяемости и отложенного выполнения в Stream API. Пример: фильтрация списка транзакций по критерию и агрегирование результатов.

Тема 8. Аннотации и рефлексия.

Аннотации в Java: встроенные (@Override, @Deprecated, @SuppressWarnings) и кастомные аннотации. Объявление собственной аннотации (@interface), элементы аннотации. Использование аннотаций в фреймворках (например, @Test в JUnit, аннотации в Spring для конфигурации бинов). Рефлексия: класс Class, получение информации о типе во время выполнения, использование java.lang.reflect (Method, Field, Constructor) для динамического вызова методов и создания объектов. Безопасность и производительность рефлексии.

Тема 9. Модульная система Java.

Концепция модулей (Java Platform Module System, JPMS) введена в Java 9: описание модуля (файл module-info.java), экспорт пакетов, требование модулей. Организация крупного приложения в виде модулей, инкапсуляция через модули. Преимущества модульности (лучшее управление зависимостями, уменьшение размера JRE через jlink). Пример: создание простого модуля и подключение библиотеки как модуля.

Тема 10. Современные технологии корпоративной разработки на Java: Java EE и Spring.

Обзор платформы Java EE (Jakarta EE): компоненты корпоративного приложения (серветы, JSP, EJB, JDBC, JMS и др.), трёхуровневая архитектура (клиент – сервер приложений – СУБД). Ограничения классической Java EE (сложность конфигурации, громоздкость приложений). Введение в Spring Framework – легковесный контейнер для создания enterprise-приложений. Принципы IoC (Inversion of Control) и DI (Dependency Injection) в Spring Spring Boot как современный подход к быстрой разработке: автоконфигурация, встроенный веб-сервер. Создание простейшего веб-приложения REST на Spring Boot (контроллер, сервис, репозиторий – в обзоре). Особенности Spring: аннотации (@Component, @Autowired, @Controller, @Service и т.д.), управление транзакциями, интеграция с БД (Spring Data). Значение Spring Framework в индустрии (де-факто стандарт enterprise-разработки на Java).

Тема 11. Шаблоны проектирования в объектно-ориентированных системах.

Понятие паттерна проектирования; обзор основных категорий GOF-паттернов: порождающие, структурные, поведенческие. Разбор примеров популярных шаблонов: Singleton (единственный экземпляр), Factory Method и Abstract Factory (создание объектов), Observer (наблюдатель), Strategy (стратегия), Decorator (декоратор) и др. Применение шаблонов на языке Java: реализация и типичные случаи использования. Влияние паттернов на архитектуру и поддерживаемость кода. Практический пример: использование паттерна Наблюдатель для оповещения об изменении финансовых показателей.

Тема 12. Сравнение ООП-языков: Java, C++, C#, Python.

Обзор реализации ООП-концепций в разных языках: классы и прототипы, статическая vs динамическая типизация, сборка мусора vs ручное управление памятью (C++), поддержка множественного наследования (в C++ и Python) vs интерфейсов (Java, C#). Управление памятью: сборщик мусора в Java/Python vs delete в C++. Особенности C#: сходство с Java (CLR vs JVM, LINQ, делегаты). Python: динамическое ООП, duck typing. Краткий анализ – сильные и слабые стороны Java по сравнению с другими (безопасность, производительность, простота разработки).

Тема 13. Юнит-тестирование и обеспечение качества кода.

Подходы к тестированию программного обеспечения: модульное (unit testing), интеграционное, системное тестирование. JUnit – фреймворк модульного тестирования для Java: написание тестовых методов с аннотациями @Test, Assertions для проверки результатов. Организация набора тестов, отчет о покрытии. Принципы TDD (разработка через тестирование). Использование логгеров (SLF4J, Log4j) для отслеживания выполнения. Инструменты анализа кода (Checkstyle, SonarQube – обзор). Практика: написание тестов для ранее реализованных классов (например, тестирование методов расчета финансовой формулы).

5.2. Учебно-тематический план

№ п/п	Наименование тем(разделов) дисциплины	Трудоемкость в часах				
		Всего	Контактная работа - Аудиторная работа			Самостоя тельная работа
			Общая, в т.ч.:	Лекции	Семинары, практическ ие занятия	
1	Введение в объектно-ориентированное программирование и Java	8	4	2	2	4
2	Классы и объекты в Java. Инкапсуляция	8	4	1	3	4
3	Наследование и полиморфизм	7	3	1	2	4
4	Обработка исключений	7	4	1	3	3
5	Коллекции данных и обобщения (Generics)	14	7	1	6	7
6	Ввод-вывод в Java. Потоки ввода-вывода	6	3	1	2	3
7	Функциональное программирование в Java: лямбда-выражения и Stream API.	7	3	1	2	4
8	Аннотации и рефлексия.	5	2	1	1	3
9	Модульная система Java.	5	1	1	0	4
10	Современные технологии корпоративной разработки на Java: Java EE и Spring.	16	8	2	6	8
11	Шаблоны проектирования в объектно-ориентированных системах.	13	6	2	4	7
12	Сравнение ООП-языков: Java, C++, C#, Python.	6	2	1	1	4
13	Юнит-тестирование и обеспечение качества кода.	6	3	1	2	3
	Итого	108	50	16	34	58

5.3. Содержание семинаров

Наименование тем (разделов) дисциплины	Перечень вопросов для обсуждения на семинарах, практических занятиях	Формы проведения занятий
Введение в объектно-ориентированное программирование и Java	В чем ключевые различия процедурного и объектно-ориентированного подходов? Что означают принципы ООП: абстракция, инкапсуляция, наследование, полиморфизм? Что такое JVM, JDK, JRE и как они связаны между собой? Что такое байт-код и как выполняется Java-приложение? В чем смысл принципа «Write Once, Run Anywhere» и за счет чего он реализуется?	Практические (семинарские) занятия
Классы и объекты в Java. Инкапсуляция	Чем отличаются класс и объект в Java? Что такое поля, методы, конструкторы; зачем нужна перегрузка? Как работают модификаторы доступа и какие задачи решает инкапсуляция? Зачем нужны пакеты и как они помогают организации кода? В чем смысл принципа единственной ответственности (SRP) при проектировании классов?	Практические (семинарские) занятия
Наследование и полиморфизм	Что такое наследование и как используется extends? Чем отличается перегрузка от переопределения методов? Что такое полиморфизм времени выполнения и позднее связывание? Абстрактные классы и интерфейсы: в чем различия и когда что применять? Какую роль играет класс Object и методы toString(), equals(), hashCode()?	Практические (семинарские) занятия
Обработка исключений	Что такое исключения и как устроена их иерархия в Java? Checked vs unchecked: в чем разница и как это влияет на проектирование? Как работают try-catch-finally, throw и throws? Зачем создавать пользовательские исключения и какие правила для них важны? Какую роль играют логирование и диагностика ошибок?	Практические (семинарские) занятия
Коллекции данных и обобщения (Generics)	Чем отличаются List, Set, Map и когда применять каждую структуру? ArrayList vs LinkedList: отличия по сложности операций и сценарии выбора. HashMap vs TreeMap, HashSet vs TreeSet: чем отличаются и что дают сортируемые структуры? Зачем нужны Comparable и Comparator? Для чего нужны generics и какие у них ограничения (type erasure)?	Практические (семинарские) занятия
Ввод-вывод в Java. Потоки ввода-вывода	Байтовые и символьные потоки: в чем различия и что выбрать? Зачем используются	Практические (семинарские) занятия

	буферизированные потоки (Buffered*)?Как работают File/Paths/Files и чем подход NIO отличается от классического I/O?Что такое кодировка и какие проблемы она вызывает при чтении/записи текстов?Как корректно обрабатывать IOException?	
Функциональное программирование в Java: лямбда-выражения и Stream API.	Что такое лямбда-выражение и функциональный интерфейс?Какие задачи решают Predicate, Function, Consumer и т. п. ?Как устроен конвейер Stream: промежуточные и терминальные операции?Что означает «ленивость» вычислений в Stream API?В чем особенности и риски parallelStream()?	Практические (семинарские) занятия
Аннотации и рефлексия.	Что такое аннотации и какие бывают Retention/Target?Как создать собственную аннотацию и где она используется?Что такое reflection и какие базовые операции она позволяет выполнять?Какие типовые сценарии применения рефлексии в Java-фреймворках?Каковы ограничения/риски рефлексии (производительность, безопасность, инкапсуляция)?	Практические (семинарские) занятия
Модульная система Java.	Зачем введена модульная система Java (JPMS)?Что описывает module-info. java (requires/exports/opens)?Что такое сильная инкапсуляция модулей и чем она отличается от пакетов?Какие проблемы возникают при миграции «старых» проектов на модули?Что такое automatic modules и когда они появляются?	Практические (семинарские) занятия
Современные технологии корпоративной разработки на Java: Java EE и Spring.	Чем концептуально отличаются подходы Java EE (Jakarta EE) и Spring?Что такое IoC и DI и почему они важны в корпоративной разработке?Какую роль играют основные стереотипы/аннотации Spring (@Component, @Service, @Controller, @Autowired)?Зачем нужен Spring Boot и что дает автоконфигурация?Как выглядит типовая слоистая архитектура приложения (controller/service/repository)?	Практические (семинарские) занятия
Шаблоны проектирования в объектно-ориентированных системах.	Что такое паттерн проектирования и чем он отличается от «архитектурного стиля»?Какие есть группы паттернов (порождающие/структурные/поведенческие)?В каких ситуациях уместны Singleton и Factory Method?В чем идея Observer и Strategy (задачи, которые они решают)?Как связаны принципы SOLID и применение паттернов?	Практические (семинарские) занятия
Сравнение ООП-языков: Java, C++, C#,	Чем отличаются модели исполнения и платформы JVM/. NET/нативная компиляция?Статическая и	Практические (семинарские) занятия

Python.	динамическая типизация: плюсы/минусы на практике. Как различается управление памятью (GC vs manual) и последствия для разработки?Как в языках реализуются наследование/интерфейсы/множественное наследование?Какие типовые области применения сильнее у Java по сравнению с C++/C#/Python?	
Юнит-тестирование и обеспечение качества кода.	Что такое модульное тестирование и какую проблему оно решает?Базовая структура теста в JUnit и роль assertions. Что такое TDD и когда оно оправдано?Что означает «покрытие кода тестами» и как его правильно интерпретировать?Какие практики повышают тестопригодность кода?	Практические (семинарские) занятия

6. Перечень учебно-методического обеспечения для самостоятельной работы обучающихся по дисциплине

6.1. Перечень вопросов, отводимых на самостоятельное освоение дисциплины, формы внеаудиторной самостоятельной работы

Наименование тем (разделов) дисциплины	Перечень вопросов, отводимых на самостоятельное освоение	Формы внеаудиторной самостоятельной работы
Введение в объектно-ориентированное программирование и Java	В чем различие процедурного и объектно-ориентированного подходов (по целям, структуре кода и сопровождению)? Что означает каждый из принципов ООП: инкапсуляция, наследование, полиморфизм, абстракция? Что такое JVM, JDK и JRE: назначение каждого компонента. Что такое байт-код и каков общий цикл компиляции/запуска Java-программы? Что означает принцип «Write Once, Run Anywhere» и какие ограничения у него есть? Какова роль IDE и инструментов сборки (Maven/Gradle) в жизненном цикле Java-проекта?	работа с рекомендуемыми источниками литературы; конспектирование; выполнение практической работы.
Классы и объекты в Java. Инкапсуляция	Что такое класс и объект в языке Java. Как они соотносятся с понятиями в ООП. Какие члены класса (поля, методы, блоки инициализации, внутренние классы) существуют в Java. Различия между методами экземпляра и статическими методами. Понятие инкапсуляции. Зачем она нужна при проектировании программных систем. Что такое модификаторы доступа (public, private, protected, default) и как они ограничивают доступ к членам класса. Как реализуется принцип единственной ответственности в проектировании классов. Назначение геттеров и сеттеров. Почему прямой доступ к полям нарушает инкапсуляцию. Автоматическая генерация методов toString(), equals() и hashCode() в IDE. Что означает перегрузка методов (method overloading) и как она реализуется в Java. Использование классов-оболочек (Integer, Double и др.) и их взаимодействие с объектами и примитивами. Отличие между this и ссылкой на объект. Случаи, где this необходим.	работа с рекомендуемыми источниками литературы; конспектирование; выполнение практической работы.
Наследование и полиморфизм	Понятие и назначение наследования в объектно-ориентированном программировании. Ключевое слово extends. Различие между абстрактным классом и интерфейсом. Когда использовать каждый из них. Механизм переопределения	работа с рекомендуемыми источниками литературы; конспектирование; выполнение

	методов. Использование аннотации @Override. Понятие полиморфизма. Виды полиморфизма в Java. Позднее связывание (dynamic dispatch) и его влияние на архитектуру приложений. Использование ключевых слов super и this в контексте иерархии классов. Ограничения множественного наследования в Java и способы их обхода с помощью интерфейсов. Иерархия классов в стандартной библиотеке Java (Object, Comparable, Serializable и др.).	практической работы.
Обработка исключений	Что такое исключительная ситуация (exception) в контексте исполнения программы. Иерархия классов исключений в Java: Throwable, Exception, RuntimeException, Error и их потомки. Различие между проверяемыми (checked) и непроверяемыми (unchecked) исключениями. Как работает блок try-catch-finally. Что произойдет, если исключение не перехвачено. Назначение ключевых слов throw и throws. Различие между ними. Порядок обработки нескольких catch-блоков. Как работает «вложенная» обработка. Что произойдет, если в блоке finally возникает новое исключение. Создание пользовательских исключений: правила наследования от Exception и реализация конструктора. Как логировать ошибки. Использование printStackTrace(), логгеров (Logger, SLF4J). Практическое применение обработки ошибок при чтении из файлов, делении на ноль и некорректном пользовательском вводе. Антипаттерны в обработке исключений: пустой catch, подавление исключений, избыточные throws.	работа с рекомендуемыми источниками литературы; конспектирование; выполнение практической работы.
Коллекции данных и обобщения (Generics)	Назначение и преимущества использования коллекций в Java по сравнению с массивами. Отличия между List, Set и Map: семантика хранения, порядок элементов, уникальность. Классы ArrayList, LinkedList, HashSet, TreeSet, HashMap, TreeMap: внутренняя структура, особенности и применение. Итераторы: интерфейс Iterator, метод remove(), цикл for-each, fail-fast поведение. Введение в обобщенные типы (Generics): зачем они были введены, что такое параметризация типов. Синтаксис generics: объявление обобщенных классов, интерфейсов и методов. Ограничения обобщений в Java: type erasure, невозможность создать массив обобщенного типа. Связь generics с безопасностью типов и ранним обнаружением ошибок компиляции. Wildcards: ?, ? extends T, ? super T –	работа с рекомендуемыми источниками литературы; конспектирование; выполнение практической работы.

	зачем нужны и как используются. Принцип PECS (Producer Extends, Consumer Super). Как generics применяются в стандартных структурах данных и API (Collections, Comparator).	
Ввод-вывод в Java. Потоки ввода-вывода	Виды потоков в Java: байтовые и символьные, их иерархия и основные классы. Отличия InputStream/OutputStream и Reader/Writer, сферы применения. Buffered-потоки: для чего используются BufferedReader и BufferedWriter. File I/O: классы File, FileReader, FileWriter, PrintWriter. Современные возможности ввода-вывода: java.nio.file.*, классы Files, Paths, Path. Работа с кодировками: зачем нужно явно указывать Charset, чем отличается UTF-8 от Windows-1251. Потокобезопасность при работе с файлами. Типичные ошибки при работе с файлами: неправильное закрытие ресурсов, потеря данных. Конструкция try-with-resources: назначение и преимущества. Исключения, возникающие при работе с файлами: FileNotFoundException, IOException.	работа с рекомендуемыми источниками литературы; конспектирование; выполнение практической работы.
Функциональное программирование в Java: лямбда-выражения и Stream API.	Что такое функциональное программирование и как его элементы реализованы в Java? Понятие и назначение функциональных интерфейсов (Predicate, Consumer, Function, Supplier). Особенности синтаксиса лямбда-выражений: параметры, тело, возвращаемые значения. Область видимости переменных внутри лямбда-выражений. Понятие stream-потока (Stream<T>), создание потока из коллекции. Промежуточные и терминальные операции в Stream API: map, filter, sorted, collect, reduce, forEach. Понятие ленивых вычислений и неизменяемости в контексте Stream API. Параллельные потоки (parallelStream()) и особенности их применения. Чем лямбда-выражения отличаются от анонимных классов? Каковы плюсы и ограничения использования функционального стиля в Java?	работа с рекомендуемыми источниками литературы; конспектирование; выполнение практической работы.
Аннотации и рефлексия.	Что такое аннотации в Java? В чем отличие встроенных и пользовательских аннотаций? Основные стандартные аннотации Java: @Override, @Deprecated, @SuppressWarnings, их применение и назначение. Синтаксис объявления собственной аннотации, типы элементов в аннотации. Какие ограничения существуют при создании аннотаций? Как применять аннотации к классам, методам, полям? Что такое аннотации с retention policy SOURCE, CLASS, RUNTIME?	работа с рекомендуемыми источниками литературы; конспектирование; выполнение практической работы.

	Механизм работы рефлексии (Reflection API): получение информации о классах, методах, полях во время выполнения. Как создать экземпляр класса с помощью рефлексии?Безопасность при использовании Reflection API. Какие риски существуют?Как Java-фреймворки используют аннотации и рефлексии (пример: JUnit, Spring)?	
Модульная система Java.	Что такое Java Platform Module System (JPMS) и зачем она была введена?Основные элементы модуля: module-info. java, директивы requires, exports, opens. Отличие между exports и opens. Как модульность влияет на инкапсуляцию и сборку проекта?Какова структура модульного проекта в Java?Как связаны модули и зависимые библиотеки?Как использовать jlink и jmod? Возможные ошибки в модульной системе (ModuleNotFoundException и др.) и их устранение. Как происходит разрешение зависимостей между модулями?Как применять JPMS в IDE (например, IntelliJ IDEA или Eclipse)?	работа с рекомендуемыми источниками литературы;конспектирование;выполнение практической работы.
Современные технологии корпоративной разработки на Java: Java EE и Spring.	Какие компоненты входят в архитектуру Java EE / Jakarta EE?Что такое сервлеты и их жизненный цикл?Отличия Java EE и Spring Framework: подход к внедрению зависимостей, конфигурации, масштабируемости. Что такое IoC (Inversion of Control) и DI (Dependency Injection) в контексте Spring?Аннотации Spring: @Component, @Service, @Autowired, @RestController — назначение и сфера применения. Что такое Spring Boot и чем он отличается от классического Spring?Как работает автоконфигурация в Spring Boot?Что такое слои (controller, service, repository) и зачем они нужны? Как создать REST API с помощью Spring Boot? Понятие конфигурации приложения: application. properties, application. ymlПодключение БД и работа с репозиториями через Spring Data JPA. Жизненный цикл Spring Bean'a и управление зависимостями.	работа с рекомендуемыми источниками литературы;конспектирование;выполнение практической работы.
Шаблоны проектирования в объектно-ориентированных системах.	Что такое шаблон (паттерн) проектирования? Зачем они нужны?На какие категории делятся паттерны: порождающие, структурные, поведенческие?В чем отличие паттернов Factory Method и Abstract Factory?Когда следует применять паттерн Singleton? Его плюсы и проблемы. Что делает паттерн Observer? Примеры использования в UI и финансах. Как реализовать паттерн Strategy? Зачем отделять алгоритм от клиента?Что делает паттерн Decorator и как он	работа с рекомендуемыми источниками литературы;конспектирование;выполнение практической работы.

	помогает избегать подклассов? В чем суть SOLID-принципов? Как они связаны с паттернами? Как сочетать шаблоны в одном проекте (например, Singleton + Factory)? Чем паттерны отличаются от архитектурных стилей (MVC, MVVM)? Как документировать и обосновывать применение паттерна?	
Сравнение ООП-языков: Java, C++, C#, Python.	Какие парадигмы реализуют Java, C++, C#, Python? В чем различие между статической и динамической типизацией? Как это влияет на проектирование? Как реализованы классы и объекты в C++, Java, Python и C#? Есть ли множественное наследование в этих языках? Как оно эмулируется? Что такое интерфейсы в Java и C#? Какую роль они играют? Как устроено управление памятью: сборка мусора против ручного контроля? Как реализованы конструкторы, деструкторы и финализаторы? Отличия в синтаксисе объявления классов и методов? Как выполняются программы: компиляция, байт-код, JIT-интерпретация? Что такое properties в C# и почему их нет в Java? Как реализуется перегрузка и переопределение методов? Поддержка функционального программирования: лямбды, замыкания? В каких областях используется каждый из языков? Где Java и где Python предпочтительнее? Что такое duck typing в Python? Как реализуются исключения, какие существуют типы ошибок?	работа с рекомендуемыми источниками литературы; конспектирование; выполнение практической работы.
Юнит-тестирование и обеспечение качества кода.	Что такое юнит-тестирование? Чем оно отличается от интеграционного и системного? Как организовать структуру тестов в Java с использованием JUnit 5? Что такое test case, test suite и test runner? Какие аннотации используются в JUnit 5 (@Test, @BeforeEach, @AfterEach, @Disabled)? Что такое TDD (Test Driven Development) и в чем его преимущества? Что такое mock-объекты и зачем они нужны в тестировании? Как измеряется покрытие кода тестами (code coverage)? Почему тесты помогают в рефакторинге? Что такое анти-паттерны в тестировании? Как логирование (Log4j, SLF4J) помогает выявлять ошибки? Различия между позитивными и негативными тестами? В чем разница между assertEquals, assertThrows и assertTimeout? Почему важно тестировать исключения? Как тестировать методы с внешними зависимостями (БД, файлы)? Что делать, если тесты "хрупкие" и часто падают?	работа с рекомендуемыми источниками литературы; конспектирование; выполнение практической работы.

6.2. Перечень вопросов, заданий, тем для подготовки к текущему контролю

Примерная тематика и задания проектной работы

1. Информационная система «Учебный портал»

- Реализовать классы: , ,

Student

Course

Enrollment

- Возможности: регистрация студентов, запись на курсы, выставление оценок, расчет среднего балла

- ООП: наследование (и), интерфейсы (,)

OnlineCourse

OnCampusCourse

Gradable

Printable

- Использование коллекций (Map, List)

- Отчет в формате CSV

- Тестирование с использованием JUnit

2. Финансовый менеджер (Personal Finance Tracker)

- Классы: , ,

Transaction

Category

Wallet

- Функции: добавление расходов/доходов, сортировка, фильтрация, баланс по категориям

- ООП: полиморфизм (/), шаблон

ExpenseTransaction

IncomeTransaction

Factory

- Сериализация данных в JSON
- UI через консоль или JavaFX (по желанию)

3. Микросистема бронирования ресурсов

- Классы: , ,

Room

Booking

User

- Возможности: создание бронирования, отмена, просмотр расписания
- Использование шаблона : уведомление о подтверждении/отмене

Observer

- Тестирование всех методов бронирования
- Документирование проекта: UML-диаграммы, README

4. Консольная игра «Угадай слово» с графическим отчетом

- Разработка на основе ООП-архитектуры: , ,

Game

Player

WordGenerator

- Статистика побед/поражений сохраняется в файл
- Использование Enum, Stream API
- Внедрение шаблона : разные уровни сложности

Strategy

5. REST API-сервер (Spring Boot, опционально)

- Темы: REST-интерфейс к системе учета задач (ToDo)
- Применение Spring: , ,

@RestController

@Service

@Repository

- Разделение на слои, применение IoC и DI
- Уровень сложности выше среднего

Критерии балльной оценки различных форм текущего контроля успеваемости содержатся в соответствующих методических рекомендациях Кафедры информационных технологий Факультета информационных технологий и анализа больших данных.

7. Фонд оценочных средств для проведения промежуточной аттестации обучающихся по дисциплине

Перечень компетенций с указанием индикаторов их достижения в процессе освоения образовательной программы содержится в разделе 2. *Перечень планируемых результатов освоения образовательной программы (перечень компетенций) с указанием индикаторов их достижения и планируемых результатов обучения по дисциплине.*

Типовые контрольные задания или иные материалы, необходимые для оценки индикаторов достижения компетенций, умений и знаний

ПКН-1 Способность применять общенаучные, общинженерные знания, математические методы в сфере ИТ

1) Демонстрирует знания о современных естественнонаучных концепциях, общинженерных подходах, методах математического анализа и моделирования

Результаты обучения (умения и знания), соотнесенные с индикаторами достижения компетенции

Знать: основные разделы математики, используемые в программировании и ИТ (элементы дискретной математики, линейной алгебры, математического анализа, теории вероятностей и математической статистики – на прикладном уровне); базовые понятия и методы математического моделирования (модель, параметр, допущение, калибровка, проверка адекватности модели); роль количественных оценок и формальных моделей при разработке и анализе программных и информационных систем.

Уметь: интерпретировать математические понятия применительно к задачам программирования (например, связывать структуры данных с объектами дискретной математики: графы, множества, отношения); использовать простые математические модели для описания поведения систем (например, линейные зависимости, вероятностные модели, агрегированные показатели эффективности); объяснять, какие исходные допущения и ограничения заложены в выбранную модель.

Типовые контрольные задания

1. Тестовые вопросы на знание базовых математических и естественнонаучных понятий

Вопрос 1. Какой из перечисленных объектов корректнее всего моделировать как граф?

- a) Изменение температуры в течение суток
- b) Структуру подчиненности сотрудников в компании
- c) Список курсов валют за неделю

d) Таблицу начисления процентов по вкладу

Вопрос 2. Какой из приведенных ниже графиков лучше всего описывает зависимость времени выполнения алгоритма сортировки от числа элементов n при использовании алгоритма с квадратичной сложностью?

a) Линейная зависимость $T(n) = a \cdot n + b$

b) Квадратичная зависимость $T(n) = a \cdot n^2 + b$

c) Логарифмическая зависимость $T(n) = a \cdot \log n + b$

d) Экспоненциальная зависимость $T(n) = a \cdot 2^n$

Вопрос 3. Какое утверждение о математической модели верно?

a) Модель всегда полностью совпадает с реальным объектом

b) Модель предназначена для упрощенного описания объекта с учетом существенных характеристик

c) Модель должна включать как можно больше деталей, иначе она бесполезна

d) Математическая модель не может использоваться для прогнозирования

Вопрос 4. Какой тип величины следует использовать для представления курса валюты в финансовом приложении?

a) Целое число (int)

b) Булева переменная (boolean)

c) Вещественное число с плавающей точкой (double/BigDecimal)

d) Строка (String), так как курс – это текст

Вопрос 5. Какое из утверждений о зависимости «спрос–цена» в простейшей экономической модели наиболее типично?

a) Спрос растет линейно с ростом цены

b) Спрос не зависит от цены

c) Спрос обратно пропорционален цене (при прочих равных условиях)

d) Спрос всегда экспоненциально растет с ценой

2. Задания по моделированию по словесному описанию (выделить ключевые величины и предложить простую математическую модель)

Задание 1. «Магазин бытовой техники»

В интернет-магазине бытовой техники ежедневно фиксируется количество заказов и средний чек за день. Руководство хочет приблизительно оценивать дневную выручку и прогнозировать ее при изменении среднего чека.

Выделите ключевые величины предметной области (минимум 3).

Предложите простую математическую модель для оценки дневной выручки (например, в виде формулы).

Объясните, какие допущения вы делаете при построении модели.

Ожидаемые элементы ответа:

Переменные:

N – количество заказов в день;

C – средний чек (средняя сумма одного заказа);

R – общая дневная выручка.

Модель: $R = N \cdot C$.

Допущения: все заказы примерно одинаковы по сумме; не учитываются скидки, возвраты, налоги и т.п.

Задание 2. «Время обработки запросов»

Система обрабатывает входящие запросы. Известно, что в среднем за час поступает λ запросов, а среднее время обработки одного запроса — t секунд.

Определите, какие величины являются входными параметрами, а какие — результатом.

Запишите формулу для оценки средней загрузки системы (доли времени, когда система занята обработкой запросов) в течение часа.

Обоснуйте, при каких значениях параметров система будет перегружена.

Ожидаемые элементы ответа:

Входные параметры: λ (запросов в час), t (секунд на запрос).

Количество секунд в часе: $T = 3600$.

Среднее суммарное время обработки: $T_{\text{busy}} = \lambda \cdot t$.

Доля загрузки: $\rho = T_{\text{busy}} / T$.

Перегрузка при ρ близком к 1 или $\rho > 1$.

Задание 3. «Простая модель роста числа пользователей»

Мобильное приложение каждый месяц привлекает новых пользователей. Предположим, что в среднем каждый месяц число пользователей увеличивается на k процентов.

Обозначьте необходимые величины и запишите формулу для числа пользователей через n месяцев.

Как изменится ваша модель, если учесть, что часть пользователей каждый месяц перестает пользоваться приложением (отток p процентов)?

Ожидаемые элементы ответа:

U_0 – начальное число пользователей;

k – процент роста в месяц;

$$U_n = U_0 \cdot (1 + k/100)^n.$$

С учетом оттока p : эффективный коэффициент роста $(1 + k/100 - p/100)$,

модель: $U_n = U_0 \cdot (1 + (k - p)/100)^n.$

3. Конкретные устные/письменные вопросы для обсуждения модели

(вместо «устный/письменный вопрос: объяснить, почему та или иная модель подходит/не подходит...»)

Вопрос 1.

В проекте используется модель времени выполнения программы:

$$T(n) = a \cdot n^2 + b, \text{ где } n \text{ — количество обрабатываемых записей.}$$

Объясните:

В каких ситуациях такая модель может быть разумным приближением?

Когда она перестанет адекватно описывать реальное поведение системы (приведите не менее двух причин)?

Вопрос 2.

Для оценки выручки интернет-магазина использована модель $R = N \cdot C$, где N — число заказов, C — средний чек.

Почему эта модель полезна на этапе проектирования информационной системы?

Какие реальные факторы она игнорирует?

Как бы вы скорректировали модель, чтобы учесть хотя бы один из факторов (например, скидки или возвраты)?

Вопрос 3.

Рассматривается модель нагрузки на сервер: $\rho = \lambda / \mu$, где λ — средняя интенсивность поступления запросов (запросов/сек), μ — средняя интенсивность обслуживания (запросов/сек).

Почему при $\rho > 1$ такая модель указывает на невозможность устойчивой работы системы?

Какие реальные механизмы (очереди, балансировка нагрузки, масштабирование) в этой модели не отражены?

Приведите пример ситуации, когда формально $\rho < 1$, но система все равно работает неудовлетворительно.

2) Применяет знания для теоретического и экспериментального исследования в сфере разработки программного обеспечения

Результаты обучения (умения и знания), соотнесенные с индикаторами достижения компетенции

Знать: основные приемы оценивания и сравнения алгоритмов (асимптотическая сложность, порядки роста функций); базовые статистические методы (средние, дисперсия, корреляция, элементы анализа данных) для оценки характеристик программ и систем; принципы проведения вычислительных экспериментов (план эксперимента, сбор данных, анализ результатов).

Уметь: оценивать порядок сложности алгоритмов, применяемых в разрабатываемом программном обеспечении; использовать простые статистические и расчетные методы для анализа результатов работы программ (замеры времени, числа операций, объемов памяти); на основе полученных оценок делать выводы и предложения по оптимизации программных решений.

Типовые контрольные задания

1. Задачи на оценку сложности и сравнение вариантов

Задача 1. «Поиск максимума»

Даны три реализации поиска максимального элемента в массиве целых чисел длины n :

1. Вариант А — простой линейный поиск (один проход по массиву).
2. Вариант В — сортировка массива методом пузырька, затем взятие последнего элемента.
3. Вариант С — сортировка массива методом быстрой сортировки (QuickSort), затем взятие последнего элемента.

Требуется:

1. Для каждого варианта словесно описать алгоритм.

2. Оценить асимптотическую сложность по времени (O-нотация) каждого варианта.
3. Сравнить варианты по порядку роста сложности и указать, какой из них предпочтителен при больших n и почему.

Задача 2. «Линейный и бинарный поиск»

Рассматриваются два алгоритма поиска элемента в отсортированном по возрастанию массиве длиной n :

- Алгоритм 1 — линейный поиск (последовательный просмотр элементов).
- Алгоритм 2 — бинарный поиск.

Требуется:

1. Записать асимптотическую сложность каждого алгоритма (по времени).
2. Для $n = 100$, $n = 10\,000$ и $n = 1\,000\,000$ оценить (оценочно, без кода), сколько сравнений в среднем потребуется каждому алгоритму.
3. Сделать вывод, при каких размерах n переход от линейного к бинарному поиску дает существенный выигрыш.

Задача 3. «Подсчет сумм по дням»

Имеется список из n операций в интернет-магазине. Каждая операция содержит дату и сумму. Требуется для каждого дня посчитать общую сумму продаж.

Предложены два варианта решения:

- Вариант 1: для каждого дня проходить по всему списку и суммировать все операции этого дня (двойной цикл по дням и операциям).
- Вариант 2: один раз пройти по списку операций, накапливая суммы в структуре `Map<Date, BigDecimal>`.

Требуется:

1. Описать псевдокод для обоих вариантов.
2. Оценить асимптотическую сложность по времени каждого варианта.
3. Сравнить варианты и указать, какой из них эффективнее при большом количестве операций и дней, обосновать ответ.

2. Задания на проведение вычислительного эксперимента

Задание 1. «Эксперимент: линейный vs бинарный поиск»

Необходимо экспериментально сравнить время работы линейного и бинарного поиска.

1. Реализовать на Java два метода:
2. Для размеров массива $n = 10\,000, 50\,000, 100\,000, 500\,000$:
3. Зафиксировать результаты в таблице вида:

n (размер массива)	Время линейного поиска (мс)	Время бинарного поиска (мс)
10 000		
50 000		
100 000		
500 000		

1. Сделать краткий вывод (5–7 предложений) о соотношении производительности алгоритмов и сопоставить результаты с теоретическими оценками сложности.

Задание 2. «Эксперимент: разные алгоритмы сортировки»

Требуется сравнить время работы двух алгоритмов сортировки массива целых чисел:

- реализация сортировки вставками (Insertion Sort);
- использование стандартного метода `Arrays.sort()`.

1. Реализовать на Java метод `insertionSort(int[] a)`.
2. Для размеров массива $n = 1\,000, 5\,000, 10\,000, 50\,000$:
3. Составить таблицу с результатами измерений и построить (при необходимости) график зависимости времени от n .
4. Сопоставить результаты с теоретическими оценками сложности: $O(n^2)$ для Insertion Sort и $O(n \log n)$ для `Arrays.sort()`.

Задание 3. «Эксперимент: выбор структуры данных»

Рассматриваются два подхода к учету посещений страницы сайта:

- Подход А: хранить посещения в `List<String>` (список логинов пользователей) и для подсчета количества посещений пользователя каждый раз проходить по всему списку.
- Подход В: при поступлении события увеличивать счетчик в `Map<String, Integer>`, где ключ — логин пользователя.

1. Реализовать оба подхода на Java.
2. Сгенерировать $N = 100\,000$ событий посещений с ограниченным числом различных пользователей (например, 1000 уникальных логинов).
3. Измерить время, необходимое для получения количества посещений каждого пользователя:

4. Записать результаты и сделать вывод, как выбор структуры данных влияет на производительность.

3. Вопросы по улучшению эффективности алгоритмов

Вопрос 1. «Оптимизация вложенных циклов»

Дан фрагмент кода:

```
for (int i = 0; i < users.size(); i++) {  
  
    User u = users.get(i);  
  
    int count = 0;  
  
    for (int j = 0; j < orders.size(); j++) {  
  
        if (orders.get(j).getUserId().equals(u.getId())) {  
  
            count++;  
  
        }  
  
    }  
  
    System.out.println(u.getName() + ": " + count);  
  
}
```

1. Оцените асимптотическую сложность данного фрагмента по времени при $n = \text{users.size()}$, $m = \text{orders.size()}$.
2. Предложите способ улучшить производительность, изменив структуру данных и/или алгоритм.
3. Обоснуйте, как изменится оценка сложности после оптимизации.

Вопрос 2. «Оптимизация работы с коллекциями»

В программе для проверки наличия элемента в наборе строк используется следующий код:

```
boolean exists(String x, List<String> list) {  
  
    for (String s : list) {  
  
        if (s.equals(x)) {  
  
            return true;  
  
        }  
  
    }  
  
}
```

```
}  
  
return false;  
  
}
```

1. Оцените сложность данного алгоритма по времени в зависимости от размера списка n .
2. Предложите альтернативное решение с использованием другой коллекции Java, позволяющее улучшить среднюю сложность проверки принадлежности.
3. Сравните теоретические оценки сложности двух вариантов и поясните, почему новый вариант эффективнее на больших данных.

Вопрос 3. «Выбор алгоритма сортировки»

В проекте для сортировки коллекции объектов используется простая реализация сортировки пузырьком. Объем данных вырос, и время работы стало неприемлемым.

1. Как вы оцените сложность сортировки пузырьком?
2. Какие альтернативные алгоритмы сортировки (или стандартные средства Java) вы предложили бы использовать?
3. Обоснуйте выбор с точки зрения асимптотической сложности и практической производительности (например, использование `Collections.sort()` / `Arrays.sort()` с $O(n \log n)$).

ПКН-2 Способность разрабатывать алгоритмы и программы с использованием современных технологий программирования

1) Владеет объектно-ориентированным языком программирования на уровне знания синтаксиса и семантики, основ стандартной библиотеки.

Результаты обучения (умения и знания), соотнесенные с индикаторами достижения компетенции

Знать: синтаксис и семантику языка Java; основные встроенные типы данных, операции, управляющие конструкции; понятия класс, объект, интерфейс, пакет; базовые классы стандартной библиотеки Java (`java.lang`, `java.util` и др.).

Уметь: писать простые и составные программы на Java с использованием классов и методов; реализовывать основные конструкции ООП (инкапсуляция, наследование, полиморфизм) в Java; использовать базовые классы стандартной библиотеки (строки, коллекции, даты и т.п.).

Типовые контрольные задания

1. Конкретные тестовые вопросы по синтаксису и семантике Java

Вариант 1. Вопросы с выбором ответа

1. Что будет результатом выполнения следующего фрагмента?

```
int a = 5; int b = 2; double c = a / b; System.out.println(c);
```

а) 2.0

б) 2.5

в) 2

г) произойдёт ошибка компиляции

(целочисленное деление 5 / 2 даёт 2, потом неявное приведение к double → 2.0)

1. Какой из вариантов объявления массива целых чисел в Java корректен?

а) `int[] a = new int(10);`

б) `int a[10];`

в) `int[] a = new int[10];`

г) `array<int> a = new array<int>(10);`

1. Какой модификатор доступа делает член класса доступным только внутри текущего класса?

а) `public`

б) `private`

в) `protected`

г) отсутствие модификатора (`package-private`)

1. Какой оператор в Java используется для сравнения значений примитивных типов?

а) `=`

б) `==`

в) `===`

г) `equals`

1. Какое утверждение о ключевом слове `static` верно?

а) Поле `static` принадлежит объекту, а не классу.

б) Метод `static` не может обращаться к нестатическим полям напрямую.

в) Метод `static` нельзя вызывать без создания объекта.

г) Класс с `static`-методами нельзя использовать в других классах.

Вариант 2. Вопросы с кратким открытым ответом

1. Объясните разницу между оператором `==` и методом `equals()` при работе со строками в Java. Ожидается: `==` сравнивает ссылки (один ли объект в памяти), `equals()` — содержимое строк.

2. Что такое перегрузка метода (`overloading`) в Java? Приведите небольшой пример.

3. Что произойдёт, если в Java-классе не объявлен ни один конструктор? Ожидается: будет сгенерирован конструктор по умолчанию без параметров.

2. Конкретное задание: написание фрагмента программы с классами, наследованием и интерфейсами

Задание 1. «Иерархия фигур»

Реализовать простую иерархию классов для описания геометрических фигур.

1. Объявите интерфейс Shape с методом:

```
public interface Shape { double area(); // возвращает площадь фигуры }
```

1. Реализуйте класс Rectangle, который:

- реализует интерфейс Shape;
- имеет приватные поля width и height (тип double);
- содержит конструктор Rectangle(double width, double height);
- реализует метод area() как width * height.

2. Реализуйте класс Circle, который:

- также реализует интерфейс Shape;
- имеет приватное поле radius (тип double);
- содержит конструктор Circle(double radius);
- реализует метод area() как Math.PI * radius * radius.

3. Напишите метод public static double totalArea(Shape[] shapes), который принимает массив фигур и возвращает сумму их площадей.

4. В методе main создайте массив из нескольких прямоугольников и окружностей, выведите на экран общую площадь.

Задание 2. «Сотрудники и расчёт зарплаты»

1. Создайте абстрактный класс Employee со следующими членами:

- защищённые поля name (тип String) и baseSalary (тип double);
- конструктор Employee(String name, double baseSalary);
- абстрактный метод double calculateSalary();
- метод String getName().

2. Создайте класс Manager, наследующий Employee, с дополнительным полем bonus (тип double).

- Конструктор: `Manager(String name, double baseSalary, double bonus);`
 - реализация `calculateSalary()` как `baseSalary + bonus`.
3. Создайте класс `Developer`, наследующий `Employee`, с дополнительным полем `int overtimeHours` и `double overtimeRate`.
- Конструктор: `Developer(String name, double baseSalary, int overtimeHours, double overtimeRate);`
 - реализация `calculateSalary()` как `baseSalary + overtimeHours * overtimeRate`.
4. Напишите метод `printPayroll(Employee[] staff)`, который для каждого сотрудника выводит имя и рассчитанную зарплату.

3. Конкретное задание: поиск и исправление ошибок в коде

Задание 1. «Исправление класса `Account`»

Дан следующий фрагмент кода:

```
public class Account { public String owner; private double balance; public Account(String owner,
double balance) { owner = owner; balance = balance; } public void deposit(double amount) { if
(amount >= 0) { balance = balance + amount; } } public void withdraw(double amount) { if (amount >
balance) { balance = balance - amount; } } public double getbalance() { return Balance; } }
```

Требуется:

1. Найти и исправить все ошибки компиляции (регистр, имена переменных и т.п.).
2. Исправить логические ошибки в конструкторах и методе `withdraw`.
3. По возможности улучшить инкапсуляцию (обсудить модификаторы доступа и соглашения об именовании).

Ожидаемые правки (примерно):

- В конструкторе использовать `this.owner = owner; this.balance = balance;`
- В `withdraw`: условие должно быть `if (amount <= balance);`
- Метод `getBalance()` (с заглавной B) и возвращать `balance`;
- Поле `owner` целесообразно сделать `private`;
- Проверять, что `amount > 0` и т.п.

Задание 2. «Исправление иерархии классов»

Дан фрагмент кода:

```
public class Animal { private String name; public Animal(String name) { name = name; } public void speak() { System.out.println("Some sound"); } } public class Dog extends Animal { public void speak() { System.out.println("Woof"); } public Dog(String name) { Animal(name); } }
```

Требуется:

1. Найти и исправить ошибки в использовании конструкторов и инициализации полей.
2. Объяснить, почему в конструкторе Dog нельзя вызывать Animal(name); как обычный метод.
3. Исправить код так, чтобы:

- поле name корректно инициализировалось;
- конструктор подкласса вызывал конструктор суперкласса.

Ожидаемые правки (примерно):

```
public class Animal { private String name; public Animal(String name) { this.name = name; } public void speak() { System.out.println("Some sound"); } public String getName() { return name; } } public class Dog extends Animal { public Dog(String name) { super(name); } @Override public void speak() { System.out.println("Woof"); } }
```

2) Использует инструментальные средства программирования (IDE, SDK, API, популярные фреймворки и библиотеки).

Результаты обучения (умения и знания), соотнесенные с индикаторами достижения компетенции

Знать: назначение и основные функции IDE (IntelliJ IDEA / Eclipse и др.); принципы работы с JDK/SDK, сборки и запуска Java-приложений; общие принципы использования библиотек и фреймворков (подключение, документация, лицензии).

Уметь: настраивать и использовать IDE для разработки Java-проекта; подключать внешние библиотеки и фреймворки (через Maven/Gradle или вручную); использовать отладчик, просмотр стека вызовов, точки останова.

Типовые контрольные задания

1. Создание и запуск Java-проекта в IDE

«Создание и запуск консольного приложения на Java в IDE»

Цель: закрепить навыки работы с интегрированной средой разработки (IDE), настройку JDK, сборку и запуск простого приложения.

Задание:

1. Запустите выбранную IDE (например, IntelliJ IDEA / Eclipse / NetBeans).

2. Создайте новый проект Java:

- укажите имя проекта, например: FirstJavaApp;
- выберите или добавьте подходящую версию JDK (обсуждается с преподавателем).

3. Создайте класс Main в пакете ru.fa.oop (или в соответствии с принятой на кафедре схемой пакетов).

4. Реализуйте в классе Main метод:

```
public class Main { public static void main(String[] args) { System.out.println("Hello, Java OOP!"); } }
```

1. Сконфигурируйте конфигурацию запуска (Run Configuration), если это требуется IDE:

- укажите основной класс (Main);
- при необходимости настройте параметры VM/приложения.

2. Соберите и запустите приложение из IDE. Убедитесь, что в консоли вывелось сообщение Hello, Java OOP!.

3. Сохраните скриншот окна IDE с видимым исходным кодом и результатом выполнения программы в консоли.

4. Подготовьте краткий текстовый отчёт (0,5–1 страница), в котором отразите:

- название и версию использованной IDE и JDK;
- основные шаги создания и запуска проекта;
- возможные трудности и пути их решения.

2. Задание: подключение внешней библиотеки и её использование в коде

«Подключение и использование внешней библиотеки (пример: JSON)»

Цель: освоить подключение сторонних библиотек к проекту и использование их API.

Вариант с JSON-библиотекой (например, Gson или Jackson):

1. В уже созданном Java-проекте (или новом) добавьте внешнюю библиотеку для работы с JSON:

- вариант А (Maven/Gradle):
- инициализируйте проект как Maven-проект, добавьте в pom.xml зависимость, например, для Gson:

- -
 - `<dependency> <groupId>com.google.code.gson</groupId> <artifactId>gson</artifactId>
<version>2.10.1</version> <!-- версия уточняется --> </dependency>`
 - вариант В (без Maven):
 - скачайте `gson-*.jar` (или другую библиотеку);
 - подключите JAR к проекту через настройки «Project Structure» / «Libraries».
 - Создайте класс `User` с полями `name (String)`, `age (int)`, `email (String)`, геттерами/сеттерами и методом `toString()`.
 - В классе `Main` реализуйте код: Пример с `Gson` (для ориентира, студент может адаптировать):

```
import com.google.gson.Gson; public class Main { public static void main(String[] args) { Gson gson =
new Gson(); User user = new User("Ivan Petrov", 30, "ivan@example.com"); String json =
gson.toJson(user); System.out.println("JSON: " + json); User user2 = gson.fromJson(json, User.class);
System.out.println("Deserialized: " + user2); } }
```
 - создание объекта `User`;
 - сериализация объекта в строку JSON с помощью библиотеки;
 - вывод полученной JSON-строки в консоль;
 - десериализация строки JSON обратно в объект `User`;
 - вывод восстановленного объекта в консоль.
1. Убедитесь, что проект успешно компилируется и запускается, а в консоль выводится корректный JSON и восстановленный объект.
 2. Подготовьте краткий отчёт (0,5–1 страница), в котором:
 - укажите, какую библиотеку и какую версию вы использовали;
 - опишите шаги подключения библиотеки к проекту;
 - приведите фрагменты ключевого кода (сериализация/десериализация);
 - при желании — скриншоты настроек зависимостей в IDE.

(Аналогично можно оформить вариант с библиотекой логирования (SLF4J + Logback) или JDBC-драйвером — структура задания сохраняется: подключить зависимость, написать код, продемонстрировать работу, описать шаги в отчёте.)

3. Отчёт/скринкаст: использование отладчика при поиске ошибки

«Использование отладчика IDE для поиска ошибки в Java-программе»

Цель: сформировать навыки использования встроенного отладчика (breakpoint, пошаговое выполнение, просмотр значений переменных) для поиска и устранения ошибок.

Исходные данные:

Студенту выдаётся (или размещается в СДО) Java-проект с намеренно допущенной логической ошибкой.

Например, программа должна вычислять сумму положительных элементов массива, но даёт неверный результат:

```
public class Main {  
  
    public static void main(String[] args) { int[] data = {1, -2, 3, 4, -5}; int sum = 0; for (int i = 0; i <= data.length; i++) { // здесь ошибка if (data[i] > 0) { sum = sum + data[i]; } } System.out.println("Sum = " + sum); } }
```

Задание:

1. Импортируйте/откройте проект с данным кодом в IDE.
2. Настройте конфигурацию запуска программы, убедитесь, что она компилируется и запускается (возможно, с исключением `ArrayIndexOutOfBoundsException`).
3. Включите отладчик и:
 - установите точку останова (breakpoint) на строке с началом цикла `for`;
 - запустите программу в режиме отладки (Debug).
4. Пошагово выполните цикл:
 - отслеживайте значение переменной `i` и текущее значение `data[i]`;
 - обратите внимание, на каком шаге возникает ошибка или некорректное поведение.
5. Зафиксируйте:
 - на каком значении `i` возникает исключение или неверный результат;
 - какая логическая ошибка допущена в условии цикла.
6. Исправьте код (например, заменить условие `i <= data.length` на `i < data.length`) и повторно запустите программу в режиме отладки, убедившись в корректности результата.
7. Подготовьте краткий отчёт или скринкаст (по требованиям преподавателя), в котором покажите:
 - запуск отладки и установку точки останова;

- пошаговое выполнение цикла;
- просмотр значений переменных в окне отладчика;
- момент, когда обнаружена ошибка, и вариант её исправления;
- результат повторного запуска после исправления.

3) Организует кодовую базу, ориентируется в существующем коде, демонстрирует знание общепринятых соглашений и политик в области оформления кода.

Результаты обучения (умения и знания), соотнесенные с индикаторами достижения компетенции

Знать: основные соглашения по оформлению кода на Java (Java Code Conventions); принципы структурирования проекта (пакеты, модули, слои); понятия «читаемость кода», «рефакторинг», «code style».

Уметь: организовывать классы по пакетам в соответствии с логикой предметной области; читать и понимать чужой код на Java, выделять ключевые элементы; применять базовые приёмы рефакторинга (выделение методов, переименование, удаление дублирования).

Типовые контрольные задания

1. Анализ фрагмента кода и ответы на вопросы по его структуре и качеству

Задание 1. «Анализ класса UserService»

Дан фрагмент кода:

```
public class UserService { public List users; public UserService(List users) { this.users = users; }
public void addUser(String name, int age, String email) { if (name != null && name != "" && age > 0
&& email.contains("@")) { HashMap user = new HashMap(); user.put("name", name); user.put("age",
age); user.put("email", email); users.add(user); } else { System.out.println("Error!"); } } public void
print() { for (int i = 0; i < users.size(); i++) { HashMap u = (HashMap) users.get(i);
System.out.println("User:" + u.get("name") + "," + u.get("age") + "," + u.get("email")); } } }
```

Требуется:

1. Определить и перечислить нарушения стиля и проблемные места (минимум 5 пунктов).
Обратить внимание на:

- использование «сырой» коллекции List и HashMap без generics;
- модификаторы доступа;
- проверки name != "";
- сообщения об ошибках и обработку исключительных ситуаций и т.п.

2. Предложить варианты улучшения структуры класса (какие сущности стоит выделить в отдельные классы/типы).

3. Кратко ответить на вопросы:

- Почему использование «магических строк» ("name", "age", "email") считается плохой практикой?
- Чем опасны «сырые» коллекции без generics?
- Какие преимущества даст введение отдельного класса User?

(Ответ оформляется в виде короткого текста или таблицы «Проблема → Предложенное улучшение».)

2. Рефакторинг предложенного кода с нарушенным стилем

Задание 2. «Рефакторинг класса OrderPrinter»

Дан класс:

```
public class orderPRINTER { public void PrintOrders(ArrayList orders){ for (int i=0;i<orders.size();i++){ Object o = orders.get(i); System.out.println("ORDER:"+o.toString()); } } }
```

Требуется:

1. Привести класс к стандартам оформления кода Java (Java Code Conventions):

- корректно оформить отступы, переносы строк, пробелы;
- переименовать класс, метод, параметры в соответствии с общепринятыми соглашениями (OrderPrinter, printOrders, orders и т.п.);
- добавить необходимые модификаторы доступа (public, private).

2. Ввести generics для коллекции заказов (например, List<Order> вместо ArrayList).

3. Предложить более безопасный способ вывода информации о заказах (например, не полагаться на toString() «как есть», а выводить только нужные поля).

4. Кратко описать, какие именно изменения были внесены и почему они улучшают качество кода.

Задание 3. «Рефакторинг «божественного» класса»

Дан класс:

```
public class ReportManager { public void generateReport(List orders, String format, String fileName) { // фильтрация List filtered = new ArrayList(); for (Object o : orders) { Order order = (Order) o; if (order.status.equals("PAID")) { filtered.add(order); } } // сортировка Collections.sort(filtered, new
```

```
Comparator() { public int compare(Object o1, Object o2) { Order a = (Order) o1; Order b = (Order) o2;
return a.date.compareTo(b.date); } }); // генерация if (format.equals("CSV")) { // ... } else if
(format.equals("PDF")) { // ... } else if (format.equals("XML")) { // ... } // запись в файл try
{ FileWriter fw = new FileWriter(fileName); fw.write("..."); fw.close(); } catch (Exception e)
{ e.printStackTrace(); } }
```

Требуется:

1. Указать, какие принципы хорошего дизайна и стиля здесь нарушены (SRP, открытость/закрытость, типобезопасность, magic strings и т.д.).

2. Предложить план рефакторинга (без полной реализации), например:

- выделить отдельные классы/интерфейсы для фильтрации, сортировки, форматов отчётов, записи в файл;
- заменить List и «сырые» Comparator на обобщённые типы;
- избавиться от цепочки if (format.equals("...")) через паттерн Strategy или фабрику.

3. Привести пример переработанного сигнатуры метода (или нескольких методов/классов), отражающей улучшенную архитектуру.

3. Разработка небольшой кодовой базы (несколько классов и пакетов)

Задание 4. «Мини-проект: управление задачами (Task Manager)»

Цель: отработать навыки структурирования небольшой кодовой базы, соблюдения стиля и организации пакетов.

Исходная постановка:

Необходимо разработать простое консольное приложение управления задачами (to-do list) на Java. Каждая задача имеет: идентификатор, заголовок, описание, приоритет, статус (например: NEW, IN_PROGRESS, DONE).

Требуется:

1. Спроектировать структуру пакетов, например:

- ru.fa.taskmanager.model — классы модели (Task, Priority, Status и др.);
- ru.fa.taskmanager.service — классы, реализующие логику работы с задачами (TaskService);
- ru.fa.taskmanager.ui — классы для текстового интерфейса (меню, обработка команд пользователя).

2. Разработать не менее трёх–пяти классов в пакете model:

- класс Task с полями (id, title, description, priority, status, createdAt и т.п.);
- перечисления Priority и Status;
- при необходимости — дополнительные вспомогательные классы (например, TaskIdGenerator).

3. В пакете service реализовать:

- класс TaskService, обеспечивающий:
- добавление новой задачи;
- смену статуса задачи;
- поиск задач по статусу/приоритету;
- вывод списка задач.
- использовать коллекции Java (List, Map) и generics.

4. В пакете ui реализовать:

- класс ConsoleApp (или Main), который:
- отображает простое текстовое меню;
- позволяет пользователю добавлять задачи, изменять статус, просматривать список задач.

5. Соблюдать конвенции оформления Java-кода:

- имена классов с заглавной буквы в CamelCase;
- имена методов и переменных — с маленькой буквы, в формате camelCase;
- осмысленные имена пакетов, классов и методов;
- аккуратное форматирование кода (отступы, пробелы, переносы строк).

6. Подготовить краткое текстовое описание структуры кодовой базы:

- перечень пакетов и их назначение;
- список основных классов и их ответственность;
- краткий пример сценария использования (какие методы вызываются при добавлении/изменении задачи).

4) Проектирует текстовый, программный или графический интерфейс программной системы исходя из ее назначения.

Результаты обучения (умения и знания), соотнесенные с индикаторами достижения компетенции

Знать: виды пользовательских интерфейсов (CLI, GUI, web-интерфейс) и их особенности; базовые принципы удобства и эргономики интерфейса; основные библиотеки/фреймворки для создания интерфейсов на Java (Swing, JavaFX и др. – на уровне обзора).

Уметь: проектировать структуру текстового меню/CLI для прикладной задачи; реализовывать простой интерфейс взаимодействия с пользователем на Java (консольный или графический); связывать интерфейс с объектной моделью приложения.

Типовые контрольные задания

1. Разработка простого текстового меню для выбранной предметной области

«Текстовое меню для подсистемы учета заказов»

Цель: отработать навыки проектирования и реализации текстового (консольного) интерфейса взаимодействия с пользователем.

Исходная предметная область:
Подсистема «Учет заказов клиентов» (как в мини-задачах/проекте по курсу).

Задание:

1. Спроектируйте структуру текстового меню для работы с заказами. Меню верхнего уровня должно содержать, например, следующие пункты:
2. Для каждого пункта меню опишите:
3. Реализуйте текстовый интерфейс в виде Java-программы:
4. Обеспечьте:
5. Подготовьте краткий отчет (0,5–1 страница), в котором:

2. Задание: описание структуры GUI для задачи (набор окон/форм и элементов)

«Описание структуры графического интерфейса подсистемы учета клиентов»

Цель: сформировать умение проектировать структуру GUI-приложения, исходя из требований предметной области.

Исходная предметная область:
Подсистема «Учет клиентов» (например, для небольшой фирмы / банка).

Задание:

1. На основе текстового описания задачи (или собственного мини-проекта) опишите структуру графического интерфейса приложения:

2. Для каждого окна необходимо описать:
3. Оформите описание в одном из вариантов:
4. Включите в описание:
5. В отчете (1–2 страницы) кратко ответьте на вопросы:

(При необходимости можно отдельно указать, что реализация самого GUI-кода в данном задании не обязательна — главное, структура и обоснование.)

3. Защита мини-проекта, в котором реализован интерфейс к разработанной функциональности

«Защита мини-проекта с пользовательским интерфейсом»

Цель: оценить умение студента обосновывать принятые решения по проектированию интерфейса и демонстрировать работу разработанного приложения.

Исходные данные:

Студент завершил мини-проект (например, «Система учета заказов/клиентов/задач») с реализованным текстовым или графическим интерфейсом.

Требования к защите:

1. Краткая презентация проекта (5–7 минут):
2. Демонстрация пользовательского интерфейса:
3. Обоснование проектных решений:
4. Ответы на вопросы преподавателя:
5. Оценивание проводится по критериям:

Примеры практико-ориентированных заданий

1. Реализовать класс иерархию геометрических фигур (круг, прямоугольник, треугольник), где базовый класс содержит абстрактный метод вычисления площади, а подклассы реализуют его по-своему. Проверяется понимание наследования и полиморфизма.

2. Написать метод, который на вход принимает список целых чисел и возвращает новый список, содержащий только уникальные элементы исходного (использовать коллекцию Set). Оценить сложность решения.

3. Применить шаблон проектирования «Наблюдатель»: создать класс наблюдателя, реагирующего на изменение состояния другого объекта (например, при обновлении курса валют наблюдатель выводит уведомление).

Примерные вопросы для подготовки к экзамену

1. Основные принципы объектно-ориентированного программирования: инкапсуляция, наследование, полиморфизм.
2. Классы и объекты в языке Java. Определение, структура, жизненный цикл.
3. Перегрузка и переопределение методов. Отличия и примеры.
4. Конструкторы и модификаторы доступа в Java.
5. Абстрактные классы и интерфейсы. Сравнительный анализ и область применения.
6. Ключевые слова `this`, `super`, `final` и их роль в ООП-модели Java.
7. Обработка исключений в Java: механизм, иерархия, создание пользовательских исключений.
8. Стандартные коллекции Java: `List`, `Set`, `Map`. Примеры и области применения.
9. Обобщённые типы (generics) в Java. Мотивация, синтаксис, ограничения.
10. Структура и особенности `ArrayList`, `LinkedList`, `HashMap`, `TreeMap`.
11. Механизм автоупаковки и анбоксинга (`autoboxing/unboxing`).
12. Потоки ввода-вывода в Java: работа с файлами, сериализация.
13. Классы `BufferedReader`, `FileReader`, `Files`, `Paths`: возможности и различия.
14. Лямбда-выражения и функциональные интерфейсы: синтаксис и назначение.
15. `Stream API`: архитектура, основные операции, примеры использования.
16. Модульная система Java (JPMS): структура, особенности, `module-info.java`.
17. Структура Java-программы. Жизненный цикл исполнения.
18. Модификаторы доступа и область видимости в Java.

19. Шаблоны проектирования: Singleton, Factory, Observer и их реализация на Java.
20. Понятие SOLID-принципов и их реализация в Java.
21. Понятие и реализация принципа Dependency Injection (DI) в Spring Framework.
22. IoC-контейнер и аннотации в Spring: @Component, @Service, @Autowired.
23. Основы Spring Boot: структура проекта, автоконфигурация, аннотации.
24. Тестирование с использованием JUnit 5: структура теста, основные аннотации.
25. Жизненный цикл unit-тестов и концепция TDD (Test-Driven Development).
26. Параметризованные тесты в JUnit. Валидация и тест-кейсы.
27. Java vs C++: управление памятью, сборка, реализация ООП.
28. Java vs Python: строгость типизации, модель исполнения, парадигмы.
29. Сравнение Java и C#: синтаксис, свойства (properties), .NET vs JVM.
30. Использование аннотаций и механизм рефлексии в Java.
31. Проектирование и реализация мини-приложений с применением ООП.
32. Архитектура многомодульного проекта в Java.
33. Практика использования коллекций и дженериков в реальных задачах.
34. Роль интерфейсов функционального программирования в современном Java-коде.
35. Типичные ошибки при реализации ООП в Java и способы их предотвращения.

Пример экзаменационного билета

Экзаменационный билет №

1. Классы и объекты в языке Java. Определение, структура, жизненный цикл (20 баллов)
2. Понятие SOLID-принципов и их реализация в Java (20 баллов)
3. Применить шаблон проектирования «Наблюдатель»: создать класс наблюдателя, реагирующего на изменение состояния другого объекта (например, при обновлении курса валют наблюдатель выводит уведомление) (20 баллов)

8. Перечень основной и дополнительной учебной литературы, необходимой для освоения дисциплины

Основная литература:

1. Гуськова, Ольга Ивановна Объектно ориентированное программирование в Java : Учебное пособие / Московский педагогический государственный университет Москва : Московский педагогический государственный университет, 2018 240 с. ВО - Магистратура <https://znanium.com/catalog/document?id=339668> ISBN 978-5-4263-0648-6. [БИК ID: RU\infra-m\znanium\bibl\1020593]
2. Коротеев, М.В. Введение в Android разработку на Java : Учебное пособие / М.В. Коротеев, А.Ю. Шаталова Электрон. дан. Москва : КноРус, 2026 231 с. Режим доступа: book.ru Internet access <https://book.ru/book/960287> ISBN 978-5-406-15558-5. [БИК ID: RU\bookru\bibl\960287]

Дополнительная литература:

1. Шнейдеров, Е. Н. Разработка приложений на языке Java. Лабораторный практикум [Электронный ресурс] : пособие / Шнейдеров Е. Н.,Писарчик А. Ю.,Казючиц В. О. БГУИР : БГУИР, 2023 92 с. Рекомендовано УМО по образованию в области информатики и радиоэлектроники в качестве пособия для специальности 1-39 03 02 «Программируемые мобильные системы» Книга из коллекции БГУИР - Информатика <https://e.lanbook.com/book/479516> ISBN 978-985-543-561-8. [БИК ID: RU-LAN-BOOK-479516]
2. Бобырь, Максим Владимирович Программирование на языке Java. Практический курс : Учебное пособие / Юго-Западный государственный университет 1 Москва : ООО "Научно-издательский центр ИНФРА-М", 2025 189 с. (Высшее образование) ВО - Бакалавриат <https://znanium.ru/catalog/document?id=460819> ISBN 978-5-16-020136-8 ISBN 978-5-16-112676-9 (электр. издание) . [БИК ID: RU\infra-m\znanium\bibl\2160989]
3. Введение в современную Android-разработку на языке Java : учебное пособие / Рысин М. Л. Ч. 1 : Введение в современную Android-разработку на языке Java. Часть 1 : учебное пособие . Ч. 1 / Рысин М. Л. Москва : РТУ МИРЭА, 2023 132 с. Книга из коллекции РТУ МИРЭА - Информатика <https://e.lanbook.com/book/382586> ISBN 978-5-7339-1895-2. [БИК ID: RU-LAN-BOOK-382586]

4. Курбатова, И. В. Основы программирования на языке Java [Электронный ресурс] : учебное пособие для вузов / Курбатова И. В., Печкуров А. В. ; Печкуров А. В. Санкт-Петербург : Лань, 2024 348 с. Книга из коллекции Лань - Информатика <https://e.lanbook.com/book/385928> ISBN 978-5-507-48515-4. [БИК ID: RU-LAN-BOOK-385928]

5. Введение в современную Android-разработку на языке Java / Рысин М. Л. Ч. 2 : Введение в современную Android-разработку на языке Java. Часть 2 : учебное пособие . Ч. 2 / Рысин М. Л. Москва : РТУ МИРЭА, 2024 110 с. Книга из коллекции РТУ МИРЭА - Информатика <https://e.lanbook.com/book/420962> ISBN 978-5-7339-2146-4. [БИК ID: RU-LAN-BOOK-420962]

9. Перечень ресурсов информационно-телекоммуникационной сети «Интернет», необходимых для освоения дисциплины

1. Электронная библиотека Финансового университета (ЭБ) <http://elib.fa.ru/> (<http://library.fa.ru/files/elibfa.pdf>)
2. Информационно-образовательный портал Финуниверситета: <https://org.fa.ru>
3. Научная электронная библиотека eLibrary.ru <http://elibrary.ru>

10. Методические указания для обучающихся по освоению дисциплины

Общие рекомендации по изучению курса: Дисциплина сочетает теоретический материал и значительный практический компонент. Рекомендуется активно участвовать в лекциях – обращать внимание на основные определения (например, принципы ООП, структуры данных, шаблоны проектирования) и примеры кода, задавать вопросы по неясным моментам. После лекции полезно просматривать конспект и сверяться с рекомендованным учебником для углубления понимания.

На практических занятиях (семинарах) основная задача – научиться применять теорию на практике: писать код, разбирать ошибки, обсуждать варианты реализации. Для эффективного освоения, перед каждым семинаром студенту стоит повторно изучить тему лекции и попытаться выполнить базовые упражнения самостоятельно. Например, перед занятием по коллекциям – потренироваться создавать список и выполнять над ним операции. Не бойтесь ошибаться при написании кода во время занятий: каждое обнаруженное и исправленное ошибка – шаг к лучшему пониманию языка. Полезно брать с собой ноутбук либо заранее готовить шаблоны проектов, чтобы на семинаре фокусироваться на логике решения задач.

Самостоятельная работа играет большую роль в овладении навыками программирования. В рамках данной дисциплины предусмотрен курсовой мини-проект – начать работу над ним следует задолго до дедлайна. Разбейте проект на этапы: (1) постановка задачи и проектирование (нарисуйте схему классов, спланируйте функциональность), (2) реализация основных классов и методов, (3) отладка и тестирование, (4) рефакторинг и улучшение структуры, (5) написание отчета и подготовка презентации (если требуется). Регулярно показывайте промежуточный результат преподавателю или наставнику – это поможет направить работу в правильное русло. Также полезно сверяться с профессиональными требованиями.

Использование ресурсов: В процессе обучения настоятельно рекомендуется обращаться к документации и сообществам. Официальная JavaDoc – ваш главный справочник по классам и методам: привыкайте искать там информацию о незнакомых функциях. При возникновении трудностей с кодом – формулируйте проблему и ищите решение (в литературе или на доверенных форумах, таких как StackOverflow). Однако важно придерживаться академической добросовестности: если вы заимствуете фрагменты кода или идеи из внешних источников, старайтесь их понять, прокомментировать и не копировать без разбора.

Связь с профессиональной деятельностью: Освоение данной дисциплины подготовит вас к реальным задачам в индустрии. Навыки ООП и владение Java требуются

программистам в разработке самых различных приложений – от мобильных до корпоративных. Поэтому относитесь к заданиям серьёзно: то, как вы научитесь структурировать код и работать в команде над проектом сейчас, повлияет на вашу эффективность как будущего специалиста. Развивайте не только навыки кодирования, но и soft skills: работу с источниками информации, умение задавать вопросы, сотрудничать с коллегами (например, в групповых обсуждениях на семинарах).

Организация времени: Планируйте учебную нагрузку – 108 часов в семестре соответствуют примерно 6–8 часам в неделю, включая все виды работ. Еженедельно выделяйте время на: повторение конспектов (1–2 часа), выполнение домашних задач или продолжение проекта (3–4 часа), чтение литературы (2 часа). Во время выполнения проекта может понадобиться больше времени; начните заранее, чтобы избежать аврала.

Консультации и обратная связь: При затруднениях не стесняйтесь обращаться к преподавателю на консультациях. Лучше задать вопрос и разобраться, чем оставаться в неопределенности. Преподаватель может подсказать, где найти нужную информацию или как правильнее организовать решение. Также полезно обмениваться опытом с одноклассниками: обсуждение решений (не списывание, а именно обмен идеями!) часто помогает увидеть иную точку зрения и улучшить свой код.

Рекомендации по выполнению проектной работы

Проектная работа является формой текущего контроля по дисциплине и направлена на проверку практических навыков применения ООП-подхода и технологий программирования.

1. Выбор темы проекта:

- Рекомендуется выбрать тему из предложенного перечня (например, система учета финансов, электронный дневник, REST-приложение на Spring Boot), либо предложить собственную, согласовав с преподавателем.
- Работа должна включать не менее 3-х сущностей с классами, и использовать принципы наследования, инкапсуляции, полиморфизма.

2. Структура реализации:

- Используйте архитектурную схему: как минимум, слои модель – логика – пользовательский интерфейс или API.
- Примените один или несколько шаблонов проектирования (например, Singleton для хранилища данных, Factory для генерации сущностей, Observer для уведомлений).
- Используйте коллекции Java (Map, List, Set) для хранения и обработки данных.

- При необходимости – реализуйте ввод/вывод данных из/в файл (CSV, JSON, текстовый).
- Для Web-проектов – продемонстрируйте использование Spring Boot: контроллер, сервис, REST API.

3. Требования к оформлению:

- Проект должен сопровождаться README-документом, в котором кратко изложены:
- цель проекта;
- используемые технологии;
- структура классов и краткое описание логики;
- примеры запуска и вывода;
- описание выявленных и устраненных ошибок.
- При наличии – представить unit-тесты на ключевые методы, оформленные в JUnit.

4. Этапы выполнения:

- 1 неделя: выбор темы, постановка задачи;
- 2–3 неделя: проектирование структуры классов, определение логики;
- 4–5 неделя: реализация кода, модульное тестирование;
- 6 неделя: рефакторинг, оформление документации и подготовка защиты.

5. Критерии оценки:

- полнота реализации и соответствие требованиям;
- использование ООП-принципов и паттернов;
- качество кода (структурированность, читаемость);
- наличие и качество тестирования;
- оформление отчета;
- демонстрация работы проекта.

6. Защита проекта:

- Включает показ исходного кода, демонстрацию работы программы и ответы на вопросы преподавателя по архитектурным решениям, реализациям, примененным библиотекам и технологиям.

11. Перечень информационных технологий, используемых при осуществлении образовательного процесса по дисциплине, включая перечень необходимого программного обеспечения и информационных справочных систем

Комплект лицензионного программного обеспечения:

1. Операционная система
2. Текстовый редактор
3. Пакет офисных программ
4. Kaspersky

Современные профессиональные базы данных и информационные справочные системы:

1. Информационно-правовая система «Гарант»
2. Информационно-правовая система «Консультант Плюс»
3. Электронная энциклопедия: <http://ru.wikipedia.org/wiki/Wiki>
4. Система комплексного раскрытия информации «СКРИН» - <http://www.skrin.ru/>

Сертифицированные программные и аппаратные средства защиты информации:

1. не предусмотрены

12. Описание материально-технической базы, необходимой для осуществления образовательного процесса по дисциплине

1. **Компьютерный класс** для проведения учебных занятий, предусмотренных программой, в том числе групповых и индивидуальных консультаций, текущего контроля и промежуточной аттестации, оснащенный оборудованием и техническими средствами обучения: мебель аудиторная (столы, стулья, доска аудиторная), персональные компьютеры, набор демонстрационного оборудования (проектор, экран)

2. **Учебная аудитория** для проведения учебных занятий, предусмотренных программой, в том числе групповых и индивидуальных консультаций, текущего контроля и промежуточной аттестации, оснащенная оборудованием и техническими средствами обучения: мебель аудиторная (столы, стулья, доска аудиторная), набор демонстрационного оборудования (проектор, экран)

3. Библиотека, читальный зал