Фонд оценочных средств для промежуточной аттестации по дисциплине (модулю)
«Проектирование приложений на языке C#»

Направление подготовки (специальность)
09.03.04 «Программная инженерия»

Направленность (профиль)
«Разработка программно-информационных систем»

Присваиваемая квалификация
Бакалавр

Форма обучения
Очная

Год набора
2025

Челябинск, 2025 г.

09.03.04 Программная инженерия, Разработка программно-информационных систем, бакалавр, *Проектирование приложений на языке C#*, 2025, очная

Фонд оценочных средств дисциплины (модуля) одобрен и рекомендован

Проректор по учебной работе утверждено 24.02.2025 А.А. Саламатов

Ученым советом института информационных технологий

Протокол заседания № 6 от 20.02.2025

Председатель Ученого совета института информационных технологий согласовано Ю. В. Петриченко

Заседанием кафедры информационных технологий и экономической информатики

Протокол заседания № 6 от 20.02.2025

И. о. заведующего кафедрой согласовано С.А. Скрипов

Автор (составитель) С.А. Скрипов

Структура рабочей программы соответствует приказу ректора ФГБОУ ВО «ЧелГУ» от «13» апреля 2021 г. № 247-1

Содержание

1. Паспорт фонда оценочных средств

Направление подготовки: 09.03.04 Программная инженерия
Направленность: Разработка программно-информационных систем
Дисциплина: Проектирование приложений на языке C#
Семестры: 5
Форма промежуточной аттестации: зачёт
Для оценивания результатов обучения используется балльно-рейтинговая система.

2. Перечень формируемых компетенций

Изучение дисциплины «Проектирование приложений на языке C#» направлено на формирование компетенций, приведённых в 1.

Таблица 1. Результаты обучения по дисциплине.

Коды компетенции согласно ФГОС (ОПОП ВО)	Содержание компетенций согласно ФГОС (ОПОП ВО)	Индикаторы достижения компетенции согласно ОПОП	Перечень планируемых результатов обучения по дисциплине
1	2	3	4
ОПК-6	Способен разрабатывать алгоритмы и программы, пригодные для практического использования, применять основы информатики и программирования к проектированию, конструированию и тестированию программных продуктов;	ОПК-6.1. Демонстрирует знание основ информатики, теории алгоритмов, методологии и технологии программирования ОПК-6.2. Демонстрирует умения разрабатывать алгоритмические и программные решения, проводить проектирование, конструирование и тестирование программных продуктов ОПК-6.3. Имеет практический опыт использования технологий разработки программного обеспечения	Знать: теорию алгоритмов, методологию и технологию объектно-ориентированного программирования на языке C# Уметь: разрабатывать алгоритмические и программные решения на языке C# Владеть: навыками использования объектно-ориентированных технологий разработки программного обеспечения
ПК-2	Владение навыками использования различных технологий промышленной разработки программного обеспечения с применением инструментов автоматизации сборки, интеграции, тестирования и развертывания ПО	ПК-2.1. Демонстрирует знание основных принципов и технологий промышленной разработки программного обеспечения ПК-2.2. Демонстрирует умения разрабатывать программное обеспечение с применением инструментов автоматизации сборки, интеграции, тестирования и развертывания ПО ПК-2.3. Имеет практический опыт промышленной разработки программного обеспечения	Знать: основные объектно-ориентированные технологии промышленной разработки на языке C# Уметь: разрабатывать программное обеспечение на языке C# с использованием интегрированной среды разработки Владеть: навыками промышленной разработки программного обеспечения на языке C# с использованием интегрированной среды

3. Содержание оценочных средств по дисциплине

3.1. Виды оценочных средств

Таблица 2. Виды оценочных средств.

№ п/п	Код компетенции/ планируемые результаты обучения	Контролируемые темы/ разделы	Наименование оценочного средства для текущего контроля	Наименование оценочного средства на промежуточной аттестации/№ задания
1	ОПК-6.1. Демонстрирует знание основ информатики, теории алгоритмов, методологии и технологии программирования Знать: теорию алгоритмов, методологию и технологию объектно-ориентированного программирования на языке C#	Синтаксис языка C#. Объектно-ориентированое программирование Проектирование объектно-ориентированных приложений на языке C#	Тест	Задания теста № 1-20
2	ОПК-6.2. Демонстрирует умения разрабатывать алгоритмические и программные решения, проводить проектирование, конструирование и тестирование программных продуктов Уметь: разрабатывать алгоритмические и программные решения на языке C#	Синтаксис языка C#. Объектно-ориентированое программирование Проектирование объектно-ориентированных приложений на языке C#	Тест	Практические задания 1-18
3	ОПК-6.3. Имеет практический опыт использования технологий разработки программного обеспечения Владеть: навыками использования объектно-ориентированных технологий разработки программного обеспечения	Синтаксис языка C#. Объектно-ориентированое программирование Проектирование объектно-ориентированных приложений на языке C#	Тест	Практические задания 1-18
4	ПК-2.1. Демонстрирует знание основных принципов и технологий промышленной разработки программного обеспечения Знать: основные объектно-ориентированные технологии промышленной разработки на языке C#	Синтаксис языка C#. Объектно-ориентированое программирование Проектирование объектно-ориентированных приложений на языке C#	Тест	Задания теста № 1-20
5	ПК-2.2. Демонстрирует умения разрабатывать программное обеспечение с применением инструментов автоматизации сборки,	Синтаксис языка C#. Объектно-ориентированое программирование Проектирование объектно-ориентированных	Тест	Практические задания 1-18

	интеграции, тестирования и развертывания ПО Уметь: разрабатывать программное обеспечение на языке C# с использованием интегрированной среды разработки	приложений на языке C#		
6	ПК-2.3. Имеет практический опыт промышленной разработки программного обеспечения Владеть: навыками промышленной разработки программного обеспечения на языке C# с использованием интегрированной среды	Синтаксис языка C#. Объектно-ориентированое программирование Проектирование объектно-ориентированных приложений на языке C#	Тест	Практические задания 1-18

Типовые задания, критерии и показатели оценивания в рамках текущего контроля представлены в рабочей программе дисциплины (модуля). Полные комплекты оценочных средств и контрольно-измерительных материалов хранятся на кафедре.

3.2. Содержание оценочных средств

База тестовых вопросов

№ п/п	Формулировка вопроса	Варианты ответов (полужирным шрифтом – верные варианты)

1.	class SomeClass {
	public static int s;
	private int p;
	public int d;
	}
	Отметьте все корректные обращения к полям объявленного выше класса SomeClass
		• **SomeClass.s = 42**
		• SomeClass.d = 42
		• new SomeClass().s = 42
		• **new SomeClass().d = 42**
		• new SomeClass().p = 42

2.	namespace MyNamespace
	{
	internal class ClassA { }
	
	public class ClassB
	{
	public ClassA Method1() { return null; }
	private ClassB Method2() { return null; }
	}
	}
	Перечислите все способы, которыми можно избавиться от ошибок компиляции в этом коде
		• **Перенести этот код в другую сборку**
		• **Сменить модификатор доступа ClassB с public на internal**
		• Сменить модификатор доступа Method2 с private на public
		• **Сменить модификатор доступа Method1 с public на private**
		• **Сменить модификатор доступа ClassA с internal на public**

		• Сменить модификатор доступа Method1 с public на internal

3.	class ClassA { }
	
	class ClassB : ClassA { }
	
	class ClassC : ClassA { }
	
	class Program
	{
	public static void Main()
	{
	ClassA a = new ClassA();
	ClassB b = new ClassB();
	ClassC c = new ClassC();
	ClassA d = (ClassA) b;
	}
	}
	Конверсия переменной b к ClassA — это
		• **Upcast**
		• Downcast
		• Ни то, ни другое

4.	Конверсия переменной c к ClassA	• **Пройдет без ошибок** • Вызовет ошибку компиляции • Вызовет ошибку выполнения
5.	Конверсия переменной a к ClassB - это	• Upcast • **Downcast** • Ни то, ни другое
6.	Конверсия переменной d к ClassC	• Пройдет без ошибок • Вызовет ошибку компиляции • **Вызовет ошибку выполнения**
7.	К каким типам можно без ошибок привести переменную d?	• **ClassA** • **ClassB** • ClassC
8.	Конверсия переменной c к ClassB	• Upcast • Downcast • **Ни то, ни другое**
9.	interface A { } interface B : A { } class X : A {} class Y : X, B {} class Z : X {} Какие из этих следующих операторов не выбросят исключение?	• **Y as B** • **X as B** • **X** • **Y** • (B) Z
10.	Какие из этих утверждений верны для абстрактных базовых классов, но не верны для интерфейсов	• **Могут содержать поля** • **Могут содержать реализацию методов базового класса или**

		интерфейса • Могут содержать свойства • **Могут содержать методы с реализованной логикой** • **Могут содержать приватные члены**
11.	Как вы думаете, зачем может быть полезно разделение кода на слои?	• **Проще навигация по коду — новичку проще понять, где искать тот или иной класс** • **Появляется возможность повторного использования слоя предметной области в разных приложениях** • Без разделения на слои, код нельзя будет протестировать • Код, для которого критична производительность, можно вынести на более низкий уровень, и он будет работать быстрее
12.	В каком слое должен оказаться класс, описывающий контрол Windows Forms, для отображения информации о пациенте?	• **UI** • Application • Domain • infrastructure
13.	В каком слое должен оказаться вспомогательный метод расширения класса FlightLevel, определяющего каким курсом движется воздушное судно, преимущественно северным или южным?	• UI • Application • Domain • **infrastructure**
14.	Отметьте верные утверждения про разделение кода на слои согласно DDD	• Классы, описывающие предметную область должны быть в слое приложения • **Слой предметной области может зависеть только от слоя инфраструктуры** • Каждый слой может зависеть только от одного следующего слоя • Каждый слой может

		зависеть от любого другого слоя.
15.	Как разделение на слои может выглядеть в коде на C#	• Выделять принадлежность каждого члена к тому или иному слою комментарием • Помещать члены класса, относящиеся к слою X в отдельный регион с именем X • **Создать по отдельной сборке на каждый слой** • **Создать по отдельному пространству имен на каждый слой**
16.	Какой пункт НЕ является отличительной особенностью сущности (Entity)?	• При изменении свойств, у сущности сохраняется идентичность • У сущности есть идентификатор, по которому ее можно идентифицировать, например, при загрузке из хранилища • Две сущности одного типа считаются равными, если у них равны идентификаторы • **Поля сущности не могут быть Value-объектами** • Действия с сущностью как правило оформляют в виде методов класса этой сущности
17.	Какой пункт НЕ является отличительной особенностью объекта-значения (Value-object)?	• Значение как правило стараются делать неизменяемым классом • **У Значений как правило нет методов, а есть только свойства** • Два значения одного типа считаются равными,

		если у них равны все их поля • Поля объекта-значения не могут быть сами объектами значениями
18.	Какой пункт НЕ является отличительной особенностью сервиса (Service)?	• **Все операции с сущностями и значениями нужно оформлять в виде методов классов-сервисов** • Обычно сервисы создаются в начале программы в единственном экземпляре • Сервисы содержат операции над классами предметной области, которые по какой-то причине не стали методами этих классов

19.	public class Point
	{
	public readonly int X, Y;
	public Point(int x, int y)
	{
	X = x;
	Y = y;
	}
	public override bool Equals(object other)
	{
	var p = other as Point;
	return p != null && p.X == X && p.Y == Y;
	}
	
	public override int GetHashCode() => X ^ Y;
	}
	Чем является класс Point?
		• Entity-объектом
		• Сервисом
		• **Value-объектом**

20.	Point — это пример богатой (rich) или анемичной (anemic) модели?	• Богатой • **Анемичной**

База практических заданий

№ п/п	Формулировка задания
1	Некто N. написал код, выводящий список устройств, в которых до определенной даты случились критические сбои. К сожалению, N. учился программированию в начале 90-х годов, и не знаком с современными практиками. Скачайте проект Incapsulation.Failures и помогите N. отрефакторить его код: 1. Выделите новый статический метод FindDevicesFailedBeforeDate. Метод должен принимать не более 4-х аргументов. В сигнатуре метода не должно быть Dictionary-

	типов и коллекций с вложенными дженерик-типами, например, List<List<object>>. 2. Значения в аргументах devices и failureTypes должны быть инкапсулированы в сущности Device и Failure. 3. IsFailureSerious, очевидно, не на своем месте. 4. C day и times тоже не все в порядке. Сигнатуру старого метода сохраните, чтобы проходили тесты. Старый метод должен преобразовывать аргументы и вызывать новый метод.
2	Некто М. написал код, описывающий предприятие. Он даже озаботился проверкой целостности для полей этого класса, но, к сожалению, он учился программировать в конце 90-х годов, и знаком лишь со слегка устаревшими практиками проверки целостности. Скачайте проект Incapsulation.EnterpriseTask и помогите М. отрефакторить его код.
3	Нейронная сеть состоит из нейронов, каждый из которых имеет вектор весов. Иногда нужно иметь доступ к весам отдельных нейронов, например, при их инициализации. Однако, при разработке алгоритмов обучения оказывается удобным работать со всеми весами в сети как с единым вектором. Таким образом, нужно организовать различные виды доступа к одним и тем же данным. Скачайте проект Incapsulation.Weights и напишите класс Indexer, который создается как обертка над массивом double[], и открывает доступ к его подмассиву некоторой длины, начиная с некоторого элемента. Ваше решение должно проходить тесты, содержащиеся в проекте. Как и всегда, вы должны следить за целостностью данных в Indexer.
4	При работе над математическими или геометрическими задачами часто приходится создавать "фундаментальные" классы, подобные int или double, для комплексных чисел, кватернионов и т.д. Скачайте проект Incapsulation.RationalNumbers и напишите класс рационального числа. Ваше решение должно проходить тесты, содержащиеся в проекте.
5	Простейший сценарий, когда вам нужна перегрузка методов и реализация интерфейсов - написание небольших структур данных, которые должны быть совместимы с листами, словарями и т.д. Допустим, вы разрабатываете систему для анализа сообщений в техподдержку, и хотите классифицировать их по имени продукту, типу сообщения и его теме. Соответственно, вам необходим класс, обозначающий категорию сообщений с указанными полями. Скачайте проект Inheritance.DataStucture и создайте класс Category.cs. В этом классе переопределите методы Equals и GetHashCode, реализуйте интерфейс IComparable, упорядочивающий категории сначала по продукту, затем по типу и затем - по теме, а также реализуйте все операторы сравнения. Изучите тесты для того, чтобы понять детали реализации.
6	В компьютерной игре, персонаж игрока взаимодействует с различными объектами на карте. Есть всего три способа взаимодействовать: 1. Сражаться с армией. 2. Собирать сокровища. 3. Присваивать объект себе. А вот различных видов объектов на карте уже 5, а будет ещё больше. Скачайте проект Inheritance.MapObjects, откройте файл Task.cs и изучите, как это реализовано сейчас. Проблема в том, что метод Interaction.Make содержит много почти повторяющегося кода, нарушая принцип Dont Repeat Yourself. Кроме того, он будет расти с появлением новых объектов в игре.

	Выделите все поля, необходимую для каждого взаимодействия, в свой интерфейс. Отрефакторьте программу, избавившись от повторяющихся участков кода в Interaction.Make. В итоговом решении Interaction.Make должен работать только с интерфейсами, и не должен содержать упоминаний конкретных классов.
7	Какое же наследование без геометрии! Скачайте проект Inheritance.Geometry и изучите Task.cs. Проблема этого подхода в том, что каждый раз при добавлении нового типа тела придется менять метод в базовом классе. Предположим вы знаете, что в планах добавить ещё много новых геометрических примитивов. В этом случае разумно сделать метод GetVolume абстрактным и переопределить его в классах Cube, Ball и Cylinder. Сделайте это! В финальном решении не должно быть ни одного приведения типа.
8	Давайте теперь предположим, что в предыдущей задаче новых геометрических примитивов добавлять мы не собираемся. Зато собираемся добавлять новые методы для работы с уже имеющимися — они могут вычислять объем, площадь поверхности, рассчитывать точку пересечения объекта с прямой и т.д. В этом случае часто используется шаблон Visitor. Изучите этот шаблон по википедии. Определите интерфейс IVisitor и реализуйте его в двух классах DimensionsVisitor и SurfaceAreaVisitor, для рассчёта размеров (ширина, высота) и площади поверхности фигур. В класс Body добавьте абстрактный метод Accept(IVisitor visitor). Автоматизированные тесты проверяют лишь базовые требования. Проверить, что вы всё сделали правильно можно самостоятельно так: 1. В реализациях Visitor не должно быть ни одного приведения типов и ни одного if-a. Именно этой простотой решение с Visitor-ом лучше исходного с длинным if-else. 2. Работа с каждой фигурой должна оказаться в отдельном методе. А значит даже если добавится новая фигура, будет меньше возможностей случайно внести ошибку в обработку старых фигур. 3. Компилятор должен контролировать, что вы не забыли обработать ни одну из фигур: если вы забудете написать один из методов, программа даже не скомпилируется. 4. В интерфейсе IVisitor, в классе Body и всех его подклассах не должно быть никакого упоминания площади поверхности, размеров или конкретных классов Visitor-ов. А значит при добавлении новых рассчетов, эти классы не нужно будет модифицировать. 5. Для добавления нового метода работы с фигурами, должно быть достаточно добавить новый класс Visitor-a.
9	Наиболее очевидный случай использования дженериков — создание коллекций. Скачайте проект Generics.BinaryTrees и создайте в нем класс бинарного дерева поиска так, чтобы он проходил приложенные тесты. Если у вас останется много времени, оптимизируйте код метода GetEnumerable так, чтобы он работал за O(n) по времени, где n — количество элементов в дереве, и O(1) по памяти. Если времени не останется, вы можете использовать менее оптимальное решение.
10	В анализе данных бывают очень полезны таблицы. Например, строки могут соответствовать датам, столбцы — департаментам, а в ячейках может хранится выручка департамента за

	контракты на соответствующую дату. Сделайте такую таблицу, которая бы: 1. Индексировалась величинами типов, указанных при создании таблицы. 2. Имела бы две возможности для индексирования: 1. С автоматическим созданием нужных строк и столбцов «на лету» при обращении к таблице по соответствующим индексам; 2. Которая бы требовала создания столбцов и строк заранее и выбрасывала исключение при доступе к несуществующим столбцам или строкам. Скачайте проект Generics.Tables и изучите тесты, которые должна проходить ваша таблица, чтобы понять детали задания. Дополнительно подумайте над тем, как не хранить лишней информации в таблице: если в данную дату в данном департаменте не было заключено контрактов, то значение будет 0, и хранить сотни нулей, очевидно, не нужно.
11	Не так-то просто сделать упражнение на ковариацию и контравариацию, но нам удалось. Скачайте проект Generics.Robots и изучите класс Architecture.cs. Он описывает некий проект архитектуры робота. В нем есть AI, вырабатывающий команды, и Device, команды исполняющий. При этом, AI уже готовы для двух типов роботов (Builder и Shooter), а Device есть только для подвижной части. Сейчас все работает, но вам не нравится. Что это за object-ы повсюду, где строгая типизация? Конечно, RobotAI и Mover должны стать дженерик-классами, типизируемыми классом команды. Однако, когда вы это сделаете, вы обнаружите, что эта архитектура не компилируется. Здесь нужно применить ковариацию для того, чтобы исправить эту проблему.
12	Некто N разработал генератор отчетов в проекте Delegates.Reports, который считает простую статистику о погоде по нескольким параметрам за несколько дней. Его генератор расширяем, и он написал два отчета с его помощью: отчет в HTML, считающий среднее и стандартное отклонение, и отчет в Markdown, считающий медианы. Однако, что делать, если нужно посчитать медианы и вывести результат в HTML? Что, если нужен будет третий отчет в HTML? Текущее решение крайне неудобно для таких ситуаций. Помогите N отрефакторить код, переведя его с наследования на делегирование. Разделите ответственности по оформлению отчета и по вычислению показателей. В результате сам класс ReportMaker вам, возможно, уже и не понадобится.
13	Делегаты и их производные (такие, как события) можно использовать для замены классического объектно-ориентированного шаблона Наблюдатель. Этот шаблон реализован в проекте Delegates.Observers в файле ObservableStack.cs в классическом виде, так как он описан в книжках. Видно, как много инфраструктурного кода необходимо для обеспечения очень несложной функциональности в чистом ООП. Вспомните, что такое событие в языке C#, и отрефакторьте код с его использованием.
14	Часто делегаты можно использовать для тонкой настройки алгоритмов, что позволит использовать один и тот же код для решения несколько разных задач.

	Скачайте проект проекте Delegates.TreeTraversal Перед вами три задачи: 1. Дано дерево категорий продуктов, в каждой категории могут быть другие категории и собственно продукты. Вам нужно вывести список продуктов. 2. Дано дерево задача, каждая задача может содержать подзадачи. Вам нужно вывести список таких задач, у которых нет подзадач. 3. Дано бинарное дерево, у которого каждый лист содержит величину, а каждый не-лист не содержит величины. Вам нужно вывести все величины, содержащиеся в этом дереве. Вам нужно написать *один* алгоритм обхода дерева, который бы принимал в качестве аргументов делегаты, объясняющие алгоритму, как обходить дерево и какие величины выводить. Слишком сложные делегаты могут затруднять чтение кода, поэтому из всего многообразия решения выберите решение, максимально понятное неподготовленному читателю. После этого вам нужно написать реализации методов, указанных в тестах, так, чтобы тесты заработали.
15	Делегаты позволяют нам в ряде случаев обходится вообще без создания классов. Скачайте проект Delegates.PairsAnalysis. Метод Analysis.FindMaxPeriodIndex(params DateTime[] data) должен последовательно разбить список дат по парам и вернуть индекс пары с наибольшим периодом между датами. Видно, как много классов приходится писать и какую нетривиальную систему наследования использовать. Избавьтесь от лишних классов и переведите вычисления в LINQ-стиль, разработав два метода расширения • Pairs, который превращает последовательность T в последовательность Tuple<T,T> • MaxIndex, который ищет индекс максимального элемента. В классе Analysis перепишите два метода так, чтобы они использовали Pairs и MaxIndex, а три вспомогательных класса из проекта перестали использоваться.
16	Скачайте проект Ddd.Taxi. Все Value-типы, согласно DDD, должны поддерживать семантику значений, то есть сравниваться по содержимому своих свойств. Каждый раз реализовывать Equals, GetHashCode и ToString соответствующим образом — довольно муторное занятие. Часто для этого создают базовый класс, наследование от которого реализует нужным образом все эти стандартные методы. Это вам и предстоит сделать! В рамках этого задания сравнивать Value-типы можно только по значению их публичных свойств, без учета значения полей. Хотя как правильно это стоит делать на практике — вопрос дискуссионный и, скорее, предмет договорённостей в вашей команде. В файле Infrastructure/ValueType.cs реализуйте класс ValueType так, чтобы проходили все тесты в файле ValueType_Tests.cs.
17	Продолжайте в том же проекте Ddd.Taxi.

	Изучите пару классов TaxiOrder и TaxiApi — это модель предметной области по заказу такси. TaxiOrder — типичная анемичная модель. Вся логика, связанная с этим классом находится в TaxiApi. Переработайте класс TaxiOrder согласно принципам DDD. А именно: • Сгруппируйте связанные свойства TaxiOrder во вспомогательные классы: PersonName, Address, Driver. Для этого можно как использовать уже готовые классы из пространства имён Domain, так и создавать собственные классы. При проверке этой задачи ValueType из предыдущей задачи будет доступен — используйте его! • Перенесите всю логику из методов TaxiApi в методы TaxiOrder. В TaxiApi должен остаться только тестовый код и вызовы соответствующих методов TaxiOrder. • Добавьте проверки на валидность действий — кидайте InvalidOperationException, если действие в данном состоянии заказа невалидно. • Замените все поля на свойства и закройте у свойств сеттеры. Впрочем, можно закрыть и сами свойства. Проверяйте результат запуском тестов. При переработке кода, публичный интерфейс TaxiApi нужно сохранить, чтобы система тестирования смогла проверить результат вашего рефакторинга. Однако, в реальном проекте переработка TaxiOrder наверняка повлекла бы за собой переработку и TaxiApi.
18	Скачайте проект SRP.ControlDigit. В серийные номера, номера счетов и коды продуктов обычно включают так называемый контрольный разряд, значение которого вычисляется по остальным цифрам номера. Он нужен, чтобы подтверждать отсутствие ошибок при вводе этих номеров вручную или при считывании их с помощью сканеров. Есть несколько стандартизированных алгоритмов вычисления контрольного разряда. Прочитать их краткое описание с примерами можно в соответствующей статье википедии. В проекте уже реализован один метод UPC. Причем реализована без какой-либо декомпозиции. Вам нужно реализовать две других функции, но это не главное. Главное, провести декомпозицию имеющегося кода, разбив его на небольшие повторно используемые функции. В вашем решении, в классе ControlDigitAlgo должны остаться всего три коротких метода — по одному на каждый алгоритм рассчёта. Все остальные функции должны быть полезны, понятны и самодостаточны вне контекста этой задачи и находится в виде методов расширения в классе Extensions. В этой задаче действует ограничение на длину методов — не более 10 строк. Но не надо пытаться записывать сразу много операторов в одной строке — это дурной тон в программировании. После окончания ещё раз посмотрите на сигнатуры всех вспомогательных методов. Точно ли они будут понятны другому программисту без подглядывания в их код? Считайте, что эти методы не будут вызываться слишком часто, поэтому не нужно пытаться их оптимизировать, вместо этого сосредоточьтесь на простоте, понятности и возможности повторного использования.

4. Порядок проведения и критерии оценивания промежуточной аттестации

4.1.Порядок проведения промежуточной аттестации

Зачёт выставляется на основе баллов, набранных студентом в ходе выполнения курса. Студент должен ответить на вопросы закрытого типа, которые предполагают выбор вариантов ответа, а также решить практические задания. Данные решения должны пройти этап автоматического тестирования, а затем код-ревью преподавателя, с последующей защитой.

4.2.Критерии оценивания промежуточной аттестации по видам оценочных средств

4.2.1. Критерии оценивания теста

Тест формируется в системе электронного обучения MOODLE. Максимальный балл за тест — 100 баллов.

Оценка	Зачтено	Незачтено
Баллы	100-60 баллов	59-0 баллов
Уровень освоения проверяемых компетенций	высокий	низкий

4.2.2. Критерии оценивания практического задания

Задания представлены в МООС-системе ulearn, используемой в сотрудничестве с СКБ Контур. Максимальный балл за курс составляет—1135.

Оценка	Зачтено			Незачтено
Баллы	300–550 баллов	550–800	800+	0–300 баллов
Уровень освоения проверяемых компетенций	средний	высокий	очень высокий	низкий

4.3.Результаты промежуточной аттестации и уровни сформированности компетенций

При подведении итогов учитываются результаты только промежуточной аттестации:

0-299 баллов – незачет;
300+ баллов – зачет;

Особенности проведении процедуры оценивания результатов обучения инвалидов и лиц с ограниченными возможностями здоровья обозначены в рабочей программе дисциплины (модуля).

Уровни сформированности компетенций определяется следующим образом:

1. Высокий уровень сформированности компетенций соответствует оценке зачтено:

- предполагает формирование компетенций на высоком уровне;
- знание теоретических разделов изучаемой дисциплины на уровне не ниже оценки удовлетворительно;
- студент умеет применять на практике знания, полученные в рамках изучения дисциплины
- формируются навыки использования теоретических и практических разделов дисциплины для решения задач профессиональной деятельности;

2. Низкий уровень соответствует оценке незачтено.