МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

ФЕДЕРАЛЬНОЕ Государственное БЮДЖЕТНОЕ образовательное учреждение высшего образования

«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет прикладной математики, информатики и механики

Кафедра программного обеспечения и

администрирования информационных систем

Вощинская Г.Э., Лещенко Е.М.

**Структуры и алгоритмы обработки данных**

Часть 2

*Учебно-методическое пособие*

ВГУ 2019

Утверждено Научно-Методическим Советом факультета прикладной математики, информатики и механики от 16. 09. 2019

Учебно-методическое пособие подготовлено на кафедре программного обеспечения и администрирования информационных систем факультета прикладной математики, информатики и механики Воронежского государственного университета.

Рекомендовано для студентов, обучающихся по направлению 02.03.03 – Математическое обеспечение и администрирование информационных систем для проведения лабораторных занятий по предмету «Программирование на C#».

Содержание

[Задание 1 4](#_Toc20693984)

[Задание 2 45](#_Toc20693985)

[Список литературы 51](#_Toc20693986)

# Задание 1

Обобщенное программирование

Задача направлена на создание обобщенных классов, описывающих различные структуры данных, с применением итераторов, делегатов и исключений. Класс должен содержать методы для отображения его значений. Для каждой задачи также необходимо создать диаграмму классов, поясняющую архитектуру проекта. Использовать встроенную функциональность коллекций в .NET FCL запрещено.

1. **Список**

Разработать библиотеку обобщенных классов для работы со списками данных. В структуру классов входят как минимум:

* + IList<T>: IEnumerable<T> – базовый интерфейс для всех списков;
    - методы:
      * int Add(T value);
      * void Clear();
      * bool Contains(T value);
      * int IndexOf(T value);
      * void Insert(int index, T value);
      * void Remove(T value);
      * void RemoveAt(int index);
      * IList<T> subList(int fromIndex, int toIndex);
    - свойства:
      * int Count;
      * T this[int index];
  + ListException – класс, описывающий исключения, которые могут происходить в ходе работы со списком (также можно написать ряд наследников);
  + ArrayList<T>: IList<T> – класс списка на основе массива;
  + LinkedList<T>: IList<T> – класс списка на основе связного списка;
  + UnmutableList<T>: IList<T> – класс неизменяющегося списка, является оберткой над любым существующим списком (должен кидаться исключениями на вызов любого метода, изменяющего список);
  + ListUtils – класс различных операций над списком;
    - методы:
      * static bool Exists<T>(IList<T>, CheckDelegate<T>);
      * static T Find<T>(IList<T>, CheckDelegate<T>);
      * static T FindLast<T>(IList<T>, CheckDelegate<T>);
      * static int FindIndex<T>(IList<T>, CheckDelegate<T>);
      * static int FindLastIndex<T>(IList<T>, CheckDelegate<T>);
      * static IList<T> FindAll<T>(IList<T>, CheckDelegate<T>, ListConstructorDelegate<T>);
      * static IList<TO> ConvertAll<TI, TO>(IList<TI>, ConvertDelegate<TI, TO>, ListConstructorDelegate<TO>);
      * static void ForEach(IList<T>, ActionDelegate<T>);
      * static void Sort(IList<T>, CompareDelegate<T>);
      * static bool CheckForAll<T>(IList<T>, CheckDelegate<T>);
    - свойства:
      * static readonly ListConstructorDelegate<T> ArrayListConstructor;
      * static readonly ListConstructorDelegate<T> LinkedListConstructor;

Также необходимо разработать серию примеров, демонстрирующих основные аспекты работы с данной библиотекой списков.

1. **Кольцевой список**

Разработать библиотеку обобщенных классов для работы с кольцевыми списками данных. В структуру классов входят как минимум:

* + IList<T>: IEnumerable<T> – базовый интерфейс для всех кольцевых списков;
    - методы:
      * int Add(T value);
      * void Clear();
      * bool Contains(T value);
      * int IndexOf(T value);
      * void Insert(int index, T value);
      * void Remove(T value);
      * void RemoveAt(int index);
      * IList<T> subList(int fromIndex, int toIndex);
    - свойства:
      * int Count;
      * T this[int index];
  + ListException – класс, описывающий исключения, которые могут происходить в ходе работы с кольцевым списком (также можно написать ряд наследников);
  + ArrayList<T>: IList<T> – класс кольцевого списка на основе массива;
  + LinkedList<T>: IList<T> – класс кольцевого списка на основе связного списка;
  + UnmutableList<T>: IList<T> – класс неизменяющегося кольцевого списка, является оберткой над любым существующим списком (должен кидаться исключениями на вызов любого метода, изменяющего список);
  + ListUtils – класс различных операций над кольцевым списком;
    - методы:
    - static bool Exists<T>(IList<T>, CheckDelegate<T>);
    - static T Find<T>(IList<T>, CheckDelegate<T>);
    - static T FindLast<T>(IList<T>, CheckDelegate<T>);
    - static int FindIndex<T>(IList<T>, CheckDelegate<T>);
    - static int FindLastIndex<T>(IList<T>, CheckDelegate<T>);
    - static IList<T> FindAll<T>(IList<T>, CheckDelegate<T>, ListConstructorDelegate<T>);
    - static IList<TO> ConvertAll<TI, TO>(IList<TI>, ConvertDelegate<TI, TO>, ListConstructorDelegate<TO>);
    - static void ForEach(IList<T>, ActionDelegate<T>);
    - static void Sort(IList<T>, CompareDelegate<T>);
    - static bool CheckForAll<T>(IList<T>, CheckDelegate<T>);
    - свойства:
      * static readonly ListConstructorDelegate<T> ArrayListConstructor;
      * static readonly ListConstructorDelegate<T> LinkedListConstructor;

Также необходимо разработать серию примеров, демонстрирующих основные аспекты работы с данной библиотекой кольцевых списков.

1. **Двусвязный список**

Разработать библиотеку обобщенных классов для работы с двусвязными списками данных. В структуру классов входят как минимум:

* + IList<T>: IEnumerable<T> – базовый интерфейс для всех двусвязных списков;
    - методы:
      * int Add(T value);
      * void Clear();
      * bool Contains(T value);
      * int IndexOf(T value);
      * void Insert(int index, T value);
      * void Remove(T value);
      * void RemoveAt(int index);
      * IList<T> subList(int fromIndex, int toIndex);
    - свойства:
      * int Count;
      * T this[int index];
  + ListException – класс, описывающий исключения, которые могут происходить в ходе работы с двусвязным списком (также можно написать ряд наследников);
  + ArrayList<T>: IList<T> – класс двусвязного списка на основе массива;
  + LinkedList<T>: IList<T> – класс двусвязного списка на основе связного списка;
  + UnmutableList<T>: IList<T> – класс неизменяющегося двусвязного списка, является оберткой над любым существующим списком (должен кидаться исключениями на вызов любого метода, изменяющего список);
  + ListUtils – класс различных операций над двусвязным списком;
    - методы:
      * static bool Exists<T>(IList<T>, CheckDelegate<T>);
      * static T Find<T>(IList<T>, CheckDelegate<T>);
      * static T FindLast<T>(IList<T>, CheckDelegate<T>);
      * static int FindIndex<T>(IList<T>, CheckDelegate<T>);
      * static int FindLastIndex<T>(IList<T>, CheckDelegate<T>);
      * static IList<T> FindAll<T>(IList<T>, CheckDelegate<T>, ListConstructorDelegate<T>);
      * static IList<TO> ConvertAll<TI, TO>(IList<TI>, ConvertDelegate<TI, TO>, ListConstructorDelegate<TO>);
      * static void ForEach(IList<T>, ActionDelegate<T>);
      * static void Sort(IList<T>, CompareDelegate<T>);
      * static bool CheckForAll<T>(IList<T>, CheckDelegate<T>);
    - свойства:
      * static readonly ListConstructorDelegate<T> ArrayListConstructor;
      * static readonly ListConstructorDelegate<T> LinkedListConstructor;

Также необходимо разработать серию примеров, демонстрирующих основные аспекты работы с данной библиотекой двусвязных списков.

1. **Упорядоченный список**

Разработать библиотеку обобщенных классов для работы с упорядоченными списками данных. В структуру классов входят как минимум:

* + IList<T>: IEnumerable<T> – базовый интерфейс для всех упорядоченных списков;
    - методы:
      * int Add(T value);
      * void Clear();
      * bool Contains(T value);
      * int IndexOf(T value);
      * void Insert(int index, T value);
      * void Remove(T value);
      * void RemoveAt(int index);
      * IList<T> subList(int fromIndex, int toIndex);
    - свойства:
      * int Count;
      * T this[int index];
  + ListException – класс, описывающий исключения, которые могут происходить в ходе работы с упорядоченным списком (также можно написать ряд наследников);
  + ArrayList<T>: IList<T> – класс упорядоченного списка на основе массива;
  + LinkedList<T>: IList<T> – класс упорядоченного списка на основе связного списка;
  + UnmutableList<T>: IList<T> – класс неизменяющегося упорядоченного списка, является оберткой над любым существующим списком (должен кидаться исключениями на вызов любого метода, изменяющего список);
  + ListUtils – класс различных операций над упорядоченным списком;
    - методы:
      * static bool Exists<T>(IList<T>, CheckDelegate<T>);
      * static T Find<T>(IList<T>, CheckDelegate<T>);
      * static T FindLast<T>(IList<T>, CheckDelegate<T>);
      * static int FindIndex<T>(IList<T>, CheckDelegate<T>);
      * static int FindLastIndex<T>(IList<T>, CheckDelegate<T>);
      * static IList<T> FindAll<T>(IList<T>, CheckDelegate<T>, ListConstructorDelegate<T>);
      * static IList<TO> ConvertAll<TI, TO>(IList<TI>, ConvertDelegate<TI, TO>, ListConstructorDelegate<TO>);
      * static void ForEach(IList<T>, ActionDelegate<T>);
      * static void Sort(IList<T>, CompareDelegate<T>);
      * static bool CheckForAll<T>(IList<T>, CheckDelegate<T>);
    - свойства:
      * static readonly ListConstructorDelegate<T> ArrayListConstructor;
      * static readonly ListConstructorDelegate<T> LinkedListConstructor;

Также необходимо разработать серию примеров, демонстрирующих основные аспекты работы с данной библиотекой упорядоченных списков.

1. **Частично упорядоченный список**

Разработать библиотеку обобщенных классов для работы с частично упорядоченными списками данных. В структуру классов входят как минимум:

* + IList<T>: IEnumerable<T> – базовый интерфейс для всех частично упорядоченных списков;
    - методы:
      * int Add(T value);
      * void Clear();
      * bool Contains(T value);
      * int IndexOf(T value);
      * void Insert(int index, T value);
      * void Remove(T value);
      * void RemoveAt(int index);
      * IList<T> subList(int fromIndex, int toIndex);
    - свойства:
      * int Count;
      * T this[int index];
  + ListException – класс, описывающий исключения, которые могут происходить в ходе работы с частично упорядоченным списком (также можно написать ряд наследников);
  + ArrayList<T>: IList<T> – класс частично упорядоченного списка на основе массива;
  + LinkedList<T>: IList<T> – класс частично упорядоченного списка на основе связного списка;
  + UnmutableList<T>: IList<T> – класс частично упорядоченного неизменяющегося списка, является оберткой над любым существующим списком (должен кидаться исключениями на вызов любого метода, изменяющего список);
  + ListUtils – класс различных операций над частично упорядоченным списком;
    - методы:
      * static bool Exists<T>(IList<T>, CheckDelegate<T>);
      * static T Find<T>(IList<T>, CheckDelegate<T>);
      * static T FindLast<T>(IList<T>, CheckDelegate<T>);
      * static int FindIndex<T>(IList<T>, CheckDelegate<T>);
      * static int FindLastIndex<T>(IList<T>, CheckDelegate<T>);
      * static IList<T> FindAll<T>(IList<T>, CheckDelegate<T>, ListConstructorDelegate<T>);
      * static IList<TO> ConvertAll<TI, TO>(IList<TI>, ConvertDelegate<TI, TO>, ListConstructorDelegate<TO>);
      * static void ForEach(IList<T>, ActionDelegate<T>);
      * static void Sort(IList<T>, CompareDelegate<T>);
      * static bool CheckForAll<T>(IList<T>, CheckDelegate<T>);
    - свойства:
      * static readonly ListConstructorDelegate<T> ArrayListConstructor;
      * static readonly ListConstructorDelegate<T> LinkedListConstructor;

Также необходимо разработать серию примеров, демонстрирующих основные аспекты работы с данной библиотекой частично упорядоченных списков.

1. **Список с пропусками**

Разработать библиотеку обобщенных классов для работы со списками данных с пропусками. В структуру классов входят как минимум:

* + IList<T>: IEnumerable<T> – базовый интерфейс для всех списков;
    - методы:
      * int Add(T value);
      * void Clear();
      * bool Contains(T value);
      * int IndexOf(T value);
      * void Insert(int index, T value);
      * void Remove(T value);
      * void RemoveAt(int index);
      * IList<T> subList(int fromIndex, int toIndex);
    - свойства:
      * int Count;
      * T this[int index];
  + ListException – класс, описывающий исключения, которые могут происходить в ходе работы со списком с пропусками (также можно написать ряд наследников);
  + ArrayList<T>: IList<T> – класс списка с пропусками на основе массива;
  + LinkedList<T>: IList<T> – класс списка с пропусками на основе связного списка;
  + UnmutableList<T>: IList<T> – класс неизменяющегося списка с пропусками, является оберткой над любым существующим списком (должен кидаться исключениями на вызов любого метода, изменяющего список);
  + ListUtils – класс различных операций над списком с пропусками;
    - методы:
      * static bool Exists<T>(IList<T>, CheckDelegate<T>);
      * static T Find<T>(IList<T>, CheckDelegate<T>);
      * static T FindLast<T>(IList<T>, CheckDelegate<T>);
      * static int FindIndex<T>(IList<T>, CheckDelegate<T>);
      * static int FindLastIndex<T>(IList<T>, CheckDelegate<T>);
      * static IList<T> FindAll<T>(IList<T>, CheckDelegate<T>, ListConstructorDelegate<T>);
      * static IList<TO> ConvertAll<TI, TO>(IList<TI>, ConvertDelegate<TI, TO>, ListConstructorDelegate<TO>);
      * static void ForEach(IList<T>, ActionDelegate<T>);
      * static void Sort(IList<T>, CompareDelegate<T>);
      * static bool CheckForAll<T>(IList<T>, CheckDelegate<T>);
    - свойства:
      * static readonly ListConstructorDelegate<T> ArrayListConstructor;
      * static readonly ListConstructorDelegate<T> LinkedListConstructor;

Также необходимо разработать серию примеров, демонстрирующих основные аспекты работы с данной библиотекой списков с пропусками.

1. **Множество**

Разработать библиотеку обобщенных классов для работы с множествами данных. В структуру классов входят как минимум:

* + ISet<T>: IEnumerable<T> – базовый интерфейс для всех множеств;
    - методы:
      * void Add(T value);
      * void Clear();
      * bool Contains(T value);
      * void Remove(T value);
    - свойства:
      * int Count;
      * bool isEmpty;
  + SetException – класс, описывающий исключения, которые могут происходить в ходе работы с множествами (также можно написать ряд наследников от SetException);
  + ArraySet<T>: ISet<T> – класс множества на основе массива;
  + LinkedSet<T>: ISet<T> – класс множества на основе связного списка;
  + HashSet<T>: ISet<T> – класс множества на основе алгоритма хеширования (использовать методы GetHashCode и Equals типа T);
  + UnmutableSet<T>: ISet<T> – класс неизменяющегося множества, является оберткой над любым существующим множеством (должен кидаться исключениями на вызов любого метода, изменяющего множество);
  + SetUtils – класс различных операций над множеством;
    - методы:
      * static bool Exists<T>(ISet<T>, CheckDelegate<T>);
      * static ISet<T> FindAll<T>(ISet<T>, CheckDelegate<T>, SetConstructorDelegate<T>);
      * static ISet<TO> ConvertAll<TI, TO>(ISet<TI>, ConvertDelegate<TI, TO>, SetConstructorDelegate<TO>);
      * static void ForEach(ISet<T>, ActionDelegate<T>);
      * static bool CheckForAll<T>(ISet<T>, CheckDelegate<T>);
    - свойства:
      * static readonly SetConstructorDelegate<T> ArraySetConstructor;
      * static readonly SetConstructorDelegate<T> LinkedSetConstructor;
      * static readonly SetConstructorDelegate<T> HashSetConstructor;

Также необходимо разработать серию примеров, демонстрирующих основные аспекты работы с данной библиотекой множеств.

1. **Ассоциативное множество**

Разработать библиотеку обобщенных классов для работы с ассоциативными множествами данных. В структуру классов входят как минимум:

* + IMap<K, V>: IEnumerable<IMap.IEntry<K, V>> – базовый интерфейс для всех ассоциативных множеств;
    - интерфейсы:
      * IEntry<K, V>
    - свойства:
      * K Key;
      * V Value;
    - методы:
      * void Put(K key, V value);
      * void Clear();
      * bool ContainsKey(K key);
      * bool ContainsValue(V value);
      * void Remove(K key);
    - свойства:
      * int Count;
      * bool isEmpty;
      * V this[K key];
      * IEnumerable<K> Keys;
      * IEnumerable<V> Values;
  + MapException – класс, описывающий исключения, которые могут происходить в ходе работы с ассоциативными множествами (также можно написать ряд наследников от MapException);
  + ArrayMap<K, V>: IMap<K, V> – класс ассоциативного множества на основе массива;
  + LinkedMap<K, V>: IMap<K, V> – класс ассоциативного множества на основе связного списка;
  + HashMap<K, V>: IMap<K, V> – класс ассоциативного множества на основе алгоритма хеширования (использовать методы GetHashCode и Equals типа K);
  + UnmutableMap<K, V>: IMap<K, V> – класс неизменяющегося ассоциативного множества, является оберткой над любым существующим ассоциативным множеством (должен кидаться исключениями на вызов любого метода, изменяющего ассоциативное множество);
  + Properties: IMap<String, String> – класс для работы со свойствами;
  + MapUtils – класс различных операций над ассоциативным множеством;
    - методы:
      * static bool Exists<K, V>(IMap<K, V>, CheckDelegate<IMap.IEntry<K, V>>);
      * static IMap<K, V> FindAll<K, V>(IMap<K, V>, CheckDelegate<IMap.IEntry<K, V>>,
      * MapConstructorDelegate<K, V>);
      * static void ForEach(IMap<K, V>, ActionDelegate<IMap.IEntry<K, V>>);
      * static bool CheckForAll<K, V>(IMap<K, V>, CheckDelegate<IMap.IEntry<K, V>>);
    - свойства:
      * static readonly MapConstructorDelegate<K, V> ArrayMapConstructor;
      * static readonly MapConstructorDelegate<K, V> LinkedMapConstructor;
      * static readonly MapConstructorDelegate<K, V> HashMapConstructor;

Также необходимо разработать серию примеров, демонстрирующих основные аспекты работы с данной библиотекой множеств.

1. **Мультимножество**

Разработать библиотеку обобщенных классов для работы с мультимножествами данных. В структуру классов входят как минимум:

* + ISet<T>: IEnumerable<T> – базовый интерфейс для всех мультимножеств;
    - методы:
      * void Add(T value);
      * void Clear();
      * bool Contains(T value);
      * void Remove(T value);
    - свойства:
      * int Count;
      * bool isEmpty;
  + SetException – класс, описывающий исключения, которые могут происходить в ходе работы с мультимножествами (также можно написать ряд наследников от SetException);
  + ArraySet<T>: ISet<T> – класс мультимножества на основе массива;
  + LinkedSet<T>: ISet<T> – класс мультимножества на основе связного списка;
  + HashSet<T>: ISet<T> – класс мультимножества на основе алгоритма хеширования (использовать методы GetHashCode и Equals типа T);
  + UnmutableSet<T>: ISet<T> – класс неизменяющегося мультимножества, является оберткой над любым существующим множеством (должен кидаться исключениями на вызов любого метода, изменяющего множество);
  + SetUtils – класс различных операций над мультимножеством;
    - методы:
      * static bool Exists<T>(ISet<T>, CheckDelegate<T>);
      * static ISet<T> FindAll<T>(ISet<T>, CheckDelegate<T>, SetConstructorDelegate<T>);
      * static ISet<TO> ConvertAll<TI, TO>(ISet<TI>, ConvertDelegate<TI, TO>, SetConstructorDelegate<TO>);
      * static void ForEach(ISet<T>, ActionDelegate<T>);
      * static bool CheckForAll<T>(ISet<T>, CheckDelegate<T>);
    - свойства:
      * static readonly SetConstructorDelegate<T> ArraySetConstructor;
      * static readonly SetConstructorDelegate<T> LinkedSetConstructor;
      * static readonly SetConstructorDelegate<T> HashSetConstructor;

Также необходимо разработать серию примеров, демонстрирующих основные аспекты работы с данной библиотекой мультимножеств.

1. **Очередь**

Разработать библиотеку обобщенных классов для работы с очередью. В структуру классов входят как минимум:

* + IQueue<T>: IEnumerable<T> – базовый интерфейс для всех очередей;
    - методы:
      * void Push(T value);
      * void Clear();
      * T Pop();
      * T Peek();
    - свойства:
      * int Count;
      * bool isEmpty;
  + QueueException – класс, описывающий исключения, которые могут происходить в ходе работы c очередью(также можно написать ряд наследников от QueueException);
  + ArrayQueue<T>: IQueue<T> – класс очереди на основе массива;
  + LinkedQueue<T>: IQueue<T> – класс очереди на основе связного списка;
  + UnmutableQueue<T>: IQueue<T> – класс неизменяющейся очереди, является оберткой над любой существующей очередью (должен кидаться исключениями на вызов любого метода, изменяющего очередь);
  + QueueUtils – класс различных операций над очередью;
    - методы:
      * static bool Exists<T>(IQueue<T>, CheckDelegate<T>);
      * static IQueue<T> FindAll<T>(IQueue<T>, CheckDelegate<T>, QueueConstructorDelegate<T>);
      * static IQueue<TO> ConvertAll<TI, TO>(IQueue<TI>, ConvertDelegate<TI, TO>, QueueConstructorDelegate<TO>);
      * static void ForEach(IQueue<T>, ActionDelegate<T>);
      * static bool CheckForAll<T>(IQueue<T>, CheckDelegate<T>);
    - свойства:
      * static readonly QueueConstructorDelegate<T> ArrayQueueConstructor;
      * static readonly QueueConstructorDelegate<T> LinkedQueueConstructor;

Также необходимо разработать серию примеров, демонстрирующих основные аспекты работы с данной библиотекой очередей.

1. **Кольцевая очередь**

Разработать библиотеку обобщенных классов для работы с кольцевой очередью. В структуру классов входят как минимум:

* + IQueue<T>: IEnumerable<T> – базовый интерфейс для всех кольцевых очередей;
    - методы:
      * void Push(T value);
      * void Clear();
      * T Pop();
      * T Peek();
    - свойства:
      * int Count;
      * bool isEmpty;
  + QueueException – класс, описывающий исключения, которые могут происходить в ходе работы c кольцевой очередью (также можно написать ряд наследников от QueueException);
  + ArrayQueue<T>: IQueue<T> – класс кольцевой очереди на основе массива;
  + LinkedQueue<T>: IQueue<T> – класс кольцевой очереди на основе связного списка;
  + UnmutableQueue<T>: IQueue<T> – класс неизменяющейся кольцевой очереди, является оберткой над любой существующей очередью (должен кидаться исключениями на вызов любого метода, изменяющего очередь);
  + QueueUtils – класс различных операций над кольцевой очередью;
    - методы:
      * static bool Exists<T>(IQueue<T>, CheckDelegate<T>);
      * static IQueue<T> FindAll<T>(IQueue<T>, CheckDelegate<T>, QueueConstructorDelegate<T>);
      * static IQueue<TO> ConvertAll<TI, TO>(IQueue<TI>, ConvertDelegate<TI, TO>, QueueConstructorDelegate<TO>);
      * static void ForEach(IQueue<T>, ActionDelegate<T>);
      * static bool CheckForAll<T>(IQueue<T>, CheckDelegate<T>);
    - свойства:
      * static readonly QueueConstructorDelegate<T> ArrayQueueConstructor;
      * static readonly QueueConstructorDelegate<T> LinkedQueueConstructor;

Также необходимо разработать серию примеров, демонстрирующих основные аспекты работы с данной библиотекой кольцевых очередей.

1. **Двусторонняя очередь(дек)**

Разработать библиотеку обобщенных классов для работы с двусторонней очередью. В структуру классов входят как минимум:

* + IQueue<T>: IEnumerable<T> – базовый интерфейс для всех двусторонних очередей;
    - методы:
      * void PushTop(T value);
      * void PushEnd(T value);
      * void Clear();
      * T PopTop();
      * T PopEnd();
      * T PeekTop();
      * T PeekEnd();
    - свойства:
      * int Count;
      * bool isEmpty;
  + QueueException – класс, описывающий исключения, которые могут происходить в ходе работы c двусторонней очередью (также можно написать ряд наследников от QueueException);
  + ArrayQueue<T>: IQueue<T> – класс двусторонней очереди на основе массива;
  + LinkedQueue<T>: IQueue<T> – класс двусторонней очереди на основе связного списка;
  + UnmutableQueue<T>: IQueue<T> – класс неизменяющейся двусторонней очереди, является оберткой над любой существующей очередью (должен кидаться исключениями на вызов любого метода, изменяющего очередь);
  + QueueUtils – класс различных операций над двусторонней очередью;
    - методы:
      * static bool Exists<T>(IQueue<T>, CheckDelegate<T>);
      * static IQueue<T> FindAll<T>(IQueue<T>, CheckDelegate<T>, QueueConstructorDelegate<T>);
      * static IQueue<TO> ConvertAll<TI, TO>(IQueue<TI>, ConvertDelegate<TI, TO>, QueueConstructorDelegate<TO>);
      * static void ForEach(IQueue<T>, ActionDelegate<T>);
      * static bool CheckForAll<T>(IQueue<T>, CheckDelegate<T>);
    - свойства:
      * static readonly QueueConstructorDelegate<T> ArrayQueueConstructor;
      * static readonly QueueConstructorDelegate<T> LinkedQueueConstructor;

Также необходимо разработать серию примеров, демонстрирующих основные аспекты работы с данной библиотекой двусторонних очередей.

1. **Стек**

Разработать библиотеку обобщенных классов для работы со стеком. В структуру классов входят как минимум:

* + IStack<T>: IEnumerable<T> – базовый интерфейс для всех стеков;
    - методы:
      * void Push(T value);
      * void Clear();
      * T Pop();
      * T Peek();
    - свойства:
      * int Count;
      * bool isEmpty;
  + StackException – класс, описывающий исключения, которые могут происходить в ходе работы cо стеком (также можно написать ряд наследников от StackException);
  + ArrayStack<T>: IStack<T> – класс стека на основе массива;
  + LinkedStack<T>: IStack<T> – класс стека на основе связного списка;
  + UnmutableStack<T>: IStack<T> – класс неизменяющегося стека, является оберткой над любой существующим стеком (должен кидаться исключениями на вызов любого метода, изменяющего стек);
  + StackUtils – класс различных операций над стеком;
    - методы:
      * static bool Exists<T>(IStack<T>, CheckDelegate<T>);
      * static IStack<T> FindAll<T>(IStack<T>, CheckDelegate<T>, StackConstructorDelegate<T>);
      * static IStack<TO> ConvertAll<TI, TO>(IStack<TI>, ConvertDelegate<TI, TO>, StackConstructorDelegate<TO>);
      * static void ForEach(IStack<T>, ActionDelegate<T>);
      * static bool CheckForAll<T>(IStack<T>, CheckDelegate<T>);
    - свойства:
      * static readonly StackConstructorDelegate<T> ArrayStackConstructor;
      * static readonly StackConstructorDelegate<T> LinkedStackConstructor;

Также необходимо разработать серию примеров, демонстрирующих основные аспекты работы с данной библиотекой стеков.

1. **Матрица**

Разработать библиотеку обобщенных классов Matrix<T> для работы с матрицами произвольных чисел. В структуру классов входят как минимум:

* + IMatrix<T> > – базовый интерфейс для всех матриц;
    - методы:
      * IMatrix<T> FromArray(T[,] array);
    - свойства:
      * int Width;
      * int Height;
      * T this[int wIndex, int hIndex];
      * IMathCalculator<T> MathCalculator;
  + MatrixException – класс, описывающий исключения, которые могут происходить в ходе работы c классом Matrix<T>;
  + ArrayMatrix<T>:IMatrix – класс матриц на основе массива T[,] array;
    - операторы
      * static ArrayMatrix<T> operator +(ArrayMatrix<T> m1, ArrayMatrix<T> m2);
      * static ArrayMatrix<T> operator – (ArrayMatrix<T> m1, ArrayMatrix<T> m2);
      * static ArrayMatrix<T> operator \*(ArrayMatrix<T> m1, ArrayMatrix<T> m2);
      * static ArrayMatrix<T> operator /(ArrayMatrix<T> m1, ArrayMatrix<T> m2);
  + JaggedArrayMatrix<T>:IMatrix – класс матриц на основе ступенчатого массива T[][] array;
    - операторы
      * static JaggedArrayMatrix<T> operator + (JaggedArrayMatrix<T> m1, JaggedArrayMatrix<T> m2);
      * static JaggedArrayMatrix<T> operator – (JaggedArrayMatrix<T> m1, JaggedArrayMatrix<T> m2);
      * static JaggedArrayMatrix<T> operator \* (JaggedArrayMatrix<T> m1, JaggedArrayMatrix<T> m2);
      * static JaggedArrayMatrix<T> operator / (JaggedArrayMatrix<T> m1, JaggedArrayMatrix<T> m2);
  + IMathCalculator<T> – интерфейс выполнения математических операций над типом Т
    - методы:
      * T Add(T n1, T n2);
      * T Sub(T n1, T n2);
      * T Mul(T n1, T n2);
      * T Div(T n1, T n2);
  + MatrixUtils – класс различных операций над матрицей;
    - методы:
      * static bool Exists<T>(IMatrix<T>, CheckDelegate<T>);
      * static IMatrix<T> FindAll<T>(IMatrix<T>, CheckDelegate<T>, MatrixConstructorDelegate<T>);
      * static IMatrix<TO> ConvertAll<TI, TO>(IMatrix<TI>, ConvertDelegate<TI, TO>, MatrixConstructorDelegate<TO>);
      * static void ForEach(IMatrix<T>, ActionDelegate<T>);
      * static bool CheckForAll<T>(IMatrix<T>, CheckDelegate<T>);
    - свойства:
      * static readonly MatrixConstructorDelegate<T> ArrayMatrixConstructor;
      * static readonly MatrixConstructorDelegate<T> JaggedMatrixConstructor;

Также необходимо разработать серию примеров, демонстрирующих основные аспекты работы с классом Matrix, как минимум, для типов double и Complex, а также самого Matrix<T> (т.е. для матрицы матриц).

1. **Неориентированный граф**

Разработать обобщенный класс Graph<N, L> – класс для описания неориентированных графов (N – тип вершин; L – тип связей).

* + IGraph <N,L>: IEnumerable<N,L> – базовый интерфейс для всех неориентированных графов;
    - методы:
      * void AddNode(N node);
      * void AddLink(N node1,N node2);
      * void Clear();
      * bool Contains (N node);
      * void Remove(N node);
    - свойства:
      * int Count;
      * bool isEmpty;
      * IEnumerable<N> nodes;
  + GraphException – класс, описывающий исключения, которые могут происходить в ходе работы с неориентированным графом (также можно написать ряд наследников от GraphException);
  + ArrayGraph < N,L >: IGraph< N,L > – класс неориентированного графа на основе массива;
  + LinkedGraph < N,L >: IGraph< N,L > – класс неориентированногографа на основе связного списка;
  + UnmutableGraph< N,L >: IGraph< N,L > – класс неизменяющегося неориентированного графа, является оберткой над любым существующим графом (должен кидаться исключениями на вызов любого метода, изменяющего граф);
  + GraphUtils – класс различных операций над неориентированным графом;
    - методы:
      * static bool Exists< N >(IGraph<N,L>, CheckDelegate<N,L>);
      * static IGraph< N,L > FindAll< N,L >(IGraph< N,L>, CheckDelegate< N,L >,GraphConstructorDelegate< N,L>);
      * static void ForEach(IGraph< N,L >, ActionDelegate< N,L >);
      * static bool CheckForAll< N,L >(IGraph< N,L >, CheckDelegate< N,L >);
    - свойства:
      * static readonly GraphConstructorDelegate< N,L > ArrayGraphConstructor;
      * static readonly GraphConstructorDelegate< N,L > LinkedGraphConstructor;

Класс должен поддерживать базовые алгоритмы на графах (обход графа в глубину, обход графа в ширину, поиск кратчайшего пути и т.п.).

Также необходимо разработать серию примеров, демонстрирующих основные аспекты работы с данной библиотекой неориентированных графов.

1. **Ориентированный граф**

Разработать обобщенный класс Graph<N, L> – класс для описания ориентированных графов (N – тип вершин; L – тип связей).

* + IGraph <N,L>: IEnumerable<N,L> – базовый интерфейс для всех ориентированных графов;
    - методы:
      * void AddNode(N node);
      * void AddLink(N node1,N node2);
      * void Clear();
      * bool Contains (N node);
      * void Remove(N node);
    - свойства:
      * int Count;
      * bool isEmpty;
      * IEnumerable<N> nodes;
  + GraphException – класс, описывающий исключения, которые могут происходить в ходе работы с ориентированным графом (также можно написать ряд наследников от GraphException);
  + ArrayGraph < N,L >: IGraph< N,L > – класс ориентированного графа на основе массива;
  + LinkedGraph < N,L >: IGraph< N,L > – класс ориентированного графа на основе связного списка;
  + UnmutableGraph< N,L >: IGraph< N,L > – класс неизменяющегося ориентированного графа, является оберткой над любым существующим графом (должен кидаться исключениями на вызов любого метода, изменяющего граф);
  + GraphUtils – класс различных операций над ориентированным графом;
    - методы:
      * static bool Exists< N >(IGraph<N,L>, CheckDelegate<N,L>);
      * static IGraph< N,L > FindAll< N,L >(IGraph< N,L>, CheckDelegate< N,L >,GraphConstructorDelegate< N,L>);
      * static void ForEach(IGraph< N,L >, ActionDelegate< N,L >);
      * static bool CheckForAll< N,L >(IGraph< N,L >, CheckDelegate<N,L >);
    - свойства:
      * static readonly GraphConstructorDelegate< N,L > ArrayGraphConstructor;
      * static readonly GraphConstructorDelegate< N,L > LinkedGraphConstructor;

Класс должен поддерживать базовые алгоритмы на графах (обход графа в глубину, обход графа в ширину, поиск кратчайшего пути и т.п.)

Также необходимо разработать серию примеров, демонстрирующих основные аспекты работы с данной библиотекой ориентированных графов.

1. **Направленный ациклический граф**

Разработать обобщенный класс Graph<N, L> – класс для описания направленных ациклических графов (N – тип вершин; L – тип связей).

* + IGraph <N,L>: IEnumerable<N,L> – базовый интерфейс для всех направленных ациклических графов;
    - методы:
      * void AddNode(N node);
      * void AddLink(N node1,N node2);
      * void Clear();
      * bool Contains (N node);
      * void Remove(N node);
    - свойства:
      * int Count;
      * bool isEmpty;
      * IEnumerable<N> nodes;
  + GraphException – класс, описывающий исключения, которые могут происходить в ходе работы с направленным ациклическим графом (также можно написать ряд наследников от GraphException);
  + ArrayGraph < N,L >: IGraph< N,L > – класс направленного ациклического графа на основе массива;
  + LinkedGraph < N,L >: IGraph< N,L > – класс направленного ациклического графа на основе связного списка;
  + UnmutableGraph< N,L >: IGraph< N,L > – класс неизменяющегося направленного ациклического графа, является оберткой над любым существующим графом (должен кидаться исключениями на вызов любого метода, изменяющего граф);
  + GraphUtils – класс различных операций над направленным ациклическим графом;
    - методы:
      * static bool Exists< N >(IGraph<N,L>, CheckDelegate<N,L>);
      * static IGraph< N,L > FindAll< N,L >(IGraph< N,L>, CheckDelegate< N,L >,
      * GraphConstructorDelegate< N,L>);
      * static void ForEach(IGraph< N,L >, ActionDelegate< N,L >);
      * static bool CheckForAll< N,L >(IGraph< N,L >, CheckDelegate< N,L >);
    - свойства:
      * static readonly GraphConstructorDelegate< N,L > ArrayGraphConstructor;
      * static readonly GraphConstructorDelegate< N,L > LinkedGraphConstructor;

Класс должен поддерживать базовые алгоритмы на графах (обход графа в глубину, обход графа в ширину, поиск кратчайшего пути и т.п.)

Также необходимо разработать серию примеров, демонстрирующих основные аспекты работы с данной библиотекой направленных ациклических графов.

1. **Двоичное дерево поиска**

Разработать обобщенный класс Tree<T> – класс для описания двоичного дерева поиска.

ITree <T>: IEnumerable<T> – базовый интерфейс для всех двоичных деревьев поиска;

* + - методы:
      * void Add (T node);
      * void Clear();
      * bool Contains (T node);
      * void Remove(T node);
    - свойства:
      * int Count;
      * bool isEmpty;
      * IEnumerable<T> nodes;
  + Tree Exception – класс, описывающий исключения, которые могут происходить в ходе работы с двоичным деревом поиска (также можно написать ряд наследников от TreeException);
  + ArrayTree < T>: ITree < T > – класс двоичного дерева поиска на основе массива;
  + LinkedTree < T >: ITree < T > – класс двоичного дерева поиска на основе связного списка;
  + UnmutableTree < T >: ITree < T > – класс неизменяющегося двоичного дерева поиска, является оберткой над любым существующим деревом (должен кидаться исключениями на вызов любого метода, изменяющего дерево);
  + Tree Utils – класс различных операций над двоичным деревом поиска;
    - методы:
      * static bool Exists< T >(ITree < T >, CheckDelegate< T >);
      * static ITree < T > FindAll< T >(ITree < T >, CheckDelegate<T>,TreeConstructorDelegate< T >);
      * static void ForEach(ITree < T >, ActionDelegate< T >);
      * static bool CheckForAll< T >(ITree < T >, CheckDelegate<T >);
    - свойства:
      * static readonly TreeConstructorDelegate< T > ArrayTreeConstructor;
      * static readonly TreeConstructorDelegate< T > LinkedTreeConstructor;

Также необходимо разработать серию примеров, демонстрирующих основные аспекты работы с данной библиотекой двоичных деревьев поиска.

1. **Двоичное дерево**

Разработать обобщенный класс Tree<T> – класс для описания двоичного дерева.

ITree <T>: IEnumerable<T> – базовый интерфейс для всех двоичных деревьев;

* + - методы:
      * void Add (T node);
      * void Clear();
      * bool Contains (T node);
      * void Remove(T node);
    - свойства:
      * int Count;
      * bool isEmpty;
      * IEnumerable<T> nodes;
  + Tree Exception – класс, описывающий исключения, которые могут происходить в ходе работы с двоичным деревом (также можно написать ряд наследников от TreeException);
  + ArrayTree < T>: ITree < T > – класс двоичного дерева на основе массива;
  + LinkedTree < T >: ITree < T > – класс двоичного дерева на основе связного списка;
  + UnmutableTree < T >: ITree < T > – класс неизменяющегося двоичного дерева, является оберткой над любым существующим деревом (должен кидаться исключениями на вызов любого метода, изменяющего дерево);
  + Tree Utils – класс различных операций над двоичным деревом;
    - методы:
      * static bool Exists< T >(ITree < T >, CheckDelegate< T >);
      * static ITree < T > FindAll< T >(ITree < T >, CheckDelegate<T>,TreeConstructorDelegate< T >);
      * static void ForEach(ITree < T >, ActionDelegate< T >);
      * static bool CheckForAll< T >(ITree < T >, CheckDelegate<T >);
    - свойства:
      * static readonly TreeConstructorDelegate< T > ArrayTreeConstructor;
      * static readonly TreeConstructorDelegate< T > LinkedTreeConstructor;

Также необходимо разработать серию примеров, демонстрирующих основные аспекты работы с данной библиотекой двоичных деревьев.

1. **Сбалансированное дерево**

Разработать обобщенный класс Tree<T> – класс для описания сбалансированного дерева.

ITree <T>: IEnumerable<T> – базовый интерфейс для всех сбалансированных деревьев;

* + - методы:
      * void Add (T node);
      * void Clear();
      * bool Contains (T node);
      * void Remove(T node);
    - свойства:
      * int Count;
      * bool isEmpty;
      * IEnumerable<T> nodes;
  + Tree Exception – класс, описывающий исключения, которые могут происходить в ходе работы со сбалансированным деревом (также можно написать ряд наследников от TreeException);
  + ArrayTree < T>: ITree < T > – класс сбалансированного дерева на основе массива;
  + LinkedTree < T >: ITree < T > – класс сбалансированного дерева на основе связного списка;
  + UnmutableTree < T >: ITree < T > – класс неизменяющегося сбалансированного дерева, является оберткой над любым существующим деревом (должен кидаться исключениями на вызов любого метода, изменяющего дерево);
  + Tree Utils – класс различных операций над сбалансированным деревом;
    - методы:
      * static bool Exists< T >(ITree < T >, CheckDelegate< T >);
      * static ITree < T > FindAll< T >(ITree < T >, CheckDelegate<T>,TreeConstructorDelegate< T >);
      * static void ForEach(ITree < T >, ActionDelegate< T >);
      * static bool CheckForAll< T >(ITree < T >, CheckDelegate<>);
    - свойства:
      * static readonly TreeConstructorDelegate< T > ArrayTreeConstructor;
      * static readonly TreeConstructorDelegate< T > LinkedTreeConstructor;

Также необходимо разработать серию примеров, демонстрирующих основные аспекты работы с данной библиотекой сбалансированных деревьев.

1. **B+ дерево**

Разработать обобщенный класс Tree<T> – класс для описания B+ дерева.

ITree <T>: IEnumerable<T> – базовый интерфейс для всех B+деревьев;

* + - методы:
      * void Add (T node);
      * void Clear();
      * bool Contains (T node);
      * void Remove(T node);
    - свойства:
      * int Count;
      * bool isEmpty;
      * IEnumerable<T> nodes;
  + Tree Exception – класс, описывающий исключения, которые могут происходить в ходе работы с B+деревом (также можно написать ряд наследников от TreeException);
  + ArrayTree < T>: ITree < T > – класс B+дерева на основе массива;
  + LinkedTree < T >: ITree < T > – класс B+дерева на основе связного списка;
  + UnmutableTree < T >: ITree < T > – класс неизменяющегося B+дерева, является оберткой над любым существующим деревом (должен кидаться исключениями на вызов любого метода, изменяющего дерево);
  + Tree Utils – класс различных операций над B+деревом;
    - методы:
      * static bool Exists< T >(ITree < T >, CheckDelegate< T >);
      * static ITree < T > FindAll< T >(ITree < T >, CheckDelegate<T>,TreeConstructorDelegate<T >);
      * static void ForEach(ITree < T >, ActionDelegate< T >);
      * static bool CheckForAll< T >(ITree < T >, CheckDelegate<T >);
    - свойства:
      * static readonly TreeConstructorDelegate< T > ArrayTreeConstructor;
      * static readonly TreeConstructorDelegate< T > LinkedTreeConstructor;

Также необходимо разработать серию примеров, демонстрирующих основные аспекты работы с данной библиотекой B+деревьев.

1. **Max-куча**

Разработать обобщенный класс Heap<T> – класс для описания max- кучи.

IHeap <T>: IEnumerable<T> – базовый интерфейс для всех max- куч;

* + - методы:
      * void Add (T node);
      * void Clear();
      * bool Contains (T node);
      * void Remove(T node);
    - свойства:
      * int Count;
      * bool isEmpty;
      * IEnumerable<T> nodes;
  + Heap Exception – класс, описывающий исключения, которые могут происходить в ходе работы с max- кучей (также можно написать ряд наследников от HeapException);
  + ArrayHeap < T>: IHeap < T > – класс max- кучи на основе массива;
  + LinkedHeap < T >: IHeap < T > – класс max- кучи на основе связного списка;
  + UnmutableHeap < T >: IHeap< T > – класс неизменяющейся max- кучи, является оберткой над любым существующей кучей(должен кидаться исключениями на вызов любого метода, изменяющего кучу);
  + HeapUtils – класс различных операций над max- кучей;
    - методы:
      * static bool Exists< T >(IHeap < T >, CheckDelegate< T >);
      * static IHeap < T > FindAll< T >(IHeap < T >, CheckDelegate<T>, HeapConstructorDelegate< T >);
      * static void ForEach(IHeap < T >, ActionDelegate< T >);
      * static bool CheckForAll< T >(IHeap < T >, CheckDelegate<T >);
    - свойства:
      * static readonly HeapConstructorDelegate< T > ArrayHeapConstructor;
      * static readonly HeapConstructorDelegate< T > LinkedHeapConstructor;

Также необходимо разработать серию примеров, демонстрирующих основные аспекты работы с данной библиотекой max- куч.

1. **Min-куча**

Разработать обобщенный класс Heap<T> – класс для описания min-кучи.

IHeap <T>: IEnumerable<T> – базовый интерфейс для всех min-куч;

* + - методы:
      * void Add (T node);
      * void Clear();
      * bool Contains (T node);
      * void Remove(T node);
    - свойства:
      * int Count;
      * bool isEmpty;
      * IEnumerable<T> nodes;
  + Heap Exception – класс, описывающий исключения, которые могут происходить в ходе работы с min-кучей (также можно написать ряд наследников от HeapException);
  + ArrayHeap < T>: IHeap < T > – класс min-кучи на основе массива;
  + LinkedHeap < T >: IHeap < T > – класс min-кучи на основе связного списка;
  + UnmutableHeap < T >: IHeap< T > – класс неизменяющейся min-кучи, является оберткой над любым существующей кучей(должен кидаться исключениями на вызов любого метода, изменяющего кучу);
  + HeapUtils – класс различных операций над min-кучей;
    - методы:
      * static bool Exists< T >(IHeap < T >, CheckDelegate< T >);
      * static IHeap < T > FindAll< T >(IHeap < T >, CheckDelegate<T>, HeapConstructorDelegate< T >);
      * static void ForEach(IHeap < T >, ActionDelegate< T >);
      * static bool CheckForAll< T >(IHeap < T >, CheckDelegate<T>);
    - свойства:
      * static readonly HeapConstructorDelegate< T > ArrayHeapConstructor;
      * static readonly HeapConstructorDelegate< T > LinkedHeapConstructor;

Также необходимо разработать серию примеров, демонстрирующих основные аспекты работы с данной библиотекой min-куч.

1. **Двоичная куча**

Разработать обобщенный класс Heap<T> – класс для описания двоичной кучи.

IHeap <T>: IEnumerable<T> – базовый интерфейс для всех двоичных куч;

* + - методы:
      * void Add (T node);
      * void Clear();
      * bool Contains (T node);
      * void Remove(T node);
    - свойства:
      * int Count;
      * bool isEmpty;
      * IEnumerable<T> nodes;
  + Heap Exception – класс, описывающий исключения, которые могут происходить в ходе работы с двоичной кучей (также можно написать ряд наследников от HeapException);
  + ArrayHeap < T>: IHeap < T > – класс двоичной кучи на основе массива;
  + LinkedHeap < T >: IHeap < T > – класс двоичной кучи на основе связного списка;
  + UnmutableHeap < T >: IHeap< T > – класс неизменяющейся двоичной кучи, является оберткой над любым существующей кучей(должен кидаться исключениями на вызов любого метода, изменяющего кучу);
  + HeapUtils – класс различных операций над двоичной кучей;
    - методы:
      * static bool Exists< T >(IHeap < T >, CheckDelegate< T >);
      * static IHeap < T > FindAll< T >(IHeap < T >, CheckDelegate<T>,HeapConstructorDelegate< T >);
      * static void ForEach(IHeap < T >, ActionDelegate< T >);
      * static bool CheckForAll< T >(IHeap < T >, CheckDelegate<T>);
    - свойства:
      * static readonly HeapConstructorDelegate< T > ArrayHeapConstructor;
      * static readonly HeapConstructorDelegate< T > LinkedHeapConstructor;

Также необходимо разработать серию примеров, демонстрирующих основные аспекты работы с данной библиотекой двоичных куч.

1. **Полином**

Разработать библиотеку обобщенных классов Polynom<T> для работы с полиномами произвольных чисел. В структуру классов входят как минимум:

* + IPolynom <T> > – базовый интерфейс для всех полиномов;
    - методы:
      * IPolynom<T> FromArray(T[] array);
      * T Value(T x);
    - свойства:
      * int Power;
      * T this[int Index];
      * IMathCalculator<T> MathCalculator;
  + PolynomException – класс, описывающий исключения, которые могут происходить в ходе работы c классом Polynom<T>;
  + ArrayPolynom<T>:IPolynom – класс полиномов на основе массива;
    - операторы
      * static ArrayPolynom<T> operator +(ArrayPolynom<T> m1, ArrayPolynom<T> m2);
      * static ArrayPolynom<T> operator – (ArrayPolynom<T> m1, ArrayPolynom<T> m2);
      * static ArrayPolynom<T> operator \*(ArrayPolynom<T> m1, ArrayPolynom<T> m2);
  + LinkedPolynom<T>:IPolynom – класс полиномов на основе связного списка;
    - операторы
      * static LinkedPolynom <T> operator +( LinkedPolynom <T> m1, LinkedPolynom <T> m2);
      * static LinkedPolynom <T> operator – ( LinkedPolynom <T> m1, LinkedPolynom <T> m2);
      * static LinkedPolynom <T> operator \*( LinkedPolynom <T> m1, LinkedPolynom <T> m2);
  + IMathCalculator<T> – интерфейс выполнения математических операций над типом Т
    - методы:
      * T Add(T n1, T n2);
      * T Sub(T n1, T n2);
      * T Mul(T n1, T n2);
      * T Div(T n1, T n2);
  + PolynomUtils – класс различных операций над полиномом;
    - методы:
      * static bool Exists<T>(IPolynom<T>, CheckDelegate<T>);
      * static IPolynom<T> FindAll<T>(IPolynom<T>, CheckDelegate<T>, PolynomConstructorDelegate<T>);
      * static IPolynom<TO> ConvertAll<TI, TO>(IPolynom<TI>, ConvertDelegate<TI, TO>, PolynomConstructorDelegate<TO>);
      * static void ForEach(IPolynom<T>, ActionDelegate<T>);
      * static bool CheckForAll<T>(IPolynom<T>, CheckDelegate<T>);
    - свойства:
      * static readonly PolynomConstructorDelegate<T> ArrayPolynomConstructor;
      * static readonly PolynomConstructorDelegate<T> LinkedPolynomConstructor;

Также необходимо разработать серию примеров, демонстрирующих основные аспекты работы с классом, как минимум, для типов double и Complex

# Задание 2

Цель — создание модели, описывающей процесс, построение иерархии классов, отражающей данную предметную область. По действиям пользователя следует создавать экземпляры классов иерархии. Использовать исключения. Разработать класс менеджера, позволяющего создавать, уничтожать и модифицировать объекты (в некоторых задачах —динамика от таймера). Для каждой задачи необходимо реализовать объектную модель. В реализации задач должны присутствовать наследование, агрегация и др. взаимодействия классов. Интерфейс к задаче необходимо выполнить на WinForms с использованием компонентов. Для отображения использовать 2D-графику или графические элементы управления. Формы и компоненты используются только для отображения и ввода данных, но не для хранения объектов. Использовать исключения (exceptions).

Требования к реализации.

* Приложение реализовать как MVC. Модель — иерархия классов. View — демо в автоматическом режиме.
* Действующие объекты должны быть реализованы как потоки с необходимой синхронизацией.
* Для оповещения об изменениях использовать шаблон «Наблюдатель».
* Для создания действующих объектов использовать шаблон «Фабрика».
* Использование других шаблонов приветствуется.

1. Риэлтерское агентство. Приложение должно вести учет покупателей (физические и юридические лица), продавцов (физические и юридические лица), продаваемых объектов (квартиры, дома, нежилые помещения, офисы, склады). Смоделировать поток клиентов и покупателей с различными требованиями и заключение сделок между ними.
2. Районная поликлиника. Основные сущности: сотрудники (врачи, медсестры, заведующая, главная медсестра, мед. статисты), пациенты (взрослые, дети), болезни, учет больничных, прививки. Смоделировать процесс обслуживания пациентов.
3. Строительная фирма. Основные сущности: строящиеся объекты (дома, гаражи, квартиры), кадры (каменщики, плотники, маляры, водители, крановщики), стройматериалы (цемент, кирпич, плиты и т.д.). Смоделировать процесс строительства объектов.
4. Учет валютных операций. Основные сущности: типы валют, типы операций, клиенты. Смоделировать процесс обслуживания клиентов.
5. Домашняя видеотека. Основные сущности: диски (CD, DVD, BlueRay), фильмы, жанры, исполнители, друзья (кому и когда отдан диск на просмотр). Смоделировать обмен дисками N друзей.
6. Ведение заказов в пиццерии. Основные сущности: продукты, рецепты, готовая продукция, клиенты, адреса. Смоделировать процесс изготовления и заказа пиццы, поддерживать необходимый запас продуктов.
7. Биржа труда. Основные сущности: предприятия, их типы, вакансии, соискатели, их опыт работы, информация о полученном образовании. Смоделировать процесс поступления соискателей и вакансий на биржу труда и подбор персонала.
8. Зоомагазин. Основные сущности: товары: живые (рыбки, хомяки, мыши, попугаи, черепахи) и неживые (аквариумы, корма, средства по уходу и т.д.), покупатели. Смоделировать процесс поступления товаров на склад, убыли живого товара в процессе продажи (пример – 100 рыбок поступило, 70 продано, 30 умерло), процесс продажи (обратить внимание на продажу сопутствующих товаров).
9. Спортивный комплекс. Основные сущности: материальная база (бассейны, беговые дорожки, спортзалы), персонал (тренеры, рабочие, уборщицы, врачи), группы (спортивные и оздоровительные), спортсмены и клиенты. Смоделировать процесс работы спортивного комплекса.
10. Грузоперевозки. Основные сущности: автомашины, типы грузов, города, между которыми осуществляются перевозки, сеть дорог, грузы, их типы, клиенты. Смоделировать процесс перевозки грузов.
11. Фирма по созданию ландшафтного дизайна. Основные сущности: типы и виды растений, ландшафтных конструкций, виды работ, предлагаемые услуги (как группа из растений, конструкций и работ); заказы; объекты, над которыми идет работа. Смоделировать процесс озеленения города.
12. Прием платы за электричество. Основные сущности: клиенты (физические и юридические лица), тарифы, платежи, задолженности, пени. Смоделировать процесс потребления, отразить влияние времени года на количество потребляемой электроэнергии.
13. Автосервис. Основные сущности: автомобили, виды поломок, предоставляемые услуги и цены, клиенты. Смоделировать процесс обслуживания автомобилей города в нескольких автосервисах, выполняющих различные типы работ.
14. Учет домашних финансов. Основные сущности: члены семьи, их виды деятельности, поступления, траты, типы и подтипы расходов, связь расходов с членами семьи, накопления, долги, кредиты. Смоделировать распределение средств семьи.
15. Библиотека ВУЗа. Основные сущности: издание (учесть тип), тематика, каталог изданий, читатель, сотрудник, количество экземпляров всего, учет экземпляров выданных на руки. Смоделировать процесс взятия книг, возврата, потерь, пополнения библиотеки. Обратить внимание на сезонные пики активностей.
16. Муравейник. Типы объектов: матка, солдаты, полицейские, рабочие, личинки, вредители. Муравейник имеет склад пищи фиксированного размера. Матка создает личинки, из личинок выводятся солдаты, полицейские или рабочие муравьи и ест очень много, солдаты уменьшают число вредителей и много едят, полицейские помогают увеличить число добываемой пищи и едят меньше солдат, рабочие добывают пищу и едят меньше полицейских, личинки только едят. Показать работу муравейника для различных значений числа муравьев, вредителей, пищи.
17. Банковская система. Основные сущности: физическое лицо, юридическое лило, счет, банк, система банков. Смоделировать процесс создания банка, создание клиента, возможность взять или положить деньги в банк, возможность взять и погасить кредит.
18. Музыкальная школа. Основные сущности: ученики, преподаватели, направления обучения, изучаемые предметы, классы. Смоделировать процессы создания ученика, создания преподавателя, проведение урока индивидуального или группового и т.д.
19. Маршрутное такси. Основные сущности: пассажиры, автобусы, остановки, маршруты. Смоделировать процессы создания машрута, автобуса, пассажира, движения автобуса между остановками, вход и выход пассажиров на остановках.
20. Производство. Основные сущности: производимый продукт, его компоненты, сырье для производства, цеха для производства, рабочие, мастера, начальники. Смоделировать процессы создания производства, цехов, рабочих, мастеров, начальников, процесс изготовления продукции от сырья до готового продукта.
21. Автобаза. Основные сущности: автомобили разной грузоподъемности и разного назначения, водители, диспетчер, начальник, служба ремонта, заказы. Смоделировать процесс создания автомобиля, водителя и т.д., процесс подбора подходящего автомобиля, процесс выполнения рейса и т.д.
22. Домоуправление. Основные сущности: сотрудники – сантехники, электрики, дворники, диспетчер, заявки на выполнение работ Смоделировать процесс создания сотрудника, создания заявки, процесс выполнения заявки и т.д.
23. Мост: базовый класс – мост. Наследники – разводной мост, понтонный и обычный. Менеджер – река и два берега. Ширина реки произвольная. Плывет корабль. На его пути встречаются мосты, которые его пропускают.

# Список литературы

1. Троелсен Э. C# и платформа .NET 3.0 / Э.Троелсен. – СПб. : Питер, 2008 **.**— 1456 с.
2. Шилд Г. Полный справочник по C# / Г. Шилд . – М. : Издательский дом «Вильямс», 2004 .—752 с.
3. Рихтер Дж. CLR via C#. Программирование на платформе Microsoft .NET Framework 2.0 на языке C# / Дж. Рихтер . – М.: Издательство «Русская редакция»; СПб.: Питер, 2007.
4. Павловская Т. А. C#. Программирование на языке высокого уровня / Т.А. Павловская .— СПб : Питер, 2014 .— 432 с.
5. [Тюкачев Н.](https://lib.vsu.ru/zgate?ACTION=follow&SESSION_ID=3811&TERM=Тюкачев,%20Николай%20Аркадьевич%5b1,1004,4,101%5d&LANG=rus) А. C#. Программирование : учебное пособие : [в 3 ч.] / Н.А. Тюкачев, В.Г. Хлебостроев ; Воронеж. гос. ун-т .— Воронеж : Издательско-полиграфический центр Воронежского государственного университета, 2013

Авторы:

Вощинская Гильда Эдгаровна,

Лещенко Елена Михайловна.