

Московский государственный технический университет
имени Н.Э. Баумана

Факультет «Информатика и системы управления»
Кафедра «Компьютерные системы и сети»

М.В. Фетисов

**Методические указания
по выполнению лабораторных работ и домашнего задания
по дисциплине «Современные средства разработки
программного обеспечения»**

Электронное учебное издание



CC BY-SA 3.0

УДК 004.415.2

Рецензент:

Фетисов М.В.

Методические указания по выполнению лабораторных работ и домашнего задания по дисциплине «Современные средства разработки программного обеспечения». Электронное учебное издание. – М.: МГТУ имени Н.Э. Баумана, 2021, 66 с.

Учебное издание содержит указания и требования к порядку выполнения лабораторных работ и домашнего задания. Дается теоретический материал и предлагается пример реализации программ, аналогичных разрабатываемым на лабораторных работах и при выполнении домашнего задания. Определяются цели и объем, требования к отчетам, а также приводятся варианты заданий и примерный список контрольных вопросов для защиты выполненных заданий.

Для студентов 3 курса специальностей ИУ6 МГТУ имени Н.Э. Баумана.

Рекомендовано учебно-методической комиссией факультета «Информатика и системы управления» МГТУ им. Н.Э. Баумана

Электронное учебное издание

Михаил Вячеславович Фетисов

**Методические указания по выполнению лабораторных работ и домашнего задания по дисциплине
«Современные средства разработки программного обеспечения».**



CC BY-SA 3.0

Оглавление

ВВЕДЕНИЕ.....	6
1 ОБЩЕЕ ОПИСАНИЕ ПРАКТИКУМА.....	8
1.1 ВЗАИМОСВЯЗЬ ЭЛЕМЕНТОВ ПРАКТИКУМА.....	9
1.2 ТРЕБОВАНИЕ К СТИЛЮ КОДИРОВАНИЯ.....	9
1.3 УСТАНОВКА И НАСТРОЙКА GNU/LINUX UBUNTU РЕЛИЗА 20.04 LTS.....	10
1.4 ИСПОЛЬЗОВАНИЕ ВУЗОВСКОГО РЕПОЗИТОРИЯ ИСХОДНОГО КОДА.....	11
2 ДОМАШНЯЯ РАБОТА. ВЫДЕЛЕНИЕ ОБЩЕЙ ЧАСТИ КОДА.....	13
2.1 ВЫДЕЛЕНИЕ ОБЩЕЙ ЧАСТИ КОДА.....	13
2.2 ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ.....	14
2.3 ВАРИАНТЫ ЗАДАНИЙ.....	15
2.4 ТРЕБОВАНИЯ К ОТЧЕТУ.....	21
2.5 КОНТРОЛЬНЫЕ ВОПРОСЫ.....	21
3 ЛАБОРАТОРНАЯ РАБОТА №1. ИСПОЛЬЗОВАНИЕ ВСТРОЕННЫХ В СТАНДАРТНУЮ БИБЛИОТЕКУ C++ ПАТТЕРНОВ.....	22
3.1 ОБОБЩЕННЫЕ КОНТЕЙНЕРЫ СТАНДАРТНОЙ БИБЛИОТЕКИ C++.....	22
3.1.1 Последовательные контейнеры.....	22
3.1.2 Ассоциативные контейнеры.....	23
3.1.3 Неупорядоченные ассоциативные контейнеры.....	23
3.2 УМНЫЕ УКАЗАТЕЛИ.....	24
3.3 ИСПОЛЬЗОВАНИЕ ВУЗОВСКОГО РЕПОЗИТОРИЯ КОДА.....	24
3.4 ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ.....	24
3.5 ВАРИАНТЫ ЗАДАНИЙ.....	26
3.6 ТРЕБОВАНИЯ К ОТЧЕТУ.....	26
3.7 КОНТРОЛЬНЫЕ ВОПРОСЫ.....	26
4 ЛАБОРАТОРНАЯ РАБОТА №2. РЕАЛИЗАЦИЯ ТИПОВЫХ РЕШЕНИЙ С ПРИМЕНЕНИЕМ ШАБЛОНОВ ПРОЕКТИРОВАНИЯ.....	28
4.1 ШАБЛОНЫ ПРОЕКТИРОВАНИЯ.....	28
4.2 ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ.....	31
4.3 ВАРИАНТЫ ЗАДАНИЙ.....	33
4.4 ТРЕБОВАНИЯ К ОТЧЕТУ.....	35
4.5 КОНТРОЛЬНЫЕ ВОПРОСЫ.....	35
5 ЛАБОРАТОРНАЯ РАБОТА №3. ОПИСАНИЕ МОДЕЛИ, ИЗОЛЯЦИЯ ПРЕДМЕТНОЙ ОБЛАСТИ.....	37
5.1 ИЗОЛЯЦИЯ ПРЕДМЕТНОЙ ОБЛАСТИ.....	37

5.2	ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ.....	39
5.3	ВАРИАНТЫ ЗАДАНИЙ.....	41
5.4	ТРЕБОВАНИЯ К ОТЧЕТУ.....	41
5.5	КОНТРОЛЬНЫЕ ВОПРОСЫ.....	42
6	ЛАБОРАТОРНАЯ РАБОТА №4. РЕФАКТОРИНГ. ВЫДЕЛЕНИЕ СУЩНОСТЕЙ, ЗНАЧЕНИЙ И СЛУЖБ МОДЕЛИ, ИЗОЛЯЦИЯ ОГРАНИЧЕННОГО КОНТЕКСТА.....	43
6.1	MDD, СТРУКТУРА МОДЕЛИ, УГЛУБЛЯЮЩИЙ РЕФАКТОРИНГ.....	43
6.2	ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ.....	47
6.3	ВАРИАНТЫ ЗАДАНИЙ.....	49
6.4	ТРЕБОВАНИЯ К ОТЧЕТУ.....	52
6.5	КОНТРОЛЬНЫЕ ВОПРОСЫ.....	52
7	ЛИТЕРАТУРА.....	54
	ПРИЛОЖЕНИЕ А.....СОГЛАШЕНИЕ О СТИЛЕ КОДИРОВАНИЯ	55
	ПРИЛОЖЕНИЕ Б.....НАСТРОЙКА РАБОТЫ С ВУЗОВСКИМ РЕПОЗИТОРИЕМ КОДА.....	62
	ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ, СОКРАЩЕНИЙ И ТЕРМИНОВ.....	66

Аннотация

Настоящее электронное учебное издание содержит указания и требования к порядку выполнения лабораторных работ и домашнего задания по дисциплине «Современные средства разработки программного обеспечения». Дается теоретический материал и предлагается пример реализации программ, аналогичных разрабатываемым на лабораторных работах и при выполнении домашнего задания. Определяются цели и объем, требования к отчетам, а также приводятся варианты заданий и примерный список контрольных вопросов для защиты выполненных заданий.

Пособие предназначено для студентов третьего курса специальности «Компьютерные системы и сети».

Введение

Домашнее задание и лабораторные работы по дисциплине «Современные средства разработки программного обеспечения» посвящены приобретению практических навыков проектирования и разработки программ в парадигме объектно-ориентированного программирования (ООП) на современных стандартах языка C++, а также с применением некоторых техник предметно-ориентированного проектирования (Domain Driven Design).

Практикум по дисциплине «современные средства разработки программного обеспечения» обеспечивает базовые знания для формирования профессиональных компетенций, обозначенных в соответствии с самостоятельно установленным образовательным стандартом (СУОС 3++), основной профессиональной образовательной программой по направлению подготовки 09.03.01 «Информатика и вычислительная техника» [1]:

- способность использовать современные информационные технологии и программные средства отечественного и иностранного производства при решении задач профессиональной деятельности (ОПКС-2);
- способность решать стандартные задачи профессиональной деятельности на основе математической, информационной и библиографической культуры с применением информационно-коммуникационных технологий и с учетом основных требований информационной безопасности (ОПКС-3);
- способность собирать аппаратуру, и устанавливать отечественное и иностранное программное обеспечение для информационных и автоматизированных систем (ОПКС-5);
- способность осваивать отечественные и зарубежные методики использования программных средств для решения практических задач (ОПКС-9);
- способность выполнять работы по созданию и модификации программных или программно-аппаратных компонентов ИТ-систем цифровой экономики (ПКС-4).

А также самостоятельно устанавливаемым образовательным стандартом (СУОС 3++), основной профессиональной образовательной программой по направлению подготовки 09.03.03 «Прикладная информатика» [2]:

- способность использовать современные информационные технологии и программные средства отечественного и иностранного производства, при решении задач профессиональной деятельности (ОПКС-2);
- способность решать стандартные задачи профессиональной деятельности на основе математической, информационной и библиографической культуры с применением информационно-коммуникационных технологий и с учетом основных требований информационной безопасности (ОПКС-3);
- способность устанавливать отечественное и иностранное программное и аппаратное обеспечение для информационных и автоматизированных систем (ОПКС-5);

- способность, используя эффективные подходы и средства, разрабатывать алгоритмы и программы, пригодные для практического применения (ОПКС-7);
- способность выполнять концептуальное, функциональное и логическое проектирование компонентов программных или программно-аппаратных информационных систем цифровой экономики (ПКС-4).

Дисциплина «Современные средства разработки программного обеспечения» базируется на знаниях в следующих дисциплинах:

- «Основы программирования» [3],
- «Объектно-ориентированное программирование» [4],
- «Технология разработки программных систем» [5].

1 Общее описание практикума

Программное обеспечение (ПО) имеет естественную тенденцию к постоянному усложнению, в то время как сроки разработки имеют потребность в сокращении. Такая ситуация становится возможной благодаря развитию разнообразных приемов организации разработки (методологий), инструментальных средств, а также техник программирования.

Перечислим основные признаки сложного ПО:

- **комплексность** – то есть, собственно, сложность, которая проявляется в большом количестве объектов, их вложенности друг в друга и связями между ними;
- **длительный жизненный цикл** – то есть необходимость не только разработать ПО, но и поддерживать его развитие на протяжении длительного времени;
- **работа в команде** – что подразумевает невозможность разработать и поддерживать сложное ПО в одиночку за приемлемое время, а также устойчивость проекта в условиях текучки команды разработки.

Особенно ярко разрешение проблемы сокращения времени разработки сложного ПО проявляется в гибких методиках разработки программного обеспечения, в которых делается особый акцент на технике построения устойчивого к изменениям кода, так как в условиях интерактивной разработки могут производиться частые и существенные уточнения требований к ПО, влекущие сильные изменения в коде программ.

Для реализации устойчивого к изменениям кода используются следующие приемы:

- применение парадигмы объектно-ориентированного программирования;
- построение многоуровневой архитектуры приложения с изоляцией предметной области [10];
- применение устойчивых к изменению техник, основанных на идиомах RAII [6], шаблонах проектирования [7], принципах SOLID [8] и других;
- использование инструментальных средств, предоставляющих возможность статического анализа кода непосредственно во время его написания;
- максимальное использование приёмов автодокументирования кода (см. раздел 1.2).

Помимо написания устойчивого к изменениям кода, важным при разработке сложного ПО является работа в команде с общим репозиторием исходного кода (см. раздел 1.4). Данная техника используется при выполнении домашнего задания и в лабораторном практикуме. В частности, результаты выполнения домашней и лабораторных работ необходимо сохранять в вузовском репозитории с выполнением сценариев непрерывной интеграции [12], дается возможность сформировать «команду» из двух человек.

Для получения задания на весь практикум необходимо зарегистрироваться в вузовском репозитории кода (<https://bmstu.codes>) и выслать по электронной почте преподавателю никнейм.

Преподаватель даст доступ к группе, в которой нужно создать проекты и вышлет в ответ перечень заданий.

При выполнении работ рекомендуется использовать образцы: <https://bmstu.codes/lxx/mstd/homework-samples/dynamic-polymorphism> (домашнее задание) и <https://bmstu.codes/lxx/mstd/laboratory-works> (лабораторные работы).

Отчет должен быть составлен в виде текстового файла в формате Markdown с наименованием «README.md», который помещается в проект, а проект в вузовский репозиторий.

Все задания, примеры и шаблоны для выполнения домашнего задания и лабораторных работ оформляются на языке программирования C++ с применением стандарта ISO/IEC 14882:2017 [13], который более известен в виде обозначения C++17.

1.1 Взаимосвязь элементов практикума

Некоторые лабораторные работы должны выполняться на основе уже выполненных других заданий. В таблице 1 приведены взаимосвязь элементов практикума.

Таблица 1 – Взаимосвязь элементов практикума и условие формирования отчета

Название практикума	Зависимость от других элементов
ДР «Выделение общей части кода»	нет
ЛР №1 «Использование встроенных в стандартную библиотеку C++ паттернов»	нет
ЛР №2 «Реализация типовых решений с применением паттернов проектирования и принципов SOLID»	нет
ЛР №3 «Описание модели, изоляция предметной области»	ДР
ЛР №4 «Рефакторинг. Выделение сущностей, значений и служб модели»	ЛР №3

1.2 Требование к стилю кодирования

При разработке сложного ПО большое значение имеет формирование устойчивого к изменениям кода (такой код также часто называют «адаптивным»). При этом, в условиях длительного жизненного цикла проекта и работы в нём команды разработчиков, которая может меняться, особое значение имеет соглашение о стиле кодирования.

В приложении Б приводится перечень основных пунктов соглашения о стиле кодирования на языке программирования C++. Важно заметить, что пункты соглашения представляют собой рекомендации, которые могут быть нарушены по следующим причинам:

- нарушение улучшает читаемость кода,
- нарушение необходимо для достижения высокой производительности программы.

Если какой-либо пункт соглашения был нарушен, необходимо в коде оставить комментарий, объясняющий причину такого нарушения.

1.3 Установка и настройка GNU/Linux Ubuntu релиза 20.04 LTS

При выполнении домашнего задания и лабораторных работ необходимо использовать вузовский репозиторий исходного кода с применением сценария непрерывной интеграции. Этот сценарий выполняется на сервере в среде операционной системы GNU/Linux, поэтому рекомендуется записывать и отлаживать программу в среде той же операционной системы. В качестве дистрибутива операционной системы GNU/Linux рекомендуется взять Ubuntu версии 20.04 LTS (или более новой) в силу наибольшей совместимости со средой исполнения средств непрерывной интеграции.

В таблице 2 приведен перечень операций по установке операционной системы GNU/Linux Ubuntu релиза 20.04 LTS.

Таблица 2 – Операции по установке операционной системы GNU/Linux Ubuntu релиза 20.04 LTS

№№	Операция	Пояснения
1	Выбор способа работы с операционной системой	Операционную систему GNU/Linux Ubuntu релиза 20.04 LTS можно установить следующими способами: <ul style="list-style-type: none"> - на компьютер (ноутбук) как основную ОС; - на компьютер (ноутбук) совместно с другой ОС, например, с MS Windows; - в среду виртуализации, например, VirtualBox.
2	Установка GNU/Linux Ubuntu релизов 18.04 LTS и 20.04 LTS в соответствии с выбранным способом	Каким образом устанавливать GNU/Linux Ubuntu релиза 20.04 LTS можно прочитать/посмотреть на различных ресурсах Интернет.
3	Установка пакетов разработки	Для установки компилятора C++ нужно выполнить в терминале следующую команду: <code>sudo apt install build-essential</code>
4	Установка среды разработки	В качестве среды разработки рекомендуется использовать Visual Studio Code. Это можно сделать с сайта продукта (https://code.visualstudio.com).

№№	Операция	Пояснения
		<p>В качестве среды разработки можно также использовать Qt Creator, установив пакет «Open Source» с сайта продукта (https://www.qt.io/download). При этом, при установке на ОС GNU/Linux нужно учесть следующие особенности:</p> <ul style="list-style-type: none"> - после скачивания файла установщика, необходимо изменить его свойства, добавив разрешение на исполнение; - после установки Qt в терминале нужно выполнить следующую команду¹: <pre>sudo apt-get install libgl1-mesa-dev</pre>

1.4 Использование вузовского репозитория исходного кода

Вузовский репозиторий исходного кода [12] использует свободную версию системы управления репозиториями кода GitLab, которая предполагает использование системы управления версиями Git.

Система управления версиями Git является распределенной, то есть:

- предоставляет каждому разработчику локальную историю разработки;
- не требует захватывать файлы для выполнения изменений.

Также Git позволяет выполнять быстрое и удобное разделение и слияние версий ПО. В настоящее время Git является одной из наиболее распространенных систем управления версиями.

Система GitLab является Web-приложением и системой управления репозиториями кода для Git. Бесплатная лицензия GitLab предоставляет следующие возможности:

- организацию публичных и частных репозиториях;
- управление правами, группами;
- импорт проектов, в том числе с GitHub;
- работу с вики-страницами для проекта;
- API для внешнего управления;
- работу с доской идей и задач;
- настройку меток, ветх, шаблонов;
- возможности поиска;
- комментирование действий и изменений, объединение веток;
- реализацию непрерывной интеграции и непрерывного развертывания (CI/CD);

¹ Данная команда не обязательна для работы с консольными приложениями без использования библиотеки Qt, но если возникнет необходимость работать с библиотекой Qt, то без выполнения этой команды, при компиляции программы будет выдаваться сообщение об ошибке.

- отслеживание изменений и прогресса;
- отслеживание времени и т.д.

В приложении Б приводится последовательность действий для установки и настройки системы Git, а также подключения к GitLab и настройки работы с проектами в этой системе.

2 Домашняя работа. Выделение общей части кода

Цель: разработать программу на языке C++ управления сохраняемой в файл коллекцией карточек заданного вида, работающую в режиме командной строки, в которой общая часть кода отделена от коллекции карточек. Работа с внешним репозиторием кода и сценарием непрерывной интеграции.

Студенты на базе предложенных вариантов реализации самостоятельно реализуют программу на языке C++, работающую в режиме командной строки, в которой выполнено выделение общего кода для коллекции заданного вида карточек. Обязательно выделение инварианта классов карточки. Результат работы сохраняют в вузовский репозиторий кода, добиваются прохождения сценария непрерывной интеграции.

2.1 Выделение общей части кода

Выделение общей части кода – наиболее часто встречающаяся задача при разработке ПО. В рамках объектно-ориентированного программирования (ООП) эта задача может решаться с использованием полиморфизма. В языке программирования C++ это может достигаться следующим образом:

- выделение общей части в параметризованный класс (шаблон) с одним или несколькими параметрами – параметрический полиморфизм;
- выделение общей части в базовый класс с виртуальными методами (динамический полиморфизм).

В вузовском репозитории кода есть примеры реализации домашнего задания, которые рекомендуется использовать. В проекте [«Dirty implementation»](#) представлена реализация без выделения общей части кода. В проекте [«Parametric polymorphism»](#) представлена реализация выделения общей части кода с использованием параметрического полиморфизма. В проекте [«Dynamic polymorphism»](#) представлена реализация выделения общей части кода с использованием динамического полиморфизма – эту реализацию рекомендуется использовать в качестве образца при выполнении домашней работы.

Оба варианта полиморфизма давались в курсе ООП [4] и не требуют подробного описания в данном методическом материале. Однако важно заметить, что оба варианта активно используют идиому RAII и другие важные приемы современной разработки, которые необходимо освоить. Ниже приводятся основные определения, используемые в указанных шаблонах.

Объект – эквивалентен понятию экземпляр класса.

Состояние объекта — совокупность значений (состояний) членов класса и состояний базовых классов.

Изменение состояния объектов — изменение значения (состояния) любого члена класса или состояния базового класса.

Некорректное / недопустимое состояние объекта — недопустимая комбинация значений (состояний) членов класса и / или состояний базовых классов.

Инвариант класса — утверждение, определяющее непротиворечивое состояние объектов этого класса.

Нарушение инварианта класса — объект класса имеет некорректное состояние.

Способы контроля инварианта класса — проверка инварианта в случаях, когда объект мог изменить своё состояние. Контроль инварианта класса в языке C++ осуществляется с помощью функции `assert`, описание которой находится в заголовочном файле `<cassert>`.

Важно заметить, что нарушение инварианта класса приводит к нарушению целостности выполнения кода, который использует данный класс, а значит той задачи, которую выполняет программа. Поэтому при нарушении инварианта, оправдано производить останов работы задачи, а если программа выполняет одну задачу, то этой программы, с выводом диагностического сообщения, что и делает функция `assert`.

Вариативность состояния класса – способность объекта заданного класса изменять своё состояние без нарушения инварианта при выполнении стандартных операций над объектами (копирование, параллельный доступ).

Неизменяемый (immutable) — объект никогда не изменяет своего состояния (для таких объектов инвариант достаточно проверить в конструкторе).

Копируемый — при копировании объект не нарушает своего состояния:

- при создании (конструктор копирования),
- при передаче в качестве параметра методу (конструктор копирования),
- при присваивании (оператор присваивания).

Некопируемый — объект может быть только переносим без нарушения своего состояния.

Идиома RAII («Resource Acquisition Is Initialization», бук. «получение ресурса есть инициализация») — техника использования механизма контроля зоны видимости объекта для управления ресурсами, которые он содержит.

2.2 Порядок выполнения работы

Для выполнения домашней работы необходимо у руководителя получить задачу, в которой описана «карточка задания» и выбрать шаблон, на базе которого будет выполнено задание. Выбранный шаблон необходимо изменить в соответствии с выданной «карточкой задания». Студент в праве самостоятельно решать каким образом описать типы данных на языке C++. В дальнейшем (при выполнении ЛР№3 и ЛР№4) реализация будет изменяться

(будет выполняться рефакторинг кода), поэтому рекомендуется делать максимально простую реализацию, но при этом аккуратно контролировать инварианты классов.

Необходимо сформулировать и реализовать метод определения инварианта класса для реализации «карточки задания».

Кроме кода, образец содержит настроенный механизм тестирования, который необходимо исправить в соответствии с заданием, добавив туда больше вариантов тестирования.

Работа должна быть выполнена с использованием вузовского репозитория и с применением механизма непрерывной интеграции (см. Приложение Б.).

Домашняя работа должна быть сохранена в вузовском репозитории. Сценарий непрерывной интеграции должен содержать тесты, проверяющие основные режимы работы программы, которые должны успешно выполняться.

Отчёт должен быть выполнен в файле описания проекта «README.md», предоставлять бумажный отчет не нужно.

2.3 Варианты заданий

Варианты заданий представляют собой описание «карточек задания» и приведены в таблице 3.

Таблица 3 – Варианты домашнего задания

№№	Наименование задачи	Описание карточки
1.	Задача: «Наблюдаемые суда»	<p>Карточка «судно» должна содержать следующие данные:</p> <ul style="list-style-type: none"> - наименование судна, - сухой тоннаж, - полный тоннаж, - тип (сухогруз, танкер, круизный лайнер).
2.	Задача: «Война в долине теней»	<p>Карточка «воин» должна содержать следующие данные:</p> <ul style="list-style-type: none"> - фракция, - сила удара, - защита, - здоровье, - ловкость, - уклонение, - тип боя.

№№	Наименование задачи	Описание карточки
3.	Задача: «Автозавод»	Карточка «машина» должна содержать следующие данные: <ul style="list-style-type: none"> - наименование модели, - номер сборочного конвейера, - код краски (перечисляемый тип), - код комплектации (перечисляемый тип)
4.	Задача: «Интернет-магазин»	Карточка «товар» должна содержать следующие данные: <ul style="list-style-type: none"> - наименование товара, - стоимость товара, - дата занесения в БД, - наличие товара.
5.	Задача: «Картинная галерея»	Карточка «картина» должна содержать следующие данные: <ul style="list-style-type: none"> - наименование картины, - автор, - год создания, - живописная техника (масло, графика, другое).
6.	Задача: «Воздушная обстановка»	Карточка «воздушное судно» должна содержать следующие данные: <ul style="list-style-type: none"> - номер борта, - тип воздушного судна (пассажирский, грузовой, военный), - размер воздушного судна (большой, средний, малый), - координаты (x, y, z).
7.	Задача: «Клубная лига»	Карточка «клуб» должна содержать следующие данные: <ul style="list-style-type: none"> - название клуба, - текущее место в лиге, - количество запасных, - ФИО тренера.
8.	Задача: «Экологическая обстановка»	Карточка «населённый пункт» должна содержать следующие данные: <ul style="list-style-type: none"> - название населенного пункта, - код в справочнике ОКАТО, - количество жителей,

№№	Наименование задачи	Описание карточки
		<ul style="list-style-type: none"> - уровень загрязнения (высокий, средний, низкий).
9.	Задача: «Карта галактики»	<p>Карточка «звезда» должна содержать следующие данные:</p> <ul style="list-style-type: none"> - номер по каталогу, - наименование (если есть), - дальность от Земли, - масса, - класс (O, B, A, F, G, K, M), - звёздная система.
10.	Задача: «Коллекция монет»	<p>Карточка «экспонат» должна содержать следующие данные:</p> <ul style="list-style-type: none"> - специальное наименование (если есть), - тип металла (перечисляемый тип), - наименование валюты, - количество единиц валюты, - количество монет в коллекции.
11.	Задача: «Музыкальная коллекция»	<p>Карточка «музыкальная запись» должна содержать следующие данные:</p> <ul style="list-style-type: none"> - название произведения, - автор, - исполнитель, - альбом, - длительность, - оценка (от 0 до 5).
12.	Задача: «Магазин музыкальных инструментов»	<p>Карточка «инструмент» должна содержать следующие данные:</p> <ul style="list-style-type: none"> - название инструмента, - производитель, - модель, - стоимость, - наличие.
13.	Задача: «Магазин автозапчастей»	<p>Карточка «запчасть» должна содержать следующие данные:</p> <ul style="list-style-type: none"> - название автозапчасти, - марка автомобиля, - модель автомобиля, - стоимость,

№№	Наименование задачи	Описание карточки
		- наличие.
14.	Задача: «Выборы в Муниципалитет»	Карточка «кандидат» должна содержать следующие данные: <ul style="list-style-type: none"> - ФИО кандидата, - возраст, - среднегодовой доход, - партия, - количество проголосовавших.
15.	Задача: «Букмекерская компания»	Карточка «ставка» должна содержать следующие данные: <ul style="list-style-type: none"> - номер ставки, - номер клиента, - коэффициент, - сумма ставки.
16.	Задача: «Ресторан быстрого питания»	Карточка «блюдо» должна содержать следующие данные: <ul style="list-style-type: none"> - название блюда, - стоимость, - количество калорий, - время приготовления, - популярность (от 0 до 5).
17.	Задача: «Курьерская служба»	Карточка «заказ» должна содержать следующие данные: <ul style="list-style-type: none"> - номер заказа, - ФИО курьера, - название товара, - адрес доставки, - стоимость доставки.
18.	Задача: «Магазин приложений»	Карточка «приложений» должна содержать следующие данные: <ul style="list-style-type: none"> - название приложения, - категория, - стоимость, - размер, - количество установок, - оценка (от 0 до 5).

№№	Наименование задачи	Описание карточки
19.	Задача: «Онлайн-школа программирования»	<p>Карточка «курс» должна содержать следующие данные:</p> <ul style="list-style-type: none"> - название курса, - продолжительность, - язык программирования, - сложность курса, - стоимость обучения.
20.	Задача: «Каталог библиотеки»	<p>Карточка «книга» должна содержать следующие данные:</p> <ul style="list-style-type: none"> - название произведения, - автор, - издательство, - год издания, - жанр.
21.	Задача: «Электронный журнал»	<p>Карточка «данные студента» должна содержать следующие данные:</p> <ul style="list-style-type: none"> - ФИО студента - ID студента - год поступления - кафедра - учёная степень (перечисляемый тип) - рейтинг (от 0 до 100)
22.	Задача: «Банк»	<p>Карточка «транзакция» должна содержать следующие данные:</p> <ul style="list-style-type: none"> - получатель - отправитель - сумма - код валюты - код транзакции - дата транзакции
23.	Задача: «Авиа-рейс»	<p>Карточка «бронь» должна содержать следующие данные:</p> <ul style="list-style-type: none"> - ФИО пассажира (если есть) - номер места - номер брони - статус (занято/свободно) - стоимость

№№	Наименование задачи	Описание карточки
		- тип места (эконом/бизнес/комфорт)
24.	Задача: «Онлайн-кинотеатр»	<p>Карточка «фильм» должна содержать следующие данные:</p> <ul style="list-style-type: none"> - название фильма - режиссёр - жанр - год - рейтинг (от 0 до 5) - длительность
25.	Задача: «Музыкальный фестиваль»	<p>Карточка «выступление» должна содержать следующие данные:</p> <ul style="list-style-type: none"> - название группы - жанр - порядок выступления - время (начало, конец)

2.4 Требования к отчету

Отчет должен быть составлен в виде текстового файла в формате Markdown с наименованием «README.md». Файл должен быть сохранён в репозитории в соответствующем проекте.

В отчет необходимо вставить функцию вычисления инварианта класса на языке C++, реализованного по «карточке задания».

2.5 Контрольные вопросы

После выполнения домашнего задания нужно уметь ответить на следующие контрольные вопросы:

- 1) Что такое состояние объекта?
- 2) Что означает изменение состояния объекта?
- 3) Дайте определение некорректного (недопустимого) состояния объекта.
- 4) Дайте определение инварианта класса.
- 5) Что такое вариативность состояния класса?
- 6) Дайте описание неизменяемого объекта.
- 7) Дайте описание не копируемого объекта, когда такие объекты встречаются?

- 8) Какие есть способы предотвратить копирование не копируемых объектов в языке C++?
- 9) Объясните идиому RAII.
- 10) Приведите примеры использования идиомы RAII.

3 Лабораторная работа №1. Использование встроенных в стандартную библиотеку C++ паттернов

Цель работы: приобрести навыки применения паттернов стандартной библиотеки C++, а также работы с внешним репозиторием кода и сценарием непрерывной интеграции.

Объем работы: 4 часа.

Студенты должны проанализировать задачу и предложить её решение с использованием заданного паттерна стандартной библиотеки C++. Приводят обоснования выбранного решения. Выполняют реализацию решения в виде программы, работающей в режиме командной строки, составленную на языке C++ с применением идиомы RAII. Результат работы сохраняют в вузовский репозиторий кода, добиваются прохождения сценария непрерывной интеграции.

3.1 Обобщенные контейнеры стандартной библиотеки C++

Библиотека контейнеров является универсальной коллекцией шаблонов классов и алгоритмов, позволяющих программистам легко реализовывать общие структуры данных, такие как очереди, списки и стеки. Существует три основных вида контейнеров: последовательные контейнеры, ассоциативные контейнеры, и неупорядоченные ассоциативные контейнеры, каждый из которых предназначен для поддержки различных наборов операций.

Контейнер управляет выделяемой для его элементов памятью и предоставляет функции-члены для доступа к ним, либо непосредственного, либо через итераторы (объекты, обладающие схожими с указателями свойствами).

Большинство контейнеров обладают по крайней мере несколькими общими функциями-членами и общей функциональностью. Выбор оптимального контейнера для конкретного случая зависит не только от предоставляемой функциональности, но и от его эффективности при различных рабочих нагрузках.

3.1.1 Последовательные контейнеры

Последовательные контейнеры реализуют структуры данных с возможностью последовательного доступа к ним:

array	Статический непрерывный массив
vector	Динамический непрерывный массив
deque	Двусторонняя очередь

forward_list Односвязный список

list Двусвязный список

3.1.2 Ассоциативные контейнеры

Ассоциативные контейнеры реализуют упорядоченные структуры данных с возможностью быстрого поиска (со сложностью $O(\log n)$):

set Коллекция уникальных ключей, отсортированная по ключам

map Коллекция пар ключ-значение, отсортированная по ключам, ключи являются уникальными

multiset Коллекция ключей, отсортированная по ключам

multimap Коллекция пар ключ-значение, отсортированная по ключам

р

3.1.3 Неупорядоченные ассоциативные контейнеры

Неупорядоченные ассоциативные контейнеры реализуют неупорядоченные (хешированные) структуры данных с возможностью быстрого поиска (со средней сложностью $O(1)$, в худшем случае $O(n)$):

unordered_set Коллекция уникальных ключей, хэш-ключами

unordered_map Коллекция пар ключ-значение, хэшированы ключи, ключи являются уникальными

unordered_multiset Коллекция ключей, хэш-ключами

t

unordered_multimap Коллекция пар ключ-значение, хэшируется по ключам

ар

Другие виды контейнеров, представленных в стандартной библиотеке C++ мы рассматривать не будем.

3.2 Умные указатели

Умные указатели предоставляют автоматическое управление временем жизни объекта, безопасное в случае возникновения исключений. При работе умных указателей используется идиома RAII. Умные указатели описаны в заголовочном файле `<memory>`.

Основные классы:

<code>unique_ptr</code>	Умный указатель единоличного владения объектом
<code>shared_ptr</code>	Умный указатель разделяемого владения объектом
<code>weak_ptr</code>	Слабая ссылка на объект, управляемый <code>std::shared_ptr</code>

3.3 Использование вузовского репозитория кода

Описание основных принципов работы с вузовским репозиторием кода, его настройки и работы приведено в подразделе 1.4 и приложении Б.

3.4 Порядок выполнения работы

Суть первой лабораторной работы заключается в написании консольной программы, для которой дается два задания:

- «карточка задания», по которой необходимо написать «класс карточки» (берется карточка из домашнего задания);
- комбинация контейнера и способа хранения в нём карточки, заданная мнемокодом из символов строки и графы из таблицы 4.

Студент должен исходя из заданных «карточки задания» и комбинации «контейнер-способ хранения» составить консольную программу, в которой проинициализировать заданный контейнер заданным способом хранения карточки и вывести её в стандартный поток.

На рисунке 1 приведён пример программы для хранения набора карточек `Person` в контейнере `array` без применения умных указателей. А на рисунке 2 – с применением класса `unique_ptr`.

```

#include <iostream>
#include <string>
#include <array>

using namespace std;

struct Person
{
    string first_name;
    string last_name;
    int year_of_birth;
};

int main()
{
    array<Person,3> a
    {
        Person {"Иван", "Иванов", 1992},
        Person {"Пётр", "Петров", 1993},
        Person {"Антон", "Антонов", 1990},
    };

    for(auto o : a)
        cout << o.first_name << ' ' << o.last_name << ' ' << o.year_of_birth << endl;

    return 0;
}

```

Рисунок 1. Пример реализации программы без применения умных указателей

```

#include <iostream>
#include <string>
#include <array>
#include <memory>

using namespace std;

struct Person
{
    string first_name;
    string last_name;
    int year_of_birth;
};

int main()
{
    array<unique_ptr<Person>,3> a
    {
        make_unique<Person>(Person {"Иван", "Иванов", 1992}),
        make_unique<Person>(Person {"Пётр", "Петров", 1993}),
        make_unique<Person>(Person {"Антон", "Антонов", 1990}),
    };

    for(const auto & o : a)
        cout << o->first_name << ' ' << o->last_name << ' ' << o->year_of_birth << endl;

    return 0;
}

```

Рисунок 2. Пример реализации программы с применением умного указателя `unique_ptr`

Лабораторная работа должна быть сохранена в вузовском репозитории. Сценарий непрерывной интеграции должен содержать тесты, проверяющие основные режимы работы программы, которые должны успешно выполняться.

Отчёт должен быть выполнен в файле описания проекта «README.md», предоставлять бумажный отчет не нужно.

3.5 Варианты заданий

Варианты «карточек задания» приведены в описании домашней работы, см. раздел 2.3. Необходимо узнать свой номер карточки для домашней работы.

Варианты комбинаций контейнера и способа хранения в нём карточки, заданная мнемокодом приведены в таблице 5.

Таблица 4 – Комбинация контейнера и способа хранения в нём карточки, заданная мнемокодом

Класс контейнера (первый символ мнемокода)	Способ хранения (второй символ мнемокода)		
	Без использования умных указателей (-)	unique_ptr (U)	shared_ptr (S)
vector (V)	V-	VU	VS
list (L)	L-	LU	LS
map (M)	M-	MU	MS
multimap (I)	I-	IU	IS

3.6 Требования к отчету

Отчет должен быть составлен в виде текстового файла в формате Markdown с наименованием «README.md». Файл должен быть сохранён в репозитории в соответствующем проекте.

3.7 Контрольные вопросы

После выполнения данной контрольной работы нужно уметь ответить на следующие контрольные вопросы:

- 1) Каковы основные особенности класса `std::vector`?
- 2) Каковы основные особенности класса `std::list`?
- 3) Каковы основные особенности класса `std::map`?
- 4) Каковы основные особенности класса `std::multimap`?
- 5) Каковы основные особенности класса `std::unordered_map`?

- 6) Каковы основные особенности класса `std::unordered_multimap`?
- 7) Каковы основные особенности класса `std::unique_ptr`?
- 8) Каковы основные особенности класса `std::shared_ptr`?
- 9) Объясните отличие классов `std::vector` и `std::list`
- 10) Объясните отличие классов `std::map` и `std::unordered_map`
- 11) Чем отличаются классы умных указателей `std::unique_ptr` и `std::shared_ptr`?
- 12) Почему необходимо использовать параметризованные функции `std::make_unique` и `std::make_shared`?

4 Лабораторная работа №2. Реализация типовых решений с применением шаблонов проектирования

Цель работы: приобрести навыки применения шаблонов проектирования для решения конкретной задачи, а также работы с внешним репозиторием кода и сценарием непрерывной интеграции.

Объем работы: 5 часов.

Студенты должны проанализировать задачу и предложить её решение с использованием заданного шаблона проектирования, выполняют реализацию решения в виде программы, работающей в режиме командной строки, составленной на языке C++. Результат работы сохраняют в вузовский репозиторий кода, добиваются прохождения сценария непрерывной интеграции.

4.1 Шаблоны проектирования

Шаблоны проектирования являются наиболее часто применяемыми приемами при проектировании и реализации сложных программных продуктов. Их применение позволяет добиться реализации устойчивого к изменениям кода.

Шаблоны проектирования используют парадигму ООП, а именно динамический полиморфизм (позднее связывание), поэтому могут применяться при написании программ на всех языках программирования, поддерживающих эту технику.

Шаблоны проектирования (паттерны проектирования) – повторяемая структура взаимодействия объектов программы, представляющая собой решение проблемы проектирования в рамках некоторого часто возникающего контекста [7, 8, 9].

Далее перечислены наиболее востребованные шаблоны проектирования.

1) Основные:

- *Delegation pattern* (делегирование) – объект внешне выражает некоторое поведение, но в реальности передаёт ответственность за выполнение этого поведения связанному объекту;
- *Functional design* (функциональный дизайн) – гарантирует, что каждый модуль компьютерной программы имеет только одну обязанность и исполняет её с минимумом побочных эффектов на другие части программы (см. SRP);
- *Immutable interface* (неизменяемый интерфейс) – создание неизменяемого объекта.

- *Interface* (интерфейс) – общий метод для структурирования компьютерных программ для того, чтобы разделять ответственность и их было проще понять (см. ISP);
- *Property container* (контейнер свойств) – позволяет добавлять дополнительные свойства для класса в контейнер (внутри класса), вместо расширения класса новыми свойствами.

2) Порождающие:

- *Abstract factory* (абстрактная фабрика) – класс, который представляет собой интерфейс для создания компонентов системы;
- *Builder* (строитель) – класс, который представляет собой интерфейс для создания сложного объекта;
- *Factory method* (фабричный метод) – определяет интерфейс для создания объекта, но оставляет подклассам решение о том, какой класс инстанцировать;
- *Lazy initialization* (отложенное инициирование) – объект, инициализируемый во время первого обращения к нему;
- *Prototype* (прототип) – определяет интерфейс создания объекта через клонирование другого объекта вместо создания через конструктор;
- *Singleton* (одиночка) – класс, который может иметь только один экземпляр;
- *Multiton* (мультистон) – гарантирует, что класс имеет поименованные экземпляры объекта и обеспечивает глобальную точку доступа к ним.

3) Структурные:

- *Adapter / Wrapper* (адаптер) – объект, обеспечивающий взаимодействие двух других объектов, один из которых использует, а другой предоставляет несовместимый с первым интерфейс;
- *Bridge* (мост) – объект, используемый в проектировании программного обеспечения чтобы «разделять абстракцию и реализацию так, чтобы они могли изменяться независимо». Шаблон мост использует инкапсуляцию, агрегирование и может использовать наследование для того, чтобы разделить ответственность между классами.
- *Composite* (компоновщик) – объект, который объединяет в себе объекты, подобные ему самому;
- *Decorator* (декоратор) – класс, расширяющий функциональность другого класса без использования наследования;
- *Facade* (фасад) – объект, который абстрагирует работу с несколькими классами, объединяя их в единое целое;
- *Flyweight* (приспособленец) – это объект, представляющий себя как уникальный экземпляр в разных местах программы, но фактически не являющийся таковым;

- *Proxy* (заместитель) – объект, который является посредником между двумя другими объектами, и который реализует/ограничивает доступ к объекту, к которому обращаются через него.

4) Поведенческие:

- *Chain of responsibility* (цепочка обязанностей) – предназначен для организации в системе уровней ответственности;
- *Command* (команда) – представляет действие. Объект команды включает в себя само действие и его параметры;
- *Interpreter* (интерпретатор) – решает часто встречающуюся, но подверженную изменениям, задачу;
- *Iterator* (итератор) – представляет собой объект, позволяющий получить последовательный доступ к элементам объекта-агрегата без использования описаний каждого из объектов, входящих в состав агрегации;
- *Mediator* (посредник) – обеспечивает взаимодействие множества объектов, формируя при этом слабую связанность и избавляя объекты от необходимости явно ссылаться друг на друга;
- *Memento* (хранитель) – позволяет, не нарушая инкапсуляцию зафиксировать и сохранить внутренние состояния объекта так, чтобы позднее восстановить его в этих состояниях;
- *Null Object* (пустой объект) – предотвращает нулевые указатели, предоставляя объект «по умолчанию»;
- *Observer* (наблюдатель) – реализует у класса механизм, который позволяет объекту этого класса получать оповещения об изменении состояния других объектов и тем самым наблюдать за ними;
- *Servant* (слуга) – используется для обеспечения общей функциональности группе классов;
- *Specification* (спецификация) – служит для связывания бизнес-логики;
- *State* (состояние) – используется в тех случаях, когда во время выполнения программы объект должен менять своё поведение в зависимости от своего состояния;
- *Strategy* (стратегия) – предназначен для определения семейства алгоритмов, инкапсуляции каждого из них и обеспечения их взаимозаменяемости;
- *Template method* (шаблонный метод) – определяет основу алгоритма и позволяет наследникам переопределять некоторые шаги алгоритма, не изменяя его структуру в целом;
- *Visitor* (посетитель) – описывает операцию, которая выполняется над объектами других классов. При изменении класса *Visitor* нет необходимости изменять обслуживаемые классы.

4.2 Порядок выполнения работы

При выполнении задания ЛР №2 необходимо проявить гибкость, не пытаться точно использовать примеры кода, приведённые в методических указаниях.

Работа состоит из следующих этапов:

- 1) Получение задание на выполнение ЛР№2.
- 2) Реализация программы командной строки на языке C++, реализующей полученное задание.
- 3) Сдача отчета.

В качестве примера рассмотрим следующее задание:

Шаблон проектирования Interface (интерфейс).

Разработать программу командной строки, в которой вывод содержимого коллекции карточек Person осуществлялся бы через задаваемый интерфейс с применением шаблона проектирования Interface (интерфейс). Итоговый вариант программы должен быть сохранён в вузовский репозиторий, скомпилирован и проверен с помощью сценария непрерывной интеграции.

Шаблон проектирования «интерфейс» подходит, например, для реализации вывода сообщений в заранее неизвестный вид пользовательского интерфейса. Для этого реализуем интерфейс IReporter, поведение которого заключается в выводе информации об объекте наследуемого класса (см. рисунок 3).

```
#include <iostream>
#include <string>
#include <array>
#include <vector>
#include <exception>
#include <stdexcept>

using namespace std;

class IReporter
{
public:
    virtual void output(string output_string) = 0;
};
```

Рисунок 3. Начало текста программы с определением интерфейса

Затем реализуем класс, владеющий коллекцией карточек Person (см рисунок 4).


```

class PersonCollector
{
    struct Person
    {
        string  first_name;
        string  last_name;
        int     year_of_birth;
    };

    IReporter & _reporter;
    array<Person,3> _persons;

public:
    PersonCollector() = delete;
    PersonCollector(IReporter & reporter)
        : _reporter(reporter)
        , _persons({
            Person {"Иван", "Иванов", 1992},
            Person {"Пётр", "Петров", 1993},
            Person {"Антон", "Антонов", 1990},
        })
    {}

    void outputByIndex(int index)
    {
        if (index < 0 || index >= _persons.size())
            throw range_error("Недопустимое значение индекса");

        _reporter.output(_persons[index].first_name + " " +
            _persons[index].last_name + ", " +
            to_string(_persons[index].year_of_birth));
    }

    void outputAll()
    {
        for(int i=0; i < _persons.size(); ++i)
            outputByIndex(i);
    }
};

```

Рисунок 4. Класс владелец коллекцией карточек Person

Затем записываем реализацию интерфейса IReporter для вывода информации о содержимом на консоль (см. рисунок 5).

```

class CoutReporter : public IReporter
{
public:
    virtual void output(string output_string) override
    {
        cout << output_string << endl;
    }
};

```

Рисунок 5. Реализация интерфейса для вывода информации о содержимом на консоль

Наконец, можно записать функцию main (см. рисунок 6).

```

int main(int argc, char ** argv)
{
    try
    {
        CoutReporter    reporterToCout;
        PersonCollector collector(reporterToCout);

        vector<string> arguments(argv + 1, argv + argc);

        if (arguments.empty())
        {
            collector.outputAll();
            return 0;
        }

        if (arguments.size() == 1)
        {
            int i = stoi(arguments[0]);

            collector.outputByIndex(i);

            return 0;
        }

        cout << "*** Недопустимое количество аргументов" << endl;
        return 1;
    }
    catch(exception &e)
    {
        cout << "*** " << e.what() << endl;
        return 1;
    }
}

```

Рисунок 6. Реализация функции main

Таким образом, с помощью шаблона проектирования «интерфейс» удалось добиться отделения реализации конкретного вывода информации об содержимом от устройства вывода. Код контейнера, который использует интерфейс IReporter не связан с конкретным устройством вывода.

Лабораторная работа должна быть сохранена в вузовском репозитории. Сценарий непрерывной интеграции должен содержать тесты, проверяющие основные режимы работы программы, которые должны успешно выполняться.

Отчёт должен быть выполнен в файле описания проекта «README.md», предоставлять бумажный отчет не нужно.

4.3 Варианты заданий

Варианты заданий для выполнения лабораторной работы №2 приведены в таблице 5.

Таблица 5 – Варианты заданий для выполнения лабораторной работы №2

№№	Задание
1.	Реализуйте карточку вашего домашнего задания с использованием шаблона проектирования <i>Property container</i> (контейнер свойств). С использованием параметров командной строки выполните наполнение карточки (единственной карточки, работа с

№№	Задание
	контейнером карточек не требуется) и осуществите вывод полей карточки в стандартный поток.
2.	С использованием шаблона проектирования <i>Builder</i> (строитель) реализуйте заполнение коллекции карточек вашего домашнего задания и вывода содержимого в стандартный поток.
3.	С использованием шаблона проектирования <i>Factory method</i> (фабричный метод) выполните создание карточки из вашего домашнего задания и поместите ее в контейнер. Выполните вывод содержимого контейнера в стандартный поток.
4.	С использованием шаблона проектирования <i>Prototype</i> (прототип) выполните создание карточки из вашего домашнего задания и поместите ее в контейнер. Выполните вывод содержимого контейнера в стандартный поток.
5.	С использованием шаблона проектирования <i>Singleton</i> (одиночка) выполните создание карточки из вашего домашнего задания и поместите ее в контейнер. Выполните вывод содержимого контейнера в стандартный поток.
6.	Создайте программу журналирования всех действий с карточкой вашего домашнего задания с использованием шаблона проектирования <i>Bridge</i> (мост) так, чтобы конкретная реализация интерфейса журналирования могла быть изменена на другую без изменения кода в классе карточки. Выполните единственную реализацию журналирования в стандартный поток.
7.	С использованием шаблона проектирования <i>Composite</i> (компоновщик) выполните создание иерархии произвольных карточек (в качестве образца для примера возьмите карточку из вашего домашнего задания). Выполните вывод иерархии в стандартный поток.
8.	С использованием шаблона проектирования <i>Decorator</i> (декоратор) выполните создание объекта-декоратора карточки из вашего домашнего задания, не изменяя ее саму, добавив возможность вывода какого-либо признака (например, признака принадлежности этой карточки к особой коллекции, ее редкости, особой комбинации ее свойств и т.д.). Выполните вывод содержимого контейнера в стандартный поток с учетом выполненной декорации.
9.	С использованием шаблона проектирования <i>Proxu</i> (заместитель) реализуйте журналирование любых операций с карточкой в стандартный поток.
10.	С использованием шаблона проектирования <i>Command</i> (команда) реализуйте обработку команд из командной строки приложения. Команды должны быть следующие: добавление карточки вашего домашнего задания в контейнер, вывод содержимого контейнера в стандартный поток.
11.	С использованием шаблона проектирования <i>Null Object</i> (пустой объект) реализуйте

№№	Задание
	отображение карточки вашего домашнего задания, которые хранятся в ассоциативном массиве, если заданный через командную строку ключ не найден в контейнере. Результаты поиска нужно вывести в стандартный поток. Контейнер можно наполнить в коде программы.

4.4 Требования к отчету

Отчет должен быть составлен в виде текстового файла в формате Markdown с наименованием «README.md». Файл должен быть сохранён в репозитории в соответствующем проекте.

Помимо общих требований, отчет должен содержать UML-диаграмму классов программы, которую рекомендуется взять из автоматически сгенерированного описания классов, а также необходимо вставить в отчёт фрагмент кода с функцией `main`.

4.5 Контрольные вопросы

После выполнения данной контрольной работы нужно уметь ответить на следующие контрольные вопросы:

- 1) Дайте краткое описание шаблона проектирования *Property container* (контейнер свойств).
- 2) Дайте краткое описание шаблона проектирования *Builder* (строитель).
- 3) Дайте краткое описание шаблона проектирования *Factory method* (фабричный метод).
- 4) Дайте краткое описание шаблона проектирования *Prototype* (прототип).
- 5) Дайте краткое описание шаблона проектирования *Singleton* (одиночка).
- 6) Дайте краткое описание шаблона проектирования *Bridge* (мост).
- 7) Дайте краткое описание шаблона проектирования *Composite* (компоновщик).
- 8) Дайте краткое описание шаблона проектирования *Decorator* (декоратор).
- 9) Дайте краткое описание шаблона проектирования *Proxy* (заместитель).
- 10) Дайте краткое описание шаблона проектирования *Command* (команда).
- 11) Дайте краткое описание шаблона проектирования *Null Object* (пустой объект).
- 12) Какие преимущества предоставляет использование примененного шаблона проектирования?
- 13) Какие недостатки имеет примененный шаблон проектирования?

- 14) В каких случаях целесообразно применение заданного шаблона проектирования?
- 15) Назовите шаблоны проектирования, которые может быть целесообразно использовать совместно с заданным.
- 16) Какие вы знаете примеры использования заданного шаблона проектирования?

5 Лабораторная работа №3. Описание модели, изоляция предметной области

Цель работы: приобрести навыки описания модели предметной области, выполнения ее изоляции, а также работы с внешним репозиторием кода и сценарием непрерывной интеграции.

Объем работы: 4 часа.

Студенты должны составить словарь заданной предметной области, отобразить модель предметной области в виде UML-диаграммы, выделить уровни реализации модели с учетом принципа изоляции предметной области. Результат работы сохраняют в вузовский репозиторий кода, добиваются прохождения сценария непрерывной интеграции.

5.1 Изоляция предметной области

Предметно-ориентированное проектирование (Domain-driven design, **DDD**) – это набор принципов и схем, направленных на создание оптимальных систем объектов. DDD сводится к созданию программных абстракций, которые называются моделями предметных областей и обладает рядом плюсов:

- позволяет автоматизировать незнакомые разработчикам предметные области;
- позволяет вести разработку итерационно, постепенно усложняя ПО;
- позволяет значительно ускорить разработку сложного ПО.

DDD в свое время собрало наиболее успешные принципы и техники проектирования программного обеспечения в методологиях гибкой разработки (Agile) и стало основой для многих других техник проектирования.

Приведем несколько ключевых определений:

Область (Domain) — предметная область, к которой применяется разрабатываемое программное обеспечение.

Язык описания — используется для единого описания модели предметной области.

Модель (Model) — описывает конкретную предметную область или её часть, является базой для автоматизации.

Описание модели предметной области в DDD сводится к построению UML-диаграмм и затем реализации их в коде программы. Код модели должен отражать модель предметной области. Изменение модели влечет изменение кода и, наоборот, изменение кода означает изменение модели.

Изоляция предметной области необходима для отделения кода модели от вспомогательных конструкций, что позволяет выполнять гибкую и независимую модификацию участков кода, отвечающих за принципиально разный функционал (см. SRP SOLID или ШП

Functional design), вплоть до полной замены одних модулей без радикального изменения других.

Изоляция предметной области как правило выполняется с использованием многоуровневой архитектуры, которая имеет следующий канонический состав уровней [10]:

- 1) **Интерфейс пользователя** (уровень представления) – UI. Отвечает за вывод информации пользователю и интерпретацию команд пользователя.
- 2) **Операционный уровень** (уровень прикладных операций, уровень приложения) – AL. Определяет задачи, связанные с конкретным действием в UI (команда пользователя или потребность в информации) и распределяет их между объектами предметной области. Не хранит состояний объектов предметной области. Может играть интегрирующую роль – взаимодействовать с операционными уровнями других систем. Наиболее близкий ШП: Fasade.
- 3) **Уровень предметной области** (уровень модели, уровень бизнес-логики) – DL. Отвечает за представление понятий прикладной предметной области, рабочие состояния, бизнес-регламенты (поведение модели). Этот уровень является главной, алгоритмической частью программы.
- 4) **Инфраструктурный уровень** (уровень доступа к данным) – IL. Слой технических сервисов. Обеспечивает техническую поддержку для верхних уровней: передачу сообщений на операционном уровне, непрерывность существования объектов на уровне модели (хранение, транзакционность и т.д.), службы передачи сообщений, почтовые службы и т.д.

Каждый уровень зависит только от нижележащего слоя и может существовать без вышерасположенных слоёв (см. рисунок 7).

Для связи с верхними уровнями используются:

- обратные вызовы (callback);
- ШП Observer;
- стандартные архитектурные шаблоны:
 - Model-View-Controller (MVC),
 - Model-View-Presenter,
 - Naked objects,
 - и др.

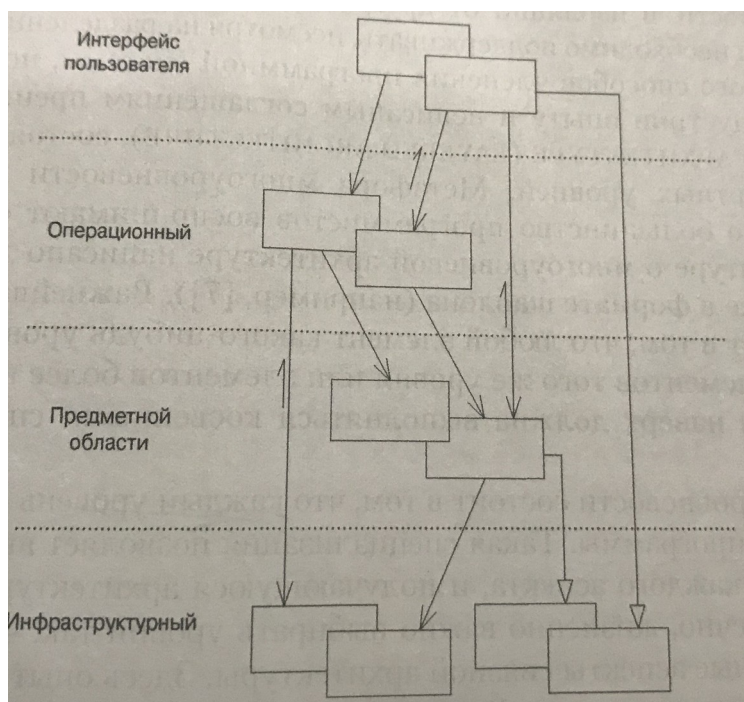


Рисунок 7 – Условная схема взаимодействия объектов программы, расположенных на разных уровнях

5.2 Порядок выполнения работы

При выполнении лабораторной работы необходимо взять за основу выполненное к этому моменту домашнее задание. Перед выполнением лабораторной работы нужно поставить тэг, отделяющий ваше домашнее задание от ЛР с помощью команды:

```
git tag -a homework -m 'Домашняя работа'
```

Суть лабораторной работы заключается в разделении исходного кода домашнего задания на модули C++ в соответствии со слоями многоуровневой архитектуры:

- 1) Модуль «**l1_UserInterface.cpp**» (без соответствующего заголовочного файла), отвечающий за реализацию интерфейса с пользователем.
- 2) Модуль «**l2_ApplicationLayer.cpp**» (заголовочный файл «**l2_ApplicationLayer.h**»), реализующий операционный уровень.
- 3) Модуль «**l3_DomainLayer.cpp**» (заголовочный файл «**l3_DomainLayer.h**»), реализующий уровень модели предметной области.
- 4) Модуль «**l4_InfrastructureLayer.cpp**» (заголовочный файл «**l4_InfrastructureLayer.h**»), реализующий инфраструктурный уровень.

Модули C++ должны располагаться в каталоге «src», а их заголовочные файлы (если есть) – в каталоге «include/hw».

Проект должен содержать следующие каталоги:

- **bin** – для хранения исполняемого файла (не сохраняется в репозитории, создается командным файлом «configure»);
- **build** – для хранения объектных файлов модулей программы (не сохраняется в репозитории, создается командным файлом «configure»);
- **doc** – для хранения конфигурационного файла «Doxyfile.cfg» и временных файлов сгенерированной документации, а также изображений;
- **include/hw** – для хранения заголовочных файлов;
- **src** – для хранения кода модулей на C++;
- **test** – для хранения автотестов (не изменяется по сравнению с выполненной ДР).

В корневом каталоге проекта должны находиться следующие файлы:

- **.gitignore** – список исключений для git;
- **.gitlab-ci.yml** – описание сценария CI;
- **configure** – командный файл, выполняющий первоначальную настройку проекта;
- **documentation** – командный файл, формирующий документацию по исходному коду проекта в каталоге doc/html;
- **hw.pro** – проектный файл для Qt Creator (не обязательный);
- **Makefile** – управляющий файл для сборки проекта с использованием утилиты make;
- **README.md** – файл описания проекта.

Для выполнения сборки проекта используется утилита make, которую нужно установить на компьютер, если она не установлена. Для выполнения сборки проекта нужно выполнить команду make из рабочего каталога проекта. Для выполнения автотестов нужно выполнить команды test/test01 и test/test02.

Диаграммы классов, описывающие модель предметной области, можно взять из генерируемой документации.

Также необходимо взять вариант задания и нарисовать диаграмму последовательности выполнения команды, изобразив прохождение выполнения команды между программными объектами, расположенными в соответствующих слоях многоуровневой архитектуры. Скан этого рисунка нужно разместить в файле описания проекта «README.md». Образец можно посмотреть в заготовке проекта в вузовском репозитории.

Лабораторная работа должна быть сохранена в вузовском репозитории. Сценарий непрерывной интеграции должен содержать тесты, проверяющие основные режимы работы программы, которые должны успешно выполняться.

Отчёт должен быть выполнен в файле описания проекта «README.md», предоставлять бумажный отчет не нужно.

5.3 Варианты заданий

Варианты заданий для выполнения лабораторной работы №3 приведены в таблице 8.

Таблица 6 – Варианты заданий для выполнения лабораторной работы №3

№№	Задание
1.	Постройте диаграмму последовательности выполнения команды save . Диаграмма должна показывать прохождение выполнения команды между программными объектами, расположенными в соответствующих слоях многоуровневой архитектуры.
2.	Постройте диаграмму последовательности выполнения команды clean . Диаграмма должна показывать прохождение выполнения команды между программными объектами, расположенными в соответствующих слоях многоуровневой архитектуры.
3.	Постройте диаграмму последовательности выполнения команды add . Диаграмма должна показывать прохождение выполнения команды между программными объектами, расположенными в соответствующих слоях многоуровневой архитектуры.
4.	Постройте диаграмму последовательности выполнения команды remove . Диаграмма должна показывать прохождение выполнения команды между программными объектами, расположенными в соответствующих слоях многоуровневой архитектуры.
5.	Постройте диаграмму последовательности выполнения команды update . Диаграмма должна показывать прохождение выполнения команды между программными объектами, расположенными в соответствующих слоях многоуровневой архитектуры.
6.	Постройте диаграмму последовательности выполнения команды view . Диаграмма должна показывать прохождение выполнения команды между программными объектами, расположенными в соответствующих слоях многоуровневой архитектуры.

5.4 Требования к отчету

Отчет должен быть составлен в виде текстового файла в формате Markdown с наименованием «README.md». Файл должен быть сохранён в репозитории в соответствующем проекте.

Помимо общих требований, отчет должен содержать диаграммы классов проекта, а также диаграмму последовательности по полученному заданию.

5.5 Контрольные вопросы

После выполнения данной контрольной работы нужно уметь ответить на следующие контрольные вопросы:

- 1) Дайте определение изоляции предметной области?
- 2) Каким образом выполняется изоляция предметной области?
- 3) Какие преимущества достигаются при изоляции предметной области?
- 4) Назовите слои многоуровневой архитектуры.
- 5) Опишите назначение уровня пользовательского интерфейса.
- 6) Опишите назначение операционного уровня.
- 7) Опишите назначение уровня предметной области.
- 8) Опишите назначение инфраструктурного уровня.

6 Лабораторная работа №4. Рефакторинг. Выделение сущностей, значений и служб модели, изоляция ограниченного контекста

Цель работы: приобрести навыки оценки необходимых изменений при уточнении модели предметной области в рамках ограниченного контекста с учётом его изоляции, а также работы с внешним репозиторием кода и сценарием непрерывной интеграции.

Объем работы: 4 часа.

Студенты должны составить план рефакторинга программы после анализа изменений в модели предметной области и выделить сущности, значения, службы и агрегаты, а также показать ограниченный контекст и способы его изоляции. Результат работы сохраняют в вузовский репозиторий кода, добиваются прохождения сценария непрерывной интеграции.

6.1 MDD, структура модели, углубляющий рефакторинг

Техника DDD чаще всего и наиболее эффективно применяется при гибких методологиях разработки, когда не только осознанно допускается, но и прямо декларируется несоответствие модели и предметной области. Это понимание основано на том, что, во-первых, в процессе автоматизации незнакомой предметной области длительное время невозможно точно отразить предметную область в модели – она просто неизвестна. Это отражение приходится делать постепенно, итерационно, постоянно консультируясь с представителями заказчика, показывая им и обсуждая результаты каждого цикла и выполняя пробные развёртывание и эксплуатацию.

Во-вторых, даже если предметная область в целом знакома, но есть другие риски, связанные с успехом автоматизации, целесообразно вести разработку опять же итерационно, в первую очередь реализуя главный функционал, наиболее востребованный заказчиком и выполнять пробные развёртывание и эксплуатацию.

Такой подход определяет важность соблюдения соответствия модели и кода программы, когда стремятся минимизировать временной лаг между разработкой модели, реализуемой в данной итерации функционального наполнения и кодом программы, что позволяет утверждать, что код программы и является моделью предметной области. Этот подход предлагается техникой проектирования по модели, которая является основой DDD.

Проектирование по модели (model-driven design, **MDD**) – процесс совместной разработки модели и её программной реализации с сохранением максимально близкого соответствия между ними [10].

Построение модели обычно выполняется в следующей ниже последовательности.

- 1) Выделение **понятий** предметной области, участвующих в реализуемом функциональном составе программы (применяется обычная техника абстрагирования ООП).
- 2) Построение **ассоциаций** между понятиями предметной области и их максимальное упрощение. Упрощение ассоциаций сводится к следующим правилам:
 - а) сведение связей к неравноправным (направленным);
 - б) определение кратности связей, устранение связей «многие-ко-многим»;
 - в) минимизация кратности за счёт ограничений ассоциации (квалификатором ассоциаций).
- 3) Выделение **сущностных понятий** предметной области – объектов (понятий) предметной области, характеризующихся непрерывностью и индивидуальностью (уникальностью) существования.
- 4) Выделение **объектов-значений** – понятий, которые представляют описательный аспект предметной области и не имеют индивидуального существования, собственной идентичности.
- 5) Выделение **служб** – операций или групп операций, предлагаемых в модели в виде обособленного интерфейса, который в отличие от сущности или объекта-значения не имеет никакого состояния.
- 6) Разделение модели на **модули** – устоявшихся элементов архитектуры программы.
- 7) Объединение объектов модели (и программы) в **агрегаты** – совокупность взаимосвязанных объектов, которые мы воспринимаем как единое целое с точки зрения изменения данных. Понятие агрегата очень важно в проектировании по модели, так как выделяет из набора понятий корневую (главную) сущность, которая позволяет управлять транзакционностью, контролем инварианта агрегата, выполнять создание целостной совокупности объектов и хранением её состояния. При работе с агрегатами следует придерживаться следующих правил:
 - а) корневой объект-сущность имеет глобальную идентичность и несёт полную ответственность за проверку инвариантов;
 - б) некоренные объекты-сущности имеют локальную идентичность – они уникальны только в границах агрегата;
 - в) нигде за пределами агрегата не может храниться ссылка на что-либо внутри него (только временные ссылки и значения);
 - г) значит, только корневые объекты агрегатов можно непосредственно получать по запросам из БД, все остальные объекты разрешается извлекать по цепочке связей;
 - д) объекты внутри агрегата могут хранить ссылки на корневые объекты других агрегатов;
 - е) операция удаления должна ликвидировать всё, что находится в границах агрегата;

ж) как только вносится изменение в любой объект агрегата, следует сразу удовлетворять все инварианты этого агрегата.

Перечисленные действия обычно выполняются на каждой итерации разработки, когда изменяется модель предметной области. При этом часто сначала выполняется рефакторинг существующей структуры модели, после которого прогоняются регрессионные тесты, а затем осуществляется добавление или изменение функциональности.

Рефакторинг – это такая реструктуризация программы, в результате которой не изменяются её функциональные возможности [10]. Рефакторинг выполняется, если необходимо сделать модель объектов проще, понятнее или как предварительный этап перед добавлением новой функциональности. Важным является то, что после рефакторинга регрессионные тесты должны выполняться без изменений.

При **углубляющем рефакторинге** модель изменяется с учётом дополнительной функциональности при этом стараются изменить структуру так, чтобы она не требовала изменений функциональных тестов, но иногда такое допускают в очень малом объёме.

Ограниченный контекст – область применения конкретной модели с определёнными границами. Ограниченные контексты дают членам группы разработки чёткое и общее представление о том, где следует соблюдать согласованность, а где можно работать независимо. В результате группы разработки могут разделять VCS и CI, а значит работать в разных темпах. Ограниченный контекст обеспечивает четко определенные безопасные гавани, позволяя моделям усложняться, не жертвуя концептуальной целостностью.

Отношения между ограниченными контекстами (а значит и между их командами разработки) могут быть перечисленных ниже видов в порядке уменьшения зависимости.

Партнёрство (partnership). Команды в двух контекстах работают вместе (достигают успеха или терпят неудачу вместе). Эти команды устанавливают процесс координированного планирования разработки и совместное управление интеграцией. Они должны сотрудничать в процессе эволюции своих интерфейсов, чтобы учитывать потребности обеих систем. Взаимозависимые функции должны быть организованы так, чтобы завершение их разработки происходило в рамках одного выпуска.

Общее ядро (shared kernel). Общая часть моделей и связанные с ней объекты образуют очень тесную взаимосвязь, которая может как способствовать работе, так и мешать ей. Обозначьте чёткую границу моделей предметной области, которую команды согласны считать общей. Ядро должно быть маленьким. Оно должно иметь общий статус и не изменяться без консультации с другой командой. Установите непрерывный интеграционный процесс и согласуйте единый язык команд.

Разработка «заказчик-поставщик» (customer-supplier development). Две команды находятся в отношениях «заказчик-поставщик» (down-up), причём успех вышестоящей команды зависит от нижестоящей (аналогия с течением реки: поставщик «сплавляет» ответы заказчику). Следует учитывать приоритеты команд «заказчиков» при планировании работы «поставщиков». Проводите переговоры и согласовывайте планы, учитывающие потребности команд «заказчиков».

Конформист (conformist). Команда-поставщик (up) по каким-то причинам не может удовлетворить потребности «заказчика» и предоставить свою модель для работы. Создайте слой объектов, имитирующих модель «поставщика».

Предохранительный уровень (anticorruption layer) – ACL. Команда «поставщик» по каким-то причинам не может обеспечить неизменный интерфейс своего модуля. Создайте трансляционный уровень (набор фасадных объектов), предназначенных для быстрой настройки взаимодействия.

Служба с открытым протоколом (open host service) – OHS. Модель является «поставщиком» для многих заказчиков. Определите протокол, предоставляющий доступ к вашей системе, как к набору служб. Откройте этот протокол для всех, кто захочет интегрироваться с вами. Этот шаблон можно реализовать в виде REST-ресурса, с которым взаимодействует клиент из мира ограниченных контекстов проекта. Обычно службу с открытым протоколом представляют в виде RPC из интерфейса прикладного программирования, но её можно реализовать через механизм сообщений.

Общедоступный язык (published language) – PL. Взаимодействие между моделями требует общий язык. Используйте в качестве средства коммуникации хорошо документированный общий язык, который может выразить необходимую информацию о предметной области, выполняя при необходимости перевод информации между другими языками. Общедоступный язык часто сочетается со службой с открытым протоколом. Его можно реализовать несколькими способами, но часто его представляют в виде XML-схемы (XML Schema, зарезервировано расширение файлов XSD), что позволяет использовать, например, XSLT для трансляции между контекстами. В последнее время популярной становится JSON-схема (JSON Schema) [11].

Большой комок грязи (big ball of mud). Выполняя анализ предметной области или существующей программной реализации предметной области, мы можем обнаружить, что существуют модули, в которых модели перемешаны, а границы стёрты. Нарисуйте границы вокруг такой смеси и обозначьте её как «большой комок грязи».

Карта контекстов позволяет наглядно представлять взаимодействие контекстов в графическом виде. На рисунке 8 показан пример карты контекстов, на которой отображены некоторые виды отношений между контекстами.

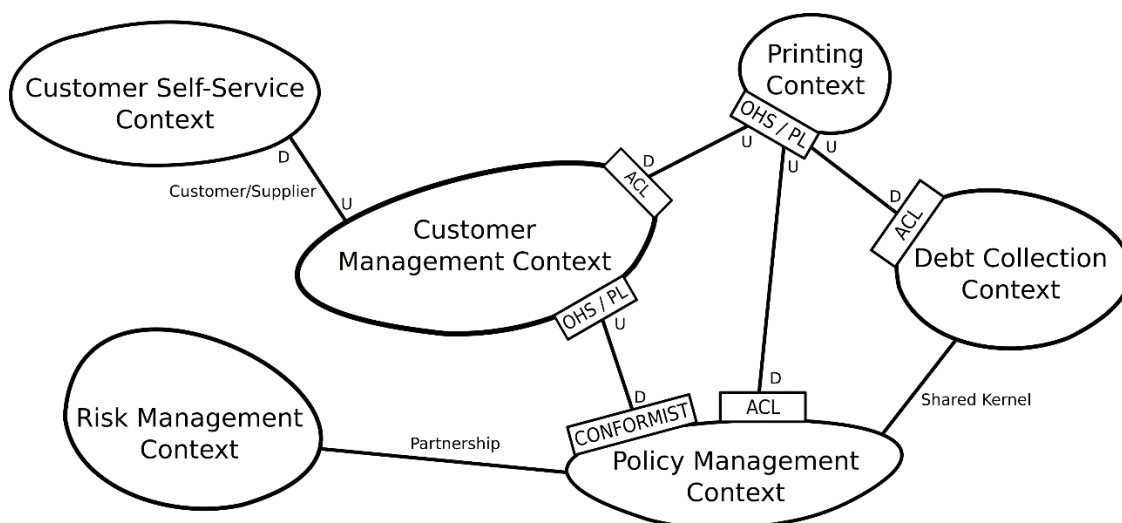


Рисунок 8. Пример карты контекстов²

Разделение модели предметной области на ограниченные контексты и определение отношений между ними часто используется при определении структуры проекта в микросервисной архитектуре.

6.2 Порядок выполнения работы

Предыдущий практикум (лабораторная работа №3) был посвящён выделению и изоляции предметной области, то есть модели, и это позволит упростить и ускорить разработку кода модели. При выполнении лабораторной работы №4 необходимо взять за основу программу, составленную при выполнении лабораторной работы №3. Перед выполнением лабораторной работы нужно поставить тэг, отделяющий вашу ЛРН№3 с помощью команды:

```
git tag -a lab3 -m 'ЛРН№3'
```

Далее нужно самостоятельно придумать или взять вариант, приведённый в подразделе , усложнения домашнего задания. Текст придуманного или взятого условия задания нужно отразить в отчёте (или в файле README.md, если выполняется СИ-вариант ЛР).

Далее нужно построить диаграмму объектов модели предметной области, выделить на ней сущности, объекты-значения и службы, а также показать агрегат, обведя объекты, которые в него входят, и выделить главный объект-сущность, который должен выступать вовне из обведённых границ агрегата. Диаграмму желательно нарисовать на листе, который затем прикрепить к отчёту (или отсканировать и вставить в файле README.md, если выполняется СИ-вариант ЛР), см. рисунок 9.

² Пример изображения карты контекстов взят из <https://github.com/ContextMapper/contextmapper.github.io>.

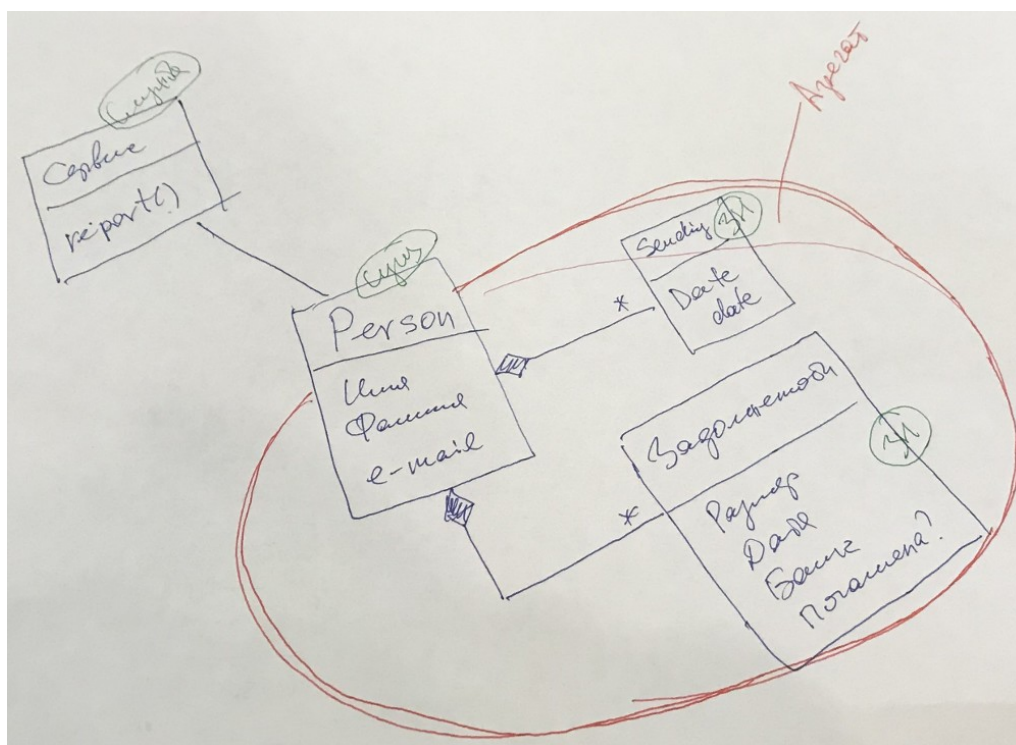


Рисунок 9. Пример диаграммы модели с помеченными сущностями, объектами-значениями, службами и выделенным агрегатом с корневым объектом-сущностью.

Затем на базе выполненной ранее лабораторной работы №3 нужно реализовать:

- рефакторинг, отражающий изменения в модели,
- убедиться, что он не повлиял на старую функциональность,
- добавить новую функциональность, доработать автотесты и
- сохранить в репозитории.

Если при построении модели выявляется несколько сущностей и/или несколько агрегатов, то нужно выбрать для реализации только один из них, а оставшиеся вынести в другой ограниченный контекст. При этом нужно указать в отчёте какое отношение между контекстами вы использовали.

Лабораторная работа должна быть сохранена в вузовском репозитории. Сценарий непрерывной интеграции должен содержать тесты, проверяющие основные режимы работы программы, которые должны успешно выполняться.

Отчёт должен быть выполнен в файле описания проекта «README.md», предоставлять бумажный отчет не нужно.

6.3 Варианты заданий

Варианты заданий для выполнения лабораторной работы №4 можно придумать самостоятельно на базе выданного ранее варианта домашнего задания (ваш вариант должен предполагать невырожденный случай агрегата) или взять в таблице 7.

Таблица 7 – Варианты заданий для выполнения лабораторной работы №4

№№ ДЗ	Наименование задачи / карточки	Задание для ЛРН ₄
1.	«Наблюдаемые суда» / «Морское судно»	Необходимо определить три морских судна, находящихся на минимальном прямом расстоянии от другого судна, которое терпит бедствие и посылает сигналы SOS.
2.	«Война в долине теней» / «Воин Долины теней»	Ваш отряд встречает ужасного тролля, у которого здоровье равно сумме здоровья всех ваших бойцов, сила удара и защита равны силе удара и защите самого сильного и защищённого из бойцов, соответственно, а все остальные характеристики равны минимальной из всего вашего отряда. Как победить тролля с минимальными потерями? В битве урон здоровью после удара вычисляется по формуле: (1) $Урон_1 = Защита_1 + Уклонение_1 - Сила_удара_2 - Ловкость_2 $
3.	«Автозавод» / «Модель автомобиля»	Предположим, что на вашем заводе 10 конвейеров. Каждому из них требуется плановый ремонт по заданному расписанию. Определите, требуется ли перенастраивать другой конвейер для выпуска продукции ремонтируемого, используя информацию об излишках продукции на заводском складе.
4.	«Интернет-магазин» / «Товар»	Составьте отчёт для курьера вашего интернет-магазина с информацией об адресах доставки товаров, основываясь на данных оплаченных, но ещё не развезённых заказах.
5.	«Картинная галерея» / «Картина»	Часть ваших картин находится на экспозиции, часть хранится на складе. Нужно подготовить отчёт по картинам определённого автора, которые можно временно передать на выставку в другом музее, не нарушая существующую экспозицию.
6.	«Воздушная обстановка» / «Позиция борта»	Ваш небольшой аэродром может принимать только малый размер воздушных объектов, а также только гражданских. Необходимо регистрировать все запросы на посадку, а давать её только разрешённым судам. В случае запроса на

№№ ДЗ	Наименование задачи / карточки	Задание для ЛР№4
		экстренную посадку можно давать разрешение средним и военным судам, при этом остальные запросы отклоняются. Составьте систему автоматического управления разрешениями на посадку (сформируйте отчет разрешений на основании текущей воздушной обстановки).
7.	«Клубная лига» / «Футбольный клуб»	Необходимо составить расписание матчей чемпионата по правилам выбывания на следующий сезон, в котором принимают участие 16 лучших команд. При этом нужно учесть распределение первых 4 по рейтингу команд так, чтобы все они потенциально могли занять призовые места.
8.	«Экологическая обстановка» / «Экологическая карточка»	Необходимо определить степень увеличения загрязнения населённых пунктов, которое может произойти из-за выброса ядовитых веществ на предприятии в одном из них. При этом увеличение загрязнения рассчитывается по степени удалённости населённого пункта от эпицентра (обратно квадрату расстояния). Получите отчёт о 5 наиболее загрязнённых населённых пунктах.
9.	«Карта галактики» / «Звездная система»	Человечество отправило автоматические станции на разведку к 5 ближайшим звёздным системам. Определите, когда мы можем получить сигналы от них о достижении цели, если все они движутся с разными скоростями.
10.	«Коллекция монет» / «Монета»	Произошло ограбление музея с редкими монетами. Страховой компании необходимо составить отчёт по страховой выплате музею, если известно, что часть экспозиции сохранилась, а условная стоимость экспонатов зафиксирована в специальной таблице.
11.	«Музыкальная коллекция» / «Музыкальный трек»	На основании предпочтений (рейтинга), устанавливаемого для треков пользователями вашего проигрывателя, вы должны предложить индивидуальные рекомендации конкретному пользователю.
12.	«Магазин музыкальных инструментов» / «Инструмент»	На основании информации о сделанных и выполненных заказах, а также наличии инструмента на складе, необходимо сделать заказ на пополнение склада с инструментами.
13.	«Магазин автозапчастей» / «Автозапчасть»	Необходимо сформировать ежемесячный отчёт о просроченных товарах для списания. Этот отчёт необходимо сохранять на постоянной основе, т.к. эта информация будет

№№ ДЗ	Наименование задачи / карточки	Задание для ЛР№4
		учитываться в будущем.
14.	«Выборы в Муниципалитет» / «Кандидат»	Необходимо построить отчёт по распределению рейтинга партий для определения мест в Муниципалитет на основании результатов выборов их кандидатов в округах. Нужно учесть, что по какому-то округу выборы могут быть признаны недействительными и все результаты по этому округу учитывать не нужно.
15.	«Букмекерская компания» / «Ставка»	Необходимо построить отчёт с перечнем причитающихся премий для победителей в очередном розыгрыше. Премия вычисляется по следующей формуле: $(1) P_i = S_r * V_i / S_c$ $(2) S_r = S_c - S_{\%}$, где: P_i – премия i-го игрока-победителя; S_r – сумма всех премий; V_i – ставка выигравшего игрока; S_c – сумма всех ставок; $S_{\%}$ – часть общей суммы, отходящей букмекерской конторе.
16.	«Ресторан быстрого питания» / «Блюдо»	Необходимо посчитать калорийность блюда на основании калорийности ингредиентов.
17.	«Курьерская служба» / «Заказ»	Составьте отчёт для курьера вашего интернет-магазина с информацией об адресах доставки товаров, основываясь на данных оплаченных, но ещё не развезённых заказах.
18.	«Магазин приложений» / «Приложение»	На основании подготовленного пользователем заказа, а также истории выполненных им заказов необходимо подготовить перечень рекомендаций для дополнительных товаров «с этим покупают следующее: ...».
19.	«Онлайн-школа программирования» / «Курс»	Необходимо составить отчёт для обзвона (или для рассылки почтового сообщения) пользователей, которые сформировали заказ, но ещё не выполнили оплату.
20.	«Каталог библиотеки» / «Книга»	Необходимо сформировать отчёт по книгам, которые находятся на руках у читателей и закончились в хранилище, чтобы заказать дополнительные экземпляры. Читатель может взять несколько книг.

№№ ДЗ	Наименование задачи / карточки	Задание для ЛР№4
21.	«Портфолио студента» / «Портфолио»	Нужно составить отчёт для премирования кураторов на основании рейтинга готовности их студентами портфолио. Отчет должен выводить кураторов по убыванию среднего рейтинга портфолио их групп.
22.	«Банк» / «Транзакция»	Необходимо выполнить проверку возможности выполнения транзакции: а) у отправителя и получателя счета должны быть разблокированы; б) у отправителя тип счёта должен допускать возможность списания средств, у получателя – возможность начисления средств; в) у отправителя на счёте должна быть сумма не меньше снимаемой для перевода. Необходимо хранить всю историю транзакций: и успешных, и нет.
23.	«Авиа-рейс» / «Бронь»	Необходимо сформировать отчёт по рейсам (№ рейса, откуда, куда) с максимальным количеством просроченных броней.
24.	«Онлайн-кинотеатр» / «Фильм»	На основании просмотренных пользователем фильмов и поставленных им рейтингов необходимо подготовить перечень рекомендаций фильмов для просмотра.
25.	«Музыкальный фестиваль» / «Группа»	Каждый раз фестиваль тяжёлого рока длится несколько дней. В день выступают 2-4 группы. Организатор фестиваля хочет получить отчёт о том какие группы выступали в день, когда было продано больше всего пива, чтобы пригласить эти группы в следующий раз.

6.4 Требования к отчету

Отчет должен быть составлен в виде текстового файла в формате Markdown с наименованием «README.md». Файл должен быть сохранён в репозитории в соответствующем проекте.

Помимо общих требований, отчет должен содержать нарисованную от руки диаграмму классов модели с выделенными на ней сущностями, объектами-значениями, службами и агрегатом.

6.5 Контрольные вопросы

После выполнения данной контрольной работы нужно уметь ответить на следующие контрольные вопросы:

М.В. Фетисов. Методические указания по выполнению лабораторных работ и домашнего задания по дисциплине «Современные средства разработки программного обеспечения»

- 1) Дайте определение понятия «проектирования по модели» (model-driven design, MDD).
- 2) Дайте определение понятия «сущность».
- 3) Дайте определение понятия «объект-значение».
- 4) Дайте определение понятия «служба».
- 5) Дайте определение понятия «модуль».
- 6) Дайте определение понятия «агрегат».
- 7) Дайте определение понятия «рефакторинг».
- 8) Почему при проектировании по модели стараются сократить время между формированием модели (созданием диаграммы классов модели) и кодированием модели?
- 9) Зачем нужен рефакторинг?
- 10) Когда выполняется рефакторинг?
- 11) Дайте определение понятия «ограниченный контекст».
- 12) Перечислите отношения между ограниченными контекстами.
- 13) В каких случаях необходимо использовать отношение «конформист» между ограниченными контекстами? Приведите пример такого отношения.
- 14) В каких случаях необходимо использовать отношение «предохранительный уровень» между ограниченными контекстами? Приведите пример такого отношения.

7 Литература

- 1 Самостоятельно устанавливаемый образовательный стандарт высшего образования МГТУ им. Н.Э. Баумана (на основе ФГОС 3++) / уровень высшего образования бакалавриат / направление подготовки 09.03.01 «Информатика и вычислительная техника». – Москва, 2019.
- 2 Самостоятельно устанавливаемый образовательный стандарт высшего образования МГТУ им. Н.Э. Баумана (на основе ФГОС 3++) / уровень высшего образования бакалавриат / направление подготовки 09.03.03 «Прикладная информатика». – Москва, 2019.
- 3 Иванова Г.С. Программирование: Учеб. для вузов. – М.: «Кнорус», 2014.
- 4 Иванова Г.С. Ничушкина Т.Н., Объектно-ориентированное программирование. Учеб. для вузов. – М.: Изд-во МГТУ им. Н.Э. Баумана, 2014.
- 5 Иванова Г.С. Технология программирования: Учеб. для вузов. – М.: «Кнорус», 2013.
- 6 Буч Г. Объектно-ориентированное проектирование с примерами применения. – М.: Конкорд, 1992. – 519 с.
- 7 Гамма Э., Хелм Р., Джонсон Р. [и др.]. Приемы объектно-ориентированного проектирования. Паттерны проектирования. – СПб.: Питер, 2001. – 366 с.
- 8 Мартин, Роберт С. Гибкая разработка программ на Java и C++: принципы паттерны и методики. – СПб. : ООО «Диалектика», 2019. – 704 с.
- 9 Холл, Гэри Маклин. Адаптивный код: гибкое кодирование с помощью паттернов проектирования и принципов SOLID, 2-е изд. – СПб.: ООО «Диалектика», 2018. – 448 с.
- 10 Эванс, Эрик. Предметно-ориентированное проектирование (DDD): структуризация сложных программных систем. – Спб. ООО «Диалектика», 2019. – 448 с.
- 11 JSON Schema. – URL: <http://json-schema.org>
- 12 Вузовский репозиторий исходных кодов GitLab. – URL: <https://bmstu.codes>
- 13 ISO/IEC 14882:2017 Programming languages — C++. URL: <https://www.iso.org/standard/68564.html>

Приложение А. Соглашение о стиле кодирования

Таблица 8 – Перечень основных правил соглашения о стиле кодирования

№№	Правило	Пояснения
1. Запрещающие соглашения		
1.1.	Следует избегать использования глобальных переменных	
1.2.	Следует избегать использования макросов	
1.3.	Следует избегать использования оператора goto	
1.4.	Следует избегать использования указателей C++	Используйте интеллектуальные указатели: <code>unique_ptr</code> , <code>shared_ptr</code> , <code>weak_ptr</code>
1.5.	Следует избегать использования массивов C++	Используйте параметризованные контейнеры стандартной библиотеки C++: <code>array</code> , <code>vector</code> , <code>deque</code> и другие
2. Соглашения об именовании		
2.1.	Имена, представляющие типы, должны быть написаны в смешанном регистре, начиная с верхнего	Пример: <code>Line, SavingsAccount</code>
2.2.	Имена переменных должны быть записаны в нижнем регистре, слова должны разделяться знаком подчеркивания	Пример: <code>line, savings_account</code> Распространённая в настоящее время практика в сообществе разработчиков C++. Позволяет легко отличать переменные от типов, предотвращает потенциальные коллизии имён, например: <code>Line line;</code> Также, записанные таким образом переменные легко отличить от названия метода (см. далее).
2.3.	Названия методов и функций должны быть глаголами, быть записанными в смешанном регистре и начинаться с нижнего	Пример: <code>getName()</code> , <code>computeTotalWidth()</code>

№№	Правило	Пояснения
2.4.	Аббревиатуры и сокращения в именах должны записываться в нижнем регистре	<p>Пример:</p> <pre>exportHtmlSource(); // НЕЛЬЗЯ: exportHTMLSource(); openDvdPlayer(); // НЕЛЬЗЯ: openDVDPlayer();</pre> <p>Использование верхнего регистра может привести к конфликту имён, описанному выше. Иначе переменные бы имели имена dVD, hTML и т. д., что не является удобочитаемым. Другая проблема уже описана выше; когда имя связано с другим, читаемость снижается; слово, следующее за аббревиатурой, не выделяется так, как следовало бы.</p>
2.5.	Членам класса с модификатором <code>private</code> следует присваивать префикс-подчёркивание	<p>Пример:</p> <pre>class SomeClass { private: int _length; }</pre> <p>Не считая имени и типа, область видимости — наиболее важное свойство переменной. Явное указание модификатора доступа в виде подчёркивания избавляет от путаницы между членами класса и локальными переменными. Это важно, поскольку переменные класса имеют большее значение, нежели переменные в методах, и к ним следует относиться более осторожно. Дополнительным эффектом от префикс-подчёркивания является разрешение проблемы именования в методах, устанавливающих значения, а также в конструкторах:</p> <pre>void setDepth (int depth) { _depth = depth; }</pre>
2.6.	Все имена следует записывать по-английски	<p>Пример:</p> <pre>file_name; // НЕ РЕКОМЕНДУЕТСЯ: imyaFayla</pre>
2.7.	Переменные, имеющие большую область видимости, следует называть длинными именами,	Имена временных переменных, используемых для хранения временных значений или индексов, лучше всего делать короткими. Программист, читающий такие

№№	Правило	Пояснения
	имеющие небольшую область видимости — короткими	переменные, должен иметь возможность предположить, что их значения не используются за пределами нескольких строк кода. Обычно это переменные i, j, k, l, m, n (для целых), а также c и d (для символов).
2.8.	Имена объектов не указываются явно, следует избегать указания названий объектов в именах методов	Пример: <pre>line.getLength();</pre> // НЕ РЕКОМЕНДУЕТСЯ: <pre>line.getLineLength();</pre> <p>Второй вариант смотрится вполне естественно в объявлении класса, но совершенно избыточен при использовании, как это и показано в примере.</p>
3. Особые правила наименования		
3.1.	Слова get/set должны быть использованы везде, где осуществляется прямой доступ к атрибуту	Пример: <pre>employee.getName();</pre> <pre>employee.setName(name);</pre> <pre>matrix.getElement(2, 4);</pre> <pre>matrix.setElement(2, 4, value);</pre>
3.2.	Множественное число следует использовать для представления наборов (коллекций) объектов	Пример: <pre>vector<Point> points;</pre> <pre>int values[];</pre> <p>Улучшает читаемость, поскольку имя даёт пользователю прямую подсказку о типе переменной и операциях, которые могут быть применены к этим элементам.</p>
3.3.	Префикс n следует использовать для представления числа объектов	Пример: <pre>n_points, n_lines</pre> <p>Обозначение взято из математики, где оно является установившимся соглашением для обозначения числа объектов.</p>
3.4.	Суффикс No следует использовать для обозначения номера сущности	Пример: <pre>table_no, employee_no</pre> <p>Обозначение взято из математики, где оно является установившимся соглашением для обозначения номера сущности.</p>
3.5.	Переменным-итераторам следует давать имена i, j, k и т. д. Или it, чтобы не путать его с целым	Пример: <pre>for(int i = 0; i < n_tables); i++)</pre>

№№	Правило	Пояснения
	типом.	<pre>{ ... }</pre> <p>Обозначение взято из математики, где оно является установившимся соглашением для обозначения итераторов.</p> <pre>for(auto it = list.begin(); it != list.end(); ++it) { Element element = *it; ... }</pre>
3.6.	Префикс <code>is</code> следует использовать только для булевых (логических) переменных и методов	<p>Пример:</p> <pre>s_set, is_visible, is_finished, is_found, is_open</pre> <p>Использование этого префикса избавляет от таких имён, как <code>status</code> или <code>flag</code>. <code>is_status</code> или <code>is_flag</code> просто не подходят, и программист вынужден выбирать более осмысленные имена.</p> <p>В некоторых ситуациях префикс <code>is</code> лучше заменить на другой: <code>has</code>, <code>can</code> или <code>should</code>:</p> <pre>bool has_license(); bool can_evaluate(); bool should_sort();</pre>
4. Файлы исходных кодов		
4.1.	Заголовочным файлам C++ следует давать расширение <code>.h</code> (предпочтительно) либо <code>.hpp</code> . Файлы исходных кодов могут иметь расширения <code>.cpp</code> . Имена файлов записываются в нижнем регистре	<p>Пример:</p> <pre>myclass.cpp, myclass.h</pre> <p>Эти расширения, одобряемые стандартом C++.</p>
4.2.	Класс следует объявлять в заголовочном файле и определять (реализовывать) в файле исходного кода, имена файлов совпадают с именем класса (но пишутся в нижнем регистре)	<p>Пример:</p> <pre>myclass.cpp, myclass.h</pre> <p>Облегчает поиск связанных с классом файлов. Очевидное исключение — шаблонные классы, которые должны быть объявлены и определены в заголовочном файле.</p>

№№	Правило	Пояснения
4.3.	Нельзя использовать специальные символы (например, TAB) и разрывы страниц	Такие символы вызывают ряд проблем, связанных с редакторами, эмуляторами терминалов и отладчиками, используемыми в программах для совместной разработки и кроссплатформенных средах.
4.4.	Заголовочные файлы должны содержать защиту от вложенного включения	Пример: <pre>#ifndef _COM_COMPANY_MODULE_CLASSNAME_H_ #define _COM_COMPANY_MODULE_CLASSNAME_H_ ... #endif // _COM_COMPANY_MODULE_CLASSNAME_H_</pre> Конструкция позволяет избегать ошибок компиляции. Это соглашение позволяет увидеть положение файла в структуре проекта и предотвращает конфликты имён.
4.5.	Директивы включения следует сортировать (по месту в иерархии системы, ниже уровень — выше позиция) и группировать. Оставляйте пустую строку между группами	Пример: <pre>#include <fstream> #include <iomanip> #include <qt/qbutton.h> #include <qt/qtextfield.h> #include "com/company/ui/PropertiesDialog.h" #include "com/company/ui/MainWindow.h"</pre> Пути включения не должны быть абсолютными. Вместо этого следует использовать директивы компилятора.
4.6.	Директивы включения должны располагаться только в начале файла	Общая практика. Избегайте нежелательных побочных эффектов, которые может вызвать «скрытое» включение где-то в середине файла исходного кода.
5. Разное		
5.1.	Следует избегать «магических» чисел в коде. Числа, отличные от 0 или 1, следует объявлять, как именованные константы	Если число само по себе не имеет очевидного значения, читаемость улучшается путём введения именованной константы. Другой подход — создание метода, с помощью которого можно было бы осуществлять доступ к константе.
5.2.	Следует использовать <code>null_ptr</code> вместо «NULL»	NULL является частью стандартной библиотеки C и устарело в C++
6. Оформление и комментарии		
6.1.	Основной отступ следует делать в четыре пробела	Пример: <pre>for(i = 0; i < n_elements; i++)</pre>

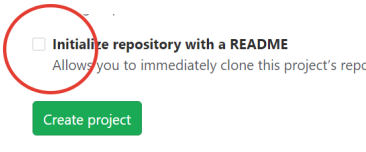
№№	Правило	Пояснения
		<pre>a[i] = 0;</pre> <p>Отступ в один или два пробела достаточно мал, чтобы отражать логическую структуру кода. Отступ более 4 пробелов делает глубоко вложенный код нечитаемым и увеличивает вероятность того, что строки придётся разбивать. Широко распространены варианты в 2, 3 или 4 пробела; причём 2 и 4 — более широко.</p>
6.2.	<p>Блоки кода следует оформлять так, как показано в примере 1 (рекомендуется), но ни в коем случае не так, как показано в примере 2. Оформление функций и классов должно следовать примеру 1</p>	<p>Пример 1:</p> <pre>while(!done) { doSomething(); done = moreToDo(); }</pre> <p>Пример 2:</p> <pre>while(!done) { doSomething(); done = moreToDo(); }</pre> <p>Пример 2 использует лишние отступы, что мешает ясному отображению логической структуры кода.</p>

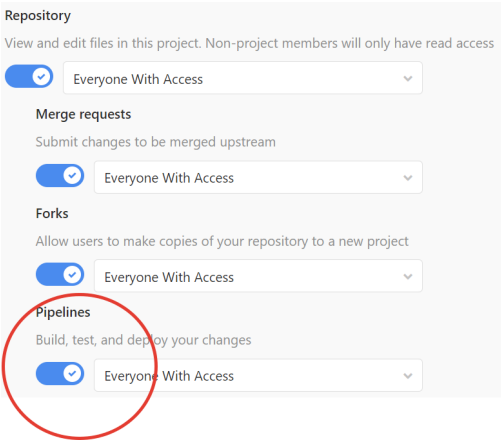
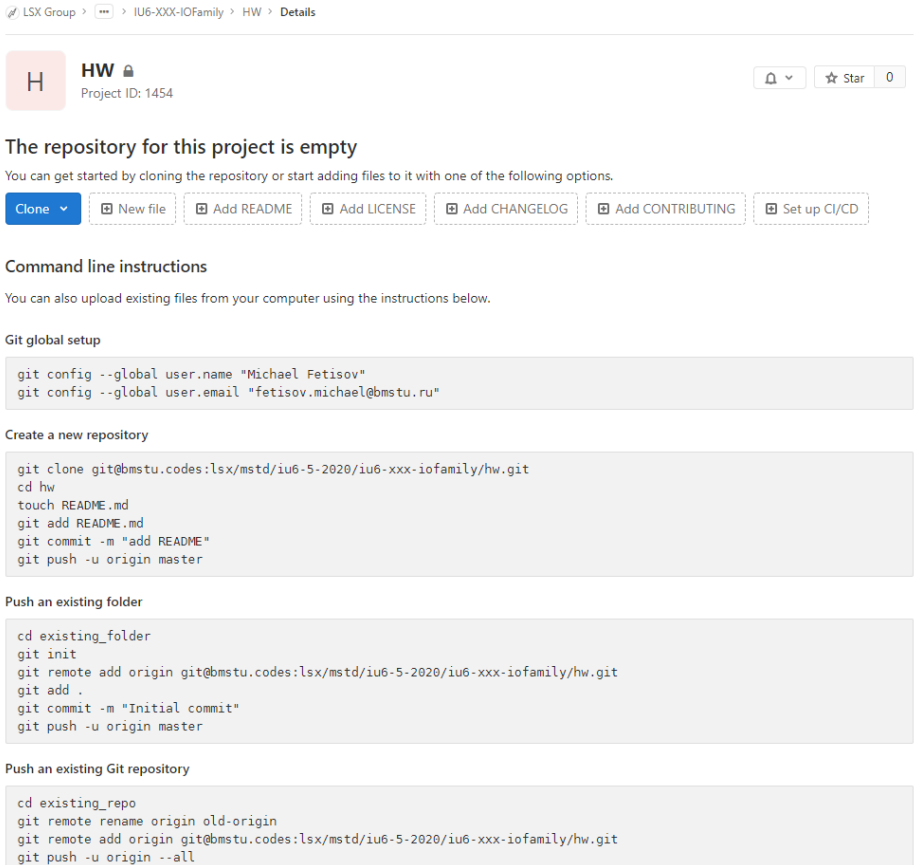
Приложение Б. Настройка работы с вузовским репозиторием кода

В таблице 9 приводится последовательность действий для установки и настройки системы Git, а также подключения к GitLab и настройки работы с проектами в этой системе.

Таблица 9 – Последовательность действий для настройки работы с вузовским репозиторием кода

№№	Действие	Пояснение
1.	Установка Git на рабочий компьютер	<p>Описание установки Git можно почитать на следующей странице: https://git-scm.com/book/ru/v1/Введение-Установка-Git.</p> <p>После завершения установки нужно проверить успешность ее выполнения, введя в терминале (или в Git Bash, если вы ставили в ОС Windows) следующую команду:</p> <pre>git --version</pre> <p>В результате ее выполнения должна вывестись версия установленной системы Git. Номер версии не должен быть меньше 2.10.</p>
2.	Конфигурация Git	<p>После установки Git необходимо указать реквизиты пользователя. Для этого нужно ввести следующие команды:</p> <pre>git config --global user.name "<ФИО>" git config --global user.email "<почта>"</pre> <p>, где <ФИО> – ваша фамилия и инициалы, <почта> – адрес вашей электронной почты.</p>
3.	Получение открытого ключа SSH	<p>Работа по протоколу SSH является предпочтительной.</p> <p>Для получения открытого ключа SSH нужно сначала проверить его наличие, возможно он у вас установлен. Для этого в ОС GNU/Linux нужно найти скрытый каталог <code>.ssh</code> и убедиться в наличии в нем файла <code>id_rsa.pub</code>. Открытый ключ SSH является содержимым данного файла.</p> <p>Если указанного каталога или файла нет, то нужно выполнить генерацию ключей SSH. Как это сделать можно почитать на следующей странице: Git-на-сервере-Генерация-открытого-SSH-ключа.</p>
4.	Регистрация на вузовском репозитории исходного кода и регистрация	<p>После регистрации на вузовском репозитории исходного кода (https://bmstu.codes) нужно зайти в личный кабинет (пункт «Настройки»), выбрать в левой панели пункт «SSH-ключи» и добавить открытый ключ SSH.</p>

№№	Действие	Пояснение
	открытого SSH-ключа	
5.	Получение доступа к группе с настроенным механизмом непрерывной интеграции	После регистрации в вузовском репозитории исходного кода нужно выслать преподавателю придуманный вами никнейм, чтобы предоставить вам доступ к группе домашних заданий и лабораторных работ. Пример наименования группы: «IU6-5-19» (последнее число может отличаться).
6.	Создание группы проектов в GitLab	<p>После получения доступа к группе домашних заданий и лабораторных работ нужно зайти в нее и создать группу проектов, в которой вы будете размещать свои проекты с выполненными домашней и лабораторными работами. Наименование группы проектов должно быть следующего формата:</p> <p>IU6-XXX-IOFamilia,</p> <p>Где XXX – номер группы, IO – инициалы, Familia – фамилия.</p> <p>Инициалы, фамилия и все остальные буквы должны быть из английского алфавита (фамилия – созвучная вашей на английском языке).</p> <p>Если практикум выполняется вдвоём, то в наименовании группы должны указываться оба студента.</p>
7.	Создание проектов	<p>Внутри созданной группы проектов нужно создать проекты HW, L1, L2, L3 и L4.</p> <p>ВНИМАНИЕ! Не нужно отмечать добавление в проект файла README.md, этот файл проще будет сделать на основе взятого из проекта-образца.</p> 
8.	Настройка проектов	В каждом проекте нужно убедиться, чтобы был включён запуск сценария непрерывной интеграции. Для этого нужно проверить пункт Pipelines настройках проекта (Settings / General / Visibility, project features, permissions / Repository / Pipelines):

№№	Действие	Пояснение
		
9.	Настройка удаленного репозитория для вашего проекта	<p>На страницах созданных пустых проектов появятся подсказки и рекомендации по настройке ваших локальных репозиториях, в том числе по настройке удаленного репозитория для вашего проекта:</p> 
10.	Сохранение изменений в удаленном репозитории	<p>После того, как будут произведены какие-либо изменения в исходном коде, нужно сохранить их в локальном, а затем в удаленном репозитории. Для этого нужно, находясь в терминале в каталоге проекта, последовательно ввести следующие команды:</p> <ul style="list-style-type: none"> - Добавление изменений в индекс репозитория:

№№	Действие	Пояснение
		<pre>git add .</pre> <ul style="list-style-type: none"> - Добавление изменений в репозиторий: <pre>git commit -m "XXX"</pre>, где XXX – комментарий к изменениям - Сохранение изменений в удаленный репозиторий: <pre>git push -u origin master</pre> <ul style="list-style-type: none"> – при первом добавлении и: <pre>git push</pre> – при последующих. <p>Старайтесь чаще делать сохранение изменений.</p>
11.	Работа непрерывной интеграции (CI)	<p>При сохранении изменений в удаленный репозиторий для проектов группы домашних заданий и лабораторных работ выполняется непрерывная интеграция, которая состоит из следующих этапов (см. файл <code>.gitlab-ci.yml</code>):</p> <ul style="list-style-type: none"> - компиляция. Компиляция проводится компилятором GNU C++ из пакета GCC (GNU Compiler Collection); - прогон автотестов; - генерация описания классов проекта и выкладывание его в page-область проекта. Посмотреть эту область можно через левую панель, пункт «Настройка» / «Страницы», где указан путь к page-области. <p>Если хотя бы один из этапов выполнен с ошибками, считается, что сборка проекта не удалась. В этом случае домашнее задание или лабораторная работа считаются не выполненными.</p>

Перечень условных обозначений, сокращений и терминов

RAII	«Resource Acquisition Is Initialization», бук. «получение ресурса есть инициализация» — техника использования механизма контроля зоны видимости объекта для управления ресурсами, которые он содержит
SOLID	Мнемонический акроним для пяти основных принципов объектно-ориентированного проектирования
ЛР	Лабораторная работа
ИУ6	– Кафедра «Компьютерные системы и сети» факультета «Информатика и системы управления» МГТУ им Н.Э. Баумана
ООП	– Объектно-ориентированное проектирование
ОПКС	Собственные общепрофессиональные компетенции
ПКС	Собственные профессиональные компетенции
ПО	– Программное обеспечение
СУОС	Самостоятельно установленный образовательный стандарт
ЯП	Язык программирования