

Документ подписан простой электронной подписью Информация о владельце: ФИО: Таскаев Сергей Валерьевич Должность: Ректор Дата подписания: 26.06.2026 11:17:29	МИНОБНАУКИ РОССИИ Федеральное государственное бюджетное образовательное учреждение высшего образования «Челябинский государственный университет» (ФГБОУ ВО «ЧелГУ»)	
Фонд оценочных средств для промежуточной аттестации по дисциплине "Проектирование приложений на языке C#" по направлению подготовки (специальности) "09.03.03 Прикладная информатика" направленности (профилю) ИТ-решения и технологии обработки данных в экономике ФГБОУ ВО «ЧелГУ»		стр. 1

**Фонд оценочных средств для промежуточной аттестации по дисциплине**

**Проектирование приложений на языке C#**

**Направление подготовки (специальность)**

**09.03.03 Прикладная информатика**

**Направленность (профиль)**

**ИТ-решения и технологии обработки данных в экономике**

**Присваиваемая квалификация (степень)**

**бакалавр**

**Форма обучения**

**заочная форма обучения**

**Год(ы) набора 2026**

**Челябинск 2026 г.**

**09.03.03 Прикладная информатика профиль ИТ-решения и технологии обработки данных в экономике, дисциплина Проектирование приложений на языке С#, 2026 год набора, заочная форма обучения**

**Фонд оценочных средств дисциплины (модуля) одобрен и рекомендован:**

Проректор по учебной работе                      утверждено 27.02.2026                      А.А. Саламатов

Ученым советом института информационных технологий

Протокол заседания № 7 от 26.02.2026

Председатель Ученого совета  
института информационных  
технологий

согласовано

Ю.В. Петриченко

**Заседанием кафедры информационных технологий и экономической информатики**

Протокол заседания №7 от 26.02.2026

Заведующий кафедрой

согласовано

С.А. Скрипов

Автор (составитель)

С.А. Скрипов

**Структура фондов оценочных средств соответствует приказу ректора ФГБОУ ВО «ЧелГУ» от 27 сентября 2022 № 573-1**



## Содержание

1. Паспорт фонда оценочных средств .....	3
2. Перечень формируемых компетенций .....	4
3. Содержание оценочных средств по дисциплине .....	5
3.1. Виды оценочных средств .....	5
3.2. Содержание оценочных средств .....	5
4. Порядок проведения и критерии оценивания промежуточной аттестации .....	16
4.1. Порядок проведения промежуточной аттестации .....	16
4.2. Критерии оценивания промежуточной аттестации по видам оценочных средств .....	16
4.3. Результаты промежуточной аттестации и уровни сформированности компетенций .....	16



## 1. Паспорт фонда оценочных средств

Направление подготовки: 09.03.03 Прикладная информатика

Направленность: ИТ-решения и технологии обработки данных в экономике

Дисциплина: Проектирование приложений на языке C#

Курс (курсы) изучения: 3

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 и 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-7	Способен разрабатывать алгоритмы и программы, пригодные для практического применения;	ОПК-7.1. Демонстрирует знание основ информатики, теории алгоритмов, методологии и технологии программирования ОПК-7.2. Демонстрирует умения разрабатывать алгоритмические и программные решения ОПК-7.3. 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: навыки использования объектно-ориентированных технологий разработки программного обеспечения



### 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-7.1. 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-ориентированное программирование Проектирование объектно-ориентированных приложений на языке C#	Тест	Задания теста № 1-20
2	ОПК-7.2. Демонстрирует умения разрабатывать алгоритмические и программные решения Уметь: разрабатывать алгоритмические и программные решения на языке C#	Синтаксис языка C#. Объектно-ориентированное программирование Проектирование объектно-ориентированных приложений на языке C#	Тест	Практические задания 1-18
3	ОПК-7.3. Имеет практический опыт использования технологий разработки программного обеспечения Владеть: навыки использования объектно-ориентированных технологий разработки программного обеспечения	Синтаксис языка C#. Объектно-ориентированное 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.	<pre>class SomeClass { public static int s; private int p; public int d; }</pre>	<ul style="list-style-type: none"><li>• <b>SomeClass.s = 42</b></li><li>• SomeClass.d = 42</li><li>• new SomeClass().s = 42</li><li>• <b>new SomeClass().d</b></li></ul>



	Отметьте все корректные обращения к полям объявленного выше класса SomeClass	<p>= 42</p> <ul style="list-style-type: none"><li>• new SomeClass().p = 42</li></ul>
2.	<pre>namespace MyNamespace {     internal class ClassA { }      public class ClassB     {         public ClassA Method1() { return null; }         private ClassB Method2() { return null; }     } }</pre> <p>Перечислите все способы, которыми можно избавиться от ошибок компиляции в этом коде</p>	<ul style="list-style-type: none"><li>• <b>Перенести этот код в другую сборку</b></li><li>• <b>Сменить модификатор доступа ClassB с public на internal</b></li><li>• Сменить модификатор доступа Method2 с private на public</li><li>• <b>Сменить модификатор доступа Method1 с public на private</b></li><li>• <b>Сменить модификатор доступа ClassA с internal на public</b></li><li>• Сменить модификатор доступа Method1 с public на internal</li></ul>
3.	<pre>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;     } }</pre> <p>Конверсия переменной b к ClassA — это</p>	<ul style="list-style-type: none"><li>• <b>Upcast</b></li><li>• Downcast</li><li>• Ни то, ни другое</li></ul>
4.	Конверсия переменной c к ClassA	<ul style="list-style-type: none"><li>• <b>Пройдет без ошибок</b></li><li>• Вызовет ошибку компиляции</li><li>• Вызовет ошибку выполнения</li></ul>
5.	Конверсия переменной a к ClassB - это	<ul style="list-style-type: none"><li>• Upcast</li><li>• <b>Downcast</b></li><li>• Ни то, ни другое</li></ul>
6.	Конверсия переменной d к ClassC	<ul style="list-style-type: none"><li>• Пройдет без ошибок</li><li>• Вызовет ошибку компиляции</li><li>• <b>Вызовет ошибку</b></li></ul>



		<b>выполнения</b>
7.	К каким типам можно без ошибок привести переменную d?	<ul style="list-style-type: none"><li>• <b>ClassA</b></li><li>• <b>ClassB</b></li><li>• <b>ClassC</b></li></ul>
8.	Конверсия переменной c к ClassB	<ul style="list-style-type: none"><li>• <b>Upcast</b></li><li>• <b>Downcast</b></li><li>• <b>Ни то, ни другое</b></li></ul>
9.	<pre>interface A { } interface B : A { }  class X : A { } class Y : X, B { } class Z : X { }</pre> Какие из этих следующих операторов не выбросят исключение?	<ul style="list-style-type: none"><li>• <b>Y as B</b></li><li>• <b>X as B</b></li><li>• <b>X</b></li><li>• <b>Y</b></li><li>• <b>(B) Z</b></li></ul>
10.	Какие из этих утверждений верны для абстрактных базовых классов, но не верны для интерфейсов	<ul style="list-style-type: none"><li>• <b>Могут содержать поля</b></li><li>• <b>Могут содержать реализацию методов базового класса или интерфейса</b></li><li>• Могут содержать свойства</li><li>• <b>Могут содержать методы с реализованной логикой</b></li><li>• <b>Могут содержать приватные члены</b></li></ul>
11.	Как вы думаете, зачем может быть полезно разделение кода на слои?	<ul style="list-style-type: none"><li>• <b>Проще навигация по коду — новичку проще понять, где искать тот или иной класс</b></li><li>• <b>Появляется возможность повторного использования слоя предметной области в разных приложениях</b></li><li>• Без разделения на слои, код нельзя будет протестировать</li><li>• Код, для которого критична производительность, можно вынести на более низкий уровень, и он будет работать быстрее</li></ul>
12.	В каком слое должен оказаться класс, описывающий контрол Windows Forms, для отображения информации о пациенте?	<ul style="list-style-type: none"><li>• <b>UI</b></li><li>• <b>Application</b></li><li>• <b>Domain</b></li></ul>



		<ul style="list-style-type: none"><li>• infrastructure</li></ul>
13.	В каком слое должен оказаться вспомогательный метод расширения класса FlightLevel, определяющего каким курсом движется воздушное судно, преимущественно северным или южным?	<ul style="list-style-type: none"><li>• UI</li><li>• Application</li><li>• Domain</li><li>• <b>infrastructure</b></li></ul>
14.	Отметьте верные утверждения про разделение кода на слои согласно DDD	<ul style="list-style-type: none"><li>• Классы, описывающие предметную область должны быть в слое приложения</li><li>• <b>Слой предметной области может зависеть только от слоя инфраструктуры</b></li><li>• Каждый слой может зависеть только от одного следующего слоя</li><li>• Каждый слой может зависеть от любого другого слоя.</li></ul>
15.	Как разделение на слои может выглядеть в коде на C#	<ul style="list-style-type: none"><li>• Выделять принадлежность каждого члена к тому или иному слою комментарием</li><li>• Помещать члены класса, относящиеся к слою X в отдельный регион с именем X</li><li>• <b>Создать по отдельной сборке на каждый слой</b></li><li>• <b>Создать по отдельному пространству имен на каждый слой</b></li></ul>
16.	Какой пункт НЕ является отличительной особенностью сущности (Entity)?	<ul style="list-style-type: none"><li>• При изменении свойств, у сущности сохраняется идентичность</li><li>• У сущности есть идентификатор, по которому ее можно идентифицировать, например, при загрузке из хранилища</li><li>• Две сущности одного типа считаются равными, если у них равны идентификаторы</li></ul>



		<ul style="list-style-type: none"><li>• <b>Поля сущности не могут быть Value-объектами</b></li><li>• Действия с сущностью как правило оформляют в виде методов класса этой сущности</li></ul>
17.	Какой пункт НЕ является отличительной особенностью объекта-значения (Value-object)?	<ul style="list-style-type: none"><li>• Значение как правило стараются делать неизменяемым классом</li><li>• <b>У Значений как правило нет методов, а есть только свойства</b></li><li>• Два значения одного типа считаются равными, если у них равны все их поля</li><li>• Поля объекта-значения не могут быть сами объектами значениями</li></ul>
18.	Какой пункт НЕ является отличительной особенностью сервиса (Service)?	<ul style="list-style-type: none"><li>• <b>Все операции с сущностями и значениями нужно оформлять в виде методов классов-сервисов</b></li><li>• Обычно сервисы создаются в начале программы в единственном экземпляре</li><li>• Сервисы содержат операции над классами предметной области, которые по какой-то причине не стали методами этих классов</li></ul>
19.	<pre>public class Point {     public readonly int X, Y;     public Point(int x, int y)     {         X = x;         Y = y;     }     public override bool Equals(object other)</pre>	<ul style="list-style-type: none"><li>• Entity-объектом</li><li>• Сервисом</li><li>• <b>Value-объектом</b></li></ul>



	<pre>{     var p = other as Point;     return p != null &amp;&amp; p.X == X &amp;&amp; p.Y == Y; }  public override int GetHashCode() =&gt; X ^ Y; }</pre> <p>Чем является класс Point?</p>	
20.	Point — это пример богатой (rich) или анемичной (anemic) модели?	<ul style="list-style-type: none"><li>• Богатой</li><li>• <b>Анемичной</b></li></ul>

База практических заданий

№ п/п	Формулировка задания
1	<p>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.</p> <p>Скачайте <a href="#">проект Incapsulation.Failures</a> и помогите N. отрефакторить его код:</p> <ol style="list-style-type: none"><li>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&lt;List&lt;object&gt;&gt;.</li><li>2. Значения в аргументах devices и failureTypes должны быть инкапсулированы в сущности Device и Failure.</li><li>3. IsFailureSerious, очевидно, не на своем месте.</li><li>4. С day и times тоже не все в порядке.</li></ol> <p>Сигнатуру старого метода сохраните, чтобы проходили тесты. Старый метод должен преобразовывать аргументы и вызывать новый метод.</p>
2	<p>Некто M. 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.</p> <p>Скачайте <a href="#">проект Incapsulation.EnterpriseTask</a> и помогите M. отрефакторить его код.</p>
3	<p>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.</p> <p>Скачайте <a href="#">проект Incapsulation.Weights</a> 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.</p>
4	<p>При работе над математическими или геометрическими задачами часто приходится создавать "фундаментальные" классы, подобные int или double, для комплексных чисел, кватернионов и т.д. Скачайте <a href="#">проект Incapsulation.RationalNumbers</a> и напишите класс рационального числа.</p> <p>Ваше решение должно проходить тесты, содержащиеся в проекте.</p>
5	<p>Простейший сценарий, когда вам нужна перегрузка методов и реализация интерфейсов - написание небольших структур данных, которые должны быть совместимы с листами, словарями и т.д.</p> <p>Допустим, вы разрабатываете систему для анализа сообщений в техподдержку, и хотите классифицировать их по имени продукту, типу сообщения и его теме. Соответственно, вам</p>



	<p>необходим класс, обозначающий категорию сообщений с указанными полями.</p> <p>Скачайте <a href="#">проект Inheritance.DataStructure</a> 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.</p>
6	<p>В компьютерной игре, персонаж игрока взаимодействует с различными объектами на карте. Есть всего три способа взаимодействовать:</p> <ol style="list-style-type: none"><li>1. Сражаться с армией.</li><li>2. Собирать сокровища.</li><li>3. Присваивать объект себе.</li></ol> <p>А вот различных видов объектов на карте уже 5, а будет ещё больше. Скачайте <a href="#">проект Inheritance.MapObjects</a>, откройте файл Task.cs и изучите, как это реализовано сейчас.</p> <p>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.</p> <p>Выделите все поля, необходимую для каждого взаимодействия, в свой интерфейс. Отрефакторьте программу, избавившись от повторяющихся участков кода в Interaction.Make.</p> <p>В итоговом решении Interaction.Make должен работать только с интерфейсами, и не должен содержать упоминаний конкретных классов.</p>
7	<p>Какое же наследование без геометрии!</p> <p>Скачайте <a href="#">проект Inheritance.Geometry</a> и изучите Task.cs. Проблема этого подхода в том, что каждый раз при добавлении нового типа тела придется менять метод в базовом классе.</p> <p>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.</p> <p>Сделайте это!</p> <p>В финальном решении не должно быть ни одного приведения типа.</p>
8	<p>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.</p> <p>В этом случае часто используется шаблон Visitor. <a href="#">Изучите этот шаблон по википедии.</a></p> <p>Определите интерфейс IVisitor и реализуйте его в двух классах DimensionsVisitor и SurfaceAreaVisitor, для расчёта размеров (ширина, высота) и площади поверхности фигур.</p> <p>В класс Body добавьте абстрактный метод Accept(IVisitor visitor).</p> <p>Автоматизированные тесты проверяют лишь базовые требования. Проверить, что вы всё сделали правильно можно самостоятельно так:</p> <ol style="list-style-type: none"><li>1. В реализациях Visitor не должно быть ни одного приведения типов и ни одного if-a.</li></ol>



	<p>Именно этой простотой решение с Visitor-ом лучше исходного с длинным if-else.</p> <ol style="list-style-type: none"><li>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.</li><li>3. Компилятор должен контролировать, что вы не забыли обработать ни одну из фигур: если вы забудете написать один из методов, программа даже не скомпилируется.</li><li>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 расчетов, эти классы не нужно будет модифицировать.</li><li>5. Для добавления нового метода работы с фигурами, должно быть достаточно добавить новый класс Visitor-a.</li></ol>
9	<p>Наиболее очевидный случай использования дженериков — создание коллекций. Скачайте <a href="#">проект Generics.BinaryTrees</a> и создайте в нем класс <a href="#">бинарного дерева поиска</a> так, чтобы он проходил приложенные тесты.</p> <p>Если у вас останется много времени, оптимизируйте код метода GetEnumerator так, чтобы он работал за <math>O(n)</math> по времени, где <math>n</math> — количество элементов в дереве, и <math>O(1)</math> по памяти. Если времени не останется, вы можете использовать менее оптимальное решение.</p>
10	<p>В анализе данных бывают очень полезны таблицы. Например, строки могут соответствовать датам, столбцы — департаментам, а в ячейках может храниться выручка департамента за контракты на соответствующую дату.</p> <p>Сделайте такую таблицу, которая бы:</p> <ol style="list-style-type: none"><li>1. Индексировалась величинами типов, указанных при создании таблицы.</li><li>2. Имела бы две возможности для индексирования:<ol style="list-style-type: none"><li>1. С автоматическим созданием нужных строк и столбцов «на лету» при обращении к таблице по соответствующим индексам;</li><li>2. Которая бы требовала создания столбцов и строк заранее и выбрасывала исключение при доступе к несуществующим столбцам или строкам.</li></ol></li></ol> <p>Скачайте <a href="#">проект Generics.Tables</a> и изучите тесты, которые должна проходить ваша таблица, чтобы понять детали задания.</p> <p>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.</p>
11	<p>Не так-то просто сделать упражнение на ковариацию и контравариацию, но нам удалось.</p> <p>Скачайте <a href="#">проект Generics.Robots</a> и изучите класс Architecture.cs. Он описывает некий проект архитектуры робота. В нем есть AI, вырабатывающий команды, и Device, команды исполняющий.</p> <p>При этом, AI уже готовы для двух типов роботов (Builder и Shooter), а Device есть только для подвижной части.</p> <p>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.</p>
12	<p>Некто N разработал <a href="#">генератор отчетов</a> 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</p>



	<p>отклонение, и отчет в Markdown, считающий медианы.</p> <p>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.</p> <p>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.</p>
13	<p>Делегаты и их производные (такие, как события) можно использовать для замены классического объектно-ориентированного шаблона <u>Наблюдатель</u>. Этот шаблон реализован в <u>проекте Delegates.Observers</u> в файле ObservableStack.cs в классическом виде, так как он описан в книжках.</p> <p>Видно, как много инфраструктурного кода необходимо для обеспечения очень несложной функциональности в чистом ООП.</p> <p>Вспомните, что такое событие в языке C#, и отрефакторьте код с его использованием.</p>
14	<p>Часто делегаты можно использовать для тонкой настройки алгоритмов, что позволит использовать один и тот же код для решения несколько разных задач.</p> <p>Скачайте проект <u>проекте Delegates.TreeTraversal</u></p> <p>Перед вами три задачи:</p> <ol style="list-style-type: none"><li>1. Дано дерево категорий продуктов, в каждой категории могут быть другие категории и собственно продукты. Вам нужно вывести список продуктов.</li><li>2. Дано дерево задача, каждая задача может содержать подзадачи. Вам нужно вывести список таких задач, у которых нет подзадач.</li><li>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.</li></ol> <p>Вам нужно написать <i>один</i> алгоритм обхода дерева, который бы принимал в качестве аргументов делегаты, объясняющие алгоритму, как обходить дерево и какие величины выводить.</p> <p>Слишком сложные делегаты могут затруднять чтение кода, поэтому из всего многообразия решения выберите решение, максимально понятное неподготовленному читателю.</p> <p>После этого вам нужно написать реализации методов, указанных в тестах, так, чтобы тесты заработали.</p>
15	<p>Делегаты позволяют нам в ряде случаев обходиться вообще без создания классов. Скачайте проект <u>Delegates.PairsAnalysis</u>.</p> <p>Метод Analysis.FindMaxPeriodIndex(params DateTime[] data) должен последовательно разбить список дат по парам и вернуть индекс пары с наибольшим периодом между датами.</p> <p>Видно, как много классов приходится писать и какую нетривиальную систему наследования использовать.</p> <p>Избавьтесь от лишних классов и переведите вычисления в LINQ-стиль, разработав два метода расширения</p> <ul style="list-style-type: none"><li>• Pairs, который превращает последовательность T в последовательность Tuple&lt;T,T&gt;</li></ul>



	<ul style="list-style-type: none"><li>• <code>MaxIndex</code>, который ищет индекс максимального элемента.</li></ul> <p>В классе <code>Analysis</code> перепишите два метода так, чтобы они использовали <code>Pairs</code> и <code>MaxIndex</code>, а три вспомогательных класса из проекта перестали использоваться.</p>
16	<p>Скачайте проект <a href="#">Ddd.Taxi</a>.</p> <p>Все Value-типы, согласно DDD, должны поддерживать семантику значений, то есть сравниваться по содержимому своих свойств. Каждый раз реализовывать <code>Equals</code>, <code>GetHashCode</code> и <code>ToString</code> соответствующим образом — довольно мутное занятие. Часто для этого создают базовый класс, наследование от которого реализует нужным образом все эти стандартные методы. Это вам и предстоит сделать!</p> <p>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.</p> <p>В файле <code>Infrastructure/ValueType.cs</code> реализуйте класс <code>ValueType</code> так, чтобы проходили все тесты в файле <code>ValueType_Tests.cs</code>.</p>
17	<p>Продолжайте в том же проекте <a href="#">Ddd.Taxi</a>.</p> <p>Изучите пару классов <code>TaxiOrder</code> и <code>TaxiApi</code> — это модель предметной области по заказу такси.</p> <p><code>TaxiOrder</code> — типичная анемичная модель. Вся логика, связанная с этим классом находится в <code>TaxiApi</code>.</p> <p>Переработайте класс <code>TaxiOrder</code> согласно принципам DDD. А именно:</p> <ul style="list-style-type: none"><li>• Сгруппируйте связанные свойства <code>TaxiOrder</code> во вспомогательные классы: <code>PersonName</code>, <code>Address</code>, <code>Driver</code>. Для этого можно как использовать уже готовые классы из пространства имён <code>Domain</code>, так и создавать собственные классы. При проверке этой задачи <code>ValueType</code> из предыдущей задачи будет доступен — используйте его!</li><li>• Перенесите всю логику из методов <code>TaxiApi</code> в методы <code>TaxiOrder</code>. В <code>TaxiApi</code> должен остаться только тестовый код и вызовы соответствующих методов <code>TaxiOrder</code>.</li><li>• Добавьте проверки на валидность действий — кидайте <code>InvalidOperationException</code>, если действие в данном состоянии заказа невалидно.</li><li>• Замените все поля на свойства и закройте у свойств сеттеры. Впрочем, можно закрыть и сами свойства.</li></ul> <p>Проверяйте результат запуском тестов.</p> <p>При переработке кода, публичный интерфейс <code>TaxiApi</code> нужно сохранить, чтобы система тестирования смогла проверить результат вашего рефакторинга. Однако, в реальном проекте переработка <code>TaxiOrder</code> наверняка повлекла бы за собой переработку и <code>TaxiApi</code>.</p>
18	<p>Скачайте проект <a href="#">SRP.ControlDigit</a>.</p> <p>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.</p> <p>Есть несколько стандартизированных алгоритмов вычисления контрольного разряда. Прочитать их краткое описание с примерами можно в соответствующей <a href="#">статье википедии</a>.</p>



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 расчёта. 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 MOOC-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 проведения 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.