**Лабораторная работа №3**

**Проектирование и реализация классов.**

**Класс** представляет собой описание объектов, схожих по характеристикам (свойствам) и поведению (методам). В программировании класс является шаблоном (типом) для создания объектов.

Данный класс определяется (т.е. отличается от других классов) набором свойств, которые определяют состояние объектов, создаваемых на основе данного класса, а также интерфейсом – набором методов (процедур и функций), которые позволяют пользователю выполнять определенные действия над объектами, т.е. определяющих поведение объектов.

Создание объектно-ориентированного приложения состоит из двух этапов: **создание классов** и **использование классов**.

1. **Создание класса**

// объявление класса Фрукт

class Fruit

{

}

Для описания свойства внутри класса создается (инкапсулируется) закрытая (Private) переменная определенного типа.

// объявление класса Фрукт

class Fruit {

// закрытая переменная-свойство «цвет»

private String color; }

Принцип инкапсуляции состоит в том, что данные скрываются от пользователя внутри класса. Доступ к свойству обеспечивается (по усмотрению разработчика класса) с помощью методов чтения значения свойства (get) и записи значения свойства (set), например:

// объявление класса Фрукт

class Fruit {

// закрытая переменная-свойство «цвет»

private String color;

// открытый метод-функция, обеспечивающий чтение свойства «цвет»

public String getColor() {

return color; }

// открытый метод-процедура, обеспечивающий запись свойства «цвет»

public void setColor(String newColor) {

color = newColor;

} }

Существуют специальные языки (нотации) графического отображения объектно-ориентированных структур, которые позволяют в наглядном виде моделировать разные аспекты проектируемой системы. Одним из наиболее распространенных является Унифицированный Язык Моделирования (UML – Unified Modeling Language). В UML класс отображается в виде прямоугольника, разбитого на три части. В верхней части прямоугольника отображается название класса, в средней части прямоугольника отображается набор полей (свойств), в нижней части прямоугольника отображается набор методов.

Для свойств могут быть указаны типы значений, для методов могут быть указаны наборы параметров и типы возвращаемых значений. Реализация (содержимое) методов не отображается.

Класс Фрукт на языке UML может быть представлен следующим образом:

Fruit

color: String

void setColor(String newColor)

String getColor()

Рис 1. Описание класса Fruit на языке UML

1. **Использование классов**

Применение классов состоит в создании объектов и обращении (вызове) методов созданных объектов.

public class TestFruit {

public static void main(String args[]) {

// создание объектов класса Fruit

Fruit apple = new Fruit();

Fruit lemon = new Fruit();

// запись значений свойств

apple.setColor(“green”);

lemon.setColor(“yellow”);

// чтение значений свойств

Console.WriteLine(apple.getColor());

Console.WriteLine (lemon.getColor());

} }

// объявление класса Фрукт

class Fruit {

// закрытая переменная-свойство «цвет»

private String color;

// открытый метод, обеспечивающий чтение свойства «цвет»

public String getColor() {

return color;

}

// открытый метод, обеспечивающий запись свойства «цвет»

public void setColor(String newColor) {

color = newColor;

} }

**Связи между классами (композиция)**

Объекты различных классов могут быть взаимосвязаны друг с другом, например, Сотрудник работает В Организации, Аэропорт предлагает Рейсы, Фрукт растет на Дереве и т.д.

В объектно-ориентированном проектировании подобные взаимосвязи (ассоциации) между классами описываются с помощью отношения композиции. Участниками отношения являются два взаимосвязанных класса и ассоциация. Ассоциация может быть однонаправленной или двунаправленной. Для каждого окончания ассоциации определяется «арность», т.е. ограничение на количество объектов данного класса, которые могут взаимодействовать с указанным количеством объектов другого класса.

**1. Однонаправленная ассоциация** означает одностороннюю связь между двумя классами, когда один класс содержит одну или много ссылок на объекты второго класса. В результате, получаем конструкцию, которая означает, что объект первого класса «знает» о связях с объектами другого класса.

В UML однонаправленная ассоциация отображается в виде сплошной стрелки от исходного класса к связываемому, с которым он взаимодействует. Множественность указывается возле окончаний стрелки в виде записи <мин. значение> [… <макс. значение>]. Если в качестве макс. значения указан символ \*, то это означает, что объектов может быть любое количество.

***Пример.***

Отобразим структуру взаимосвязи Фрукт растет на Дереве.

Tree

name: String

setName()

getName()

1

Fruit

color: String

setColor()

getColor()

0…\*

Рис 2. Однонаправленная связь между классами Fruit и Tree на языке UML

Данная структура говорит о том, что каждый фрукт «знает», на каком дереве он растет, при этом каждый объект-фрукт может расти только на одном дереве, а дерево может содержать от 0 до бесконечного множества (\*) фруктов.

**2. Двунаправленная ассоциация,** в отличие от однонаправленной, означает двустороннюю связь между классами, когда первый класс содержит ссылки на объекты второго класса, а второй класс содержит ссылки на объекты первого. В результате каждый из классов – участников ассоциации, «знает» о связях с другим классом.

Двунаправленная ассоциация в UML отображается в виде сплошной линии. Отобразим взаимосвязь Фрукт растет на Дереве в виде двунаправленной ассоциации:

0…\*

1

Tree

name: String

setName()

getName()

Fruit

color: String

setColor()

getColor()

Рис 3. Двунаправленная связь между классами Fruit и Tree на языке UML

В отличие от предыдущего примера, здесь дерево также «знает» о фруктах, которые оно содержит.

**Реализация ассоциативных связей**

В объектно-ориентированном программировании ассоциативные связи реализуются с помощью ссылок на объекты соответствующих классов, т.е. фактически ассоциации реализуются обычным путем в виде свойств класса. Ранее, мы уже использовали ассоциативные связи с объектами типа String, описывая свойства строкового типа. Разница состоит лишь в том, что ранее мы использовали стандартный класс String, а сейчас используем собственные классы.

Например, структура, отображенная на рис. 2 реализуется следующим образом.

// объявление класса Фрукт

class Fruit {

// Описание свойства color

...

// Описание связи Фрукта с Деревом

// Объявление ссылки на объект класса Дерево

private Tree tree;

// Получить значение свойства Дерево

// (узнать на каком дереве растет данный фрукт)

public Tree getTree() {

return tree;

}

// Установить значение свойства Дерево

(указать на каком дереве растет данный фрукт)

public void setTree(Tree tree) {

this.tree = tree;

}

// объявление класса Дерево

class Tree {

// Описание свойства name

...}

Двунаправленная ассоциация реализуется путем введения соответствующего свойства-связи во второй класс.

**Реализация множественных свойств, коллекции.**

Множественное свойство, например свойство «фрукты» класса Дерево, которое появляется при создании ассоциации Дерево 🡪 Фрукт на рис.3, должно представлять собой **набор ссылок** определенного типа. В этом случае внутренняя переменную можно описать в виде массива ссылок, например:

// объявление класса Дерево

class Tree {

// Описание свойства Фрукты

private Fruit fruits[] = new Fruit[100];

}

Доступ к данному свойству можно обеспечить посредством операций set и get, однако это не всегда удобно для пользователей класса, так как заставляет каждый раз оперировать целым набором объектов. В то же время могут понадобиться операции вида «**добавить фрукт**», «**удалить фрукт**» и т.п., которые достаточно сложно реализуются при использовании массивов. Также зачастую неизвестна исходная размерность массива.

Гораздо удобнее в подобных случаях использовать специальные типы данных, называемые «коллекциями». **Коллекция** – это специальный объект, представляющий собой контейнер для хранения множества ссылок на другие объекты.

Основные преимущества коллекций:

* имеют динамическую размерность (т.е. размерность не задается при объявлении и может меняться в ходе работы программы). Таким образом, размер коллекции ограничен только объемом оперативной памяти.
* могут хранить объекты любых типов (все объекты, хранимые в коллекции, приводятся к базовому типу Object)
* существуют стандартные методы добавления и удаления элементов коллекции
* элементы коллекции проиндексированы (аналогично массивам), переиндексация, при добавлении/удалении происходит автоматически.

**Использование простой коллекции**

В примерах этого раздела используется универсальный класс List<T>, который позволяет работать со строго типизированными списками объектов.

В приведенном ниже примере создается список строк, а затем выполняется перебор строк с помощью оператора foreach.

// Create a list of strings.

var salmons = new List<string>();

salmons.Add("chinook");

salmons.Add("coho");

salmons.Add("pink");

salmons.Add("sockeye");

// Iterate through the list.

foreach (var salmon in salmons)

{

Console.Write(salmon + " ");

}

// Output: chinook coho pink sockeye

**Виды коллекций**

Многие типовые коллекции предоставляются платформой .NET Framework. Каждый тип коллекции предназначен для определенной цели.

В этом разделе описываются следующие часто используемые классы коллекций:

Классы System.Collections.Generic

Классы System.Collections.Concurrent

Классы System.Collections

**Классы System.Collections.Generic**

Универсальную коллекцию можно создать, используя один из классов в пространстве имен System.Collections.Generic. Универсальная коллекция применяется в том случае, если все элементы в коллекции имеют одинаковый тип данных.

Универсальная коллекция обеспечивает строгую типизацию, позволяя добавлять данные только необходимого типа.

В таблице ниже перечислены некоторые из часто используемых классов пространства имен System.Collections.Generic.

| **Класс** | **ОПИСАНИЕ** |
| --- | --- |
| [Dictionary<TKey,TValue>](https://docs.microsoft.com/ru-ru/dotnet/api/system.collections.generic.dictionary-2) | Предоставляет коллекцию пар «ключ-значение», которые упорядочены по ключу. |
| [List<T>](https://docs.microsoft.com/ru-ru/dotnet/api/system.collections.generic.list-1) | Представляет список объектов, доступных по индексу. Предоставляет методы для поиска по списку, его сортировки и изменения. |
| [Queue<T>](https://docs.microsoft.com/ru-ru/dotnet/api/system.collections.generic.queue-1) | Представляет коллекцию объектов, которая обслуживается в порядке поступления (FIFO). |
| [SortedList<TKey,TValue>](https://docs.microsoft.com/ru-ru/dotnet/api/system.collections.generic.sortedlist-2) | Представляет коллекцию пар "ключ-значение", упорядоченных по ключу на основе реализации [IComparer<T>](https://docs.microsoft.com/ru-ru/dotnet/api/system.collections.generic.icomparer-1). |
| [Stack<T>](https://docs.microsoft.com/ru-ru/dotnet/api/system.collections.generic.stack-1) | Представляет коллекцию объектов, которая обслуживается в обратном порядке (LIFO). |

**Классы System.Collections.Concurrent**

В .NET Framework 4 или более поздней версии коллекции пространства имен System.Collections.Concurrent предоставляют эффективные потокобезопасные операции для доступа к элементам коллекции из нескольких потоков.

Классы пространства имен System.Collections.Concurrent следует использовать вместо соответствующих типов пространств имен System.Collections.Generic и System.Collections, если несколько потоков параллельно обращаются к такой коллекции.

Некоторые из классов, входящих в пространство имен System.Collections.Concurrent, — это BlockingCollection<T>, ConcurrentDictionary<TKey,TValue>, ConcurrentQueue<T> и ConcurrentStack<T>.

**Классы System.Collections**

Классы в пространстве имен System.Collections хранят элементы не в виде конкретно типизированных объектов, а как объекты типа Object.

Везде, где это возможно, следует использовать универсальные коллекции пространства имен System.Collections.Generic или пространства имен System.Collections.Concurrent вместо устаревших типов пространства имен System.Collections.

В следующей таблице перечислены некоторые из часто используемых классов пространства имен System.Collections:

| **Класс** | **ОПИСАНИЕ** |
| --- | --- |
| [ArrayList](https://docs.microsoft.com/ru-ru/dotnet/api/system.collections.arraylist) | Представляет массив объектов, размер которого динамически увеличивается по мере необходимости. |
| [Hashtable](https://docs.microsoft.com/ru-ru/dotnet/api/system.collections.hashtable) | Представляет коллекцию пар «ключ-значение», которые упорядочены по хэш-коду ключа. |
| [Queue](https://docs.microsoft.com/ru-ru/dotnet/api/system.collections.queue) | Представляет коллекцию объектов, которая обслуживается в порядке поступления (FIFO). |
| [Stack](https://docs.microsoft.com/ru-ru/dotnet/api/system.collections.stack) | Представляет коллекцию объектов, которая обслуживается в обратном порядке (LIFO). |

Пространство имен System.Collections.Specialized предоставляет специализированные и строго типизированные классы коллекций, такие как коллекции строк, связанные списки и гибридные словари.

Следует понимать, что коллекция сама по себе представляет объект определенного класса, не зависящий от хранимых в ней объектов. Коллекцию можно воспринимать как «коробку», в которой лежат объекты. Эта «коробка» позволяет складывать в нее объекты, доставать их оттуда, перемещать, узнавать количество объектов и т.д.

Одним из типов коллекций является объектный тип ArrayList (System.Collections). Объявление коллекции выглядит следующим образом:

ArrayList <название> = new ArrayList();

**Основные методы и свойства ArrayList**

.add(Object object) – добавление объекта в коллекцию

.remove (Object object) – удалить объект из коллекции

.get(index) – получить объект из коллекции по индексу

.clear() – очистить коллекцию (удалить все элементы)

.Item(Int32) – получить или задать элемент по указанному индексу

.Count – получить размер коллекции (количество элементов)

Описание множественной связи на примере класса Дерево:

// объявление класса Дерево

class Tree {

// Описание свойства name

...

// Описание свойства Фрукты

private ArrayList fruits = new ArrayList();

// Получить все фрукты на данном дереве

public ArrayList getFruits() {

return fruits;

}

// Добавить фрукт

public void addFruit(Fruit fruit) {

fruits.add(fruit);

}

// Удалить фрукт

public void removeFruit(Fruit fruit) {

fruits.remove(fruit);

}

}

## Получение элемента коллекции

Как уже отмечалось, получение элемента коллекции выполняется с помощью метода .Item(int index). Так как все объекты, хранящиеся в коллекции, приводятся к типу Object, необходимо выполнить обратное преобразование считываемого элемента к его исходному типу. Приведение к типу выполняется в соответствии со следующим синтаксисом:

(<Название типа>) <ссылка на объект>

Например, пусть tree1 – ссылка на объект класса Tree. Класс Tree содержит коллекцию fruits объектов типа Fruit, доступ к коллекции fruits обеспечивает метод getFruits(). Требуется вывести на экран названия (св-во name) всех фруктов дерева tree1:

// перечисляем все элементы коллекции tree1.getFruits()

for (int i=0; i<tree1.getFruits().size(); i++) {

// получаем i-й элемент коллекции, приводим к

// типу Fruit и присваиваем его ссылке fruit

**Fruit fruit = (Fruit)tree1.getFruits().get(i);**

// выводим на экран значение свойства name

// объекта fruit, который представляет i-й элемент коллекции

Console.WriteLine(fruit.getName());

Вариант № 1

а) **Смоделировать структуру предприятия:**

|  |  |
| --- | --- |
| **Классы** | **Свойства** |
| Фирма | название (get, set) |
| Отдел | название (get, set)  количество сотрудников (get, set) |
| Сотрудник | фио (get, set)  должность (get, set)  оклад (get, set) |

Создать один объект класса Фирма, два объекта – Отдела, и три объекта - Сотрудника, задать значения свойств, вывести на экран.

б) Связать между собой классы Фирма, Отдел и Сотрудник, так, чтобы каждая фирма содержала отделы, каждый отдел содержал сведения в какой фирме он находится и каких сотрудников он содержит, каждый сотрудник – в каком отделе он работает.

* Добавить в класс Фирма множественное свойство «отделы» (get) и методы «добавить отдел» (add), «удалить отдел» (remove).
* Добавить в класс Отдел свойство «фирма» (get) и свойство «сотрудники» (get), а также методы «добавить сотрудника» (add), «удалить сотрудника» (remove). Удалить set для свойства «количество сотрудников» и сделать так, чтобы данное свойство рассчитывалось автоматически (на основе свойства «сотрудники»).
* Добавить в класс «Сотрудник» свойство «Отдел»

Создать один объект класса Фирма, в данную фирму добавить два объекта – Отдела, в первый отдел добавить два сотрудника, во второй – одного сотрудника. Вывести на экран сотрудников, работающих в одном из отделов и их количество.

в) добавить в класс Фирма метод, осуществляющий поиск сотрудника по ФИО. Метод содержит входной параметр ФИО (String) и возвращает значение типа Сотрудник. Осуществить поиск в main(), вывести значения свойств найденного сотрудника на экран, и отобразить в каком отделе он работает.

Вариант № 2

а) **Смоделировать структуру банка:**

|  |  |
| --- | --- |
| **Классы** | **Свойства** |
| Банк | название (get, set) |
| Филиал | название (get, set)  общая сумма вкладов (get, set) |
| Вклад | фио вкладчика (get, set)  сумма вклада (get, set) |

Создать один объект класса Банк, два объекта – Филиала, и три вклада, задать значения свойств, вывести на экран.

б) Связать между собой классы Банк, Филиал и Вклад, так, чтобы каждый банк содержал филиалы, каждый филиал содержал указание в каком банке он находится, и какие вклады он содержит, каждый вклад – в каком филиале он размещен.

* Добавить в класс Банк множественное свойство «филиалы» (get) и метод «добавить филиал» (add)
* Добавить в класс Филиал свойство «банк» (get) и свойство «вклады» (get), а также методы «добавить вклад» (add), «удалить вклад» (remove). Удалить set для свойства «общая сумма вкладов» и сделать так, чтобы данное свойство рассчитывалось автоматически (на основе свойства «вклады»).
* Добавить в класс «Вклад» свойство «Филиал»,
* Удалить из класса «Вклад» set для свойства «сумма вклада» и добавить   
  метод «пополнить счет (сумма)»

Создать один объект класса Банк, в данный банк добавить два филиала, в каждый из филиалов добавить по два вклада. Вывести на экран вклады одного из филиалов, и общую сумму вкладов по этому филиалу. Выполнить пополнение счета одного из вкладов и снова вывести информация о вкладах на экран.

в) добавить в класс Банк метод, осуществляющий поиск вклада по ФИО вкладчика. Метод содержит входной параметр ФИО (String) и возвращает значение типа Вклад. Осуществить поиск в main(), вывести сумму найденного вклада на экран, и отобразить, в каком филиале он размещен.

Вариант № 3

а) **Смоделировать структуру аэропорта:**

|  |  |
| --- | --- |
| **Классы** | **Свойства** |
| Аэропорт | название (get, set) |
| Летательный аппарат | название (get, set)  макс. количество пассажиров (get, set) |
| Пассажир | фио (get, set)  №посадочного места (get, set) |

Создать один объект класса Аэропорт, два объекта класса Летательный аппарат, и три объекта - Пассажира, задать значения свойств, вывести на экран.

б) Связать между собой классы Аэропорт, Летательный аппарат и Пассажир, так, чтобы каждый аэропорт содержал сведения о своих летательных аппаратах, каждый летательный аппарат – в каком аэропорте он находится и каких пассажиров он содержит, каждый пассажир – на какой л/а он назначен.

* Добавить в класс Аэропорт множественное свойство «летательные\_аппараты» (get) и методы «добавить л/а» (add), «удалить л/а» (remove).
* Добавить в класс Летательный аппарат свойство «аэропорт» (get) и свойство «пассажиры» (get), а также методы «добавить пассажира» (add), «удалить пассажира» (remove).
* Добавить в класс Пассажир свойство «Летательный аппарат»

Создать один объект класса Аэропорт, в данный аэропорт добавить два объекта – Летательных аппарата, в первый л/а добавить одного пассажира, во второй – трех пассажиров. Вывести на экран пассажиров, назначенных в один из летательных аппаратов.

в)добавить в класс Аэропорт метод, осуществляющий поиск пассажира по названию л/а и номеру посадочного места. Метод содержит входные параметры Название л/а (String) и № посадочного места (int) и возвращает значение типа Пассажир. Осуществить поиск в main(), вывести значения свойств найденного пассажира на экран.

Вариант № 4

а) **Смоделировать структуру библиотеки:**

|  |  |
| --- | --- |
| **Классы** | **Свойства** |
| Библиотека | название (get, set) |
| Отдел (по жанрам) | название жанра (get, set)  количество изданий (get, set) |
| Издание | название (get, set)  автор (get, set)  год издания (get, set) |

Создать один объект класса Библиотека, два отдела, и три издания, задать значения свойств, вывести на экран.

б) Связать между собой классы Библиотека, Отдел и Издание, так, чтобы каждая библиотека содержала информацию какие отделы она содержит, каждый отдел – в какой библиотеке он находится, и какие издания он содержит, каждое издание – в каком отделе оно размещено.

* Добавить в класс Библиотека множественное свойство «отделы» (get) и методы «добавить отдел» (add), «удалить отдел» (remove).
* Добавить в класс Отдел свойство «библиотека» (get) и свойство «издания» (get), а также методы «добавить издание» (add), «удалить издание» (remove). Удалить set для свойства «количество изданий» и сделать так, чтобы данное свойство рассчитывалось автоматически (на основе свойства «издания»).
* Добавить в класс «Издание» свойство «Отдел»

Создать один объект класса Библиотека, в данную библиотеку добавить три объекта – Отдела, в первый отдел добавить два издания, во второй и третий – по одному изданию. Вывести на экран название и количество изданий по каждому отделу данной библиотеки. Удалить второй отдел, повторить вывод всех отделов библиотеки.

в) добавить в класс Библиотека метод, осуществляющий поиск изданий по заданному году выпуска. Метод содержит входной параметр год выпуска (int) и возвращает коллекцию или массив объектов типа Издание. Осуществить поиск в main(), вывести найденные издания на экран, и отобразить, в каком отделе они размещены.

Вариант № 5

а) **Смоделировать структуру компании сотовой связи:**

|  |  |
| --- | --- |
| **Классы** | **Свойства** |
| Компания | Название (get, set) |
| Тариф | название (get, set) |
| Абонент | фио (get, set)  номер телефона (get, set)  остаток на счете (get, set) |

Создать два объекта класса Компания, два тарифа, и три абонента, задать значения свойств, вывести на экран.

б) Связать между собой классы Компания, Тариф и Абонент, так, чтобы каждая Компания содержала Тарифы, каждый Тариф содержал информацию, о том, в какой компании он используется и каких абонентов он содержит, каждый абонент – к какому тарифу он подключен.

* Добавить в класс Компания множественное свойство «тарифы» (get) и метод «добавить тариф» (add)
* Добавить в класс Тариф свойство «компания» (get) и свойство «абоненты» (get), а также методы «добавить абонента» (add), «удалить абонента» (remove).
* Добавить в класс Тариф «количество абонентов» (get) и сделать так, чтобы данное свойство рассчитывалось автоматически (на основе множественного свойства «абоненты»).
* Добавить в класс «Абонент» свойство «Тариф»,
* Удалить из класса «Абонент» set для свойства «остаток на счете» и добавить   
  метод «пополнить счет (сумма)»

Создать два объекта класса Компания, в каждую компанию добавить по одному тарифу, на первый тариф в первой компании подключить два абонента, на второй – одного абонента. Вывести на экран абонентов одной из компаний. Выполнить пополнение счета одного из абонентов и снова вывести информация об абонентах на экран.

в) добавить в класс Компания метод, осуществляющий поиск абонента по номеру телефона. Метод содержит входной параметр Номер телефона (String или int) и возвращает значение типа Абонент. Осуществить поиск в main(), вывести фио и тариф найденного абонента на экран.

Вариант № 6

а) **Смоделировать структуру автосалона:**

|  |  |
| --- | --- |
| **Классы** | **Свойства** |
| Автосалон | название (get, set) |
| Автомобиль (марка) | название марки (get, set)  макс. количество пассажиров (get, set)  стоимость (get, set)  количество на складе(get, set)  boolean наличие (get, set) |
| Заявка на покупку | фио покупателя (get, set)  номер телефона (get, set) |

Создать один объект класса Автосалон, два автомобиля, и три заявки, задать значения свойств, вывести на экран.

б) Связать между собой классы Автосалон, Автомобиль и Заявка на покупку, так, чтобы каждый Автосалон содержал Автомобили, каждый Автомобиль содержал информацию, о том, в каком автосалоне он выставлен на продажу и о заявках на его покупку, каждая заявка – к какому автомобилю она относится.

* Добавить в класс Автосалон множественное свойство «автомобили» (get) и метод «добавить автомобиль» (add).
* Добавить в класс Автомобиль свойство «автосалон» (get) и свойство «заявки на покупку» (get), а также методы «добавить заявку» (add), «удалить заявку» (remove). Удалить set для свойства «наличие» и сделать так, чтобы данное свойство рассчитывалось автоматически (на основе свойств «количество на складе» и «заявки»).
* Добавить в класс «Заявка на покупку» свойство «автомобиль»,

Создать один объект класса Автосалон, добавить в него два автомобиля. На первый оформить одну заявку, на второй – три заявки. Вывести на экран информацию об одном из автомобилей и заявках на него.

в) добавить в класс Автосалон метод, осуществляющий поиск автомобиля по названию марки. Метод содержит входной параметр Название марки (String) и возвращает значение типа Автомобиль. Осуществить поиск в main(), вывести информацию о найденном автомобиле на экран.

Вариант № 7

а) **Смоделировать структуру музыкальной коллекции:**

|  |  |
| --- | --- |
| **Классы** | **Свойства** |
| Коллекция | название (get, set)  фио владельца (get, set) |
| Музыкальный носитель (альбом) | автор/группа (get, set)  жанр (get, set)  год выпуска (get, set)  общая продолжительность звучания (get, set) |
| Музыкальное произведение | название (get, set)  продолжительность (get, set) |

Создать один объект класса Коллекция, два музыкальных носителя (альбома), и три музыкальных произведения, задать значения свойств, вывести на экран.

б) Связать между собой классы Коллекция, Музыкальный носитель и Музыкальное произведение, так, чтобы каждая коллекция содержала музыкальные носители, каждый носитель содержал сведения о том, в какой коллекции он находится, и какие музыкальные произведения он содержит, каждое музыкальное произведение – на каком носителе оно содержится.

* Добавить в класс Коллекция множественное свойство «носители» (get) и методы «добавить носитель» (add), «удалить носитель» (remove).
* Добавить в класс Музыкальный носитель свойство «коллекция» (get) и свойство «музыкальные произведения» (get), а также методы «добавить музыкальное произведение» (add), «удалить музыкальное произведение» (remove). Удалить set для свойства «общая продолжительность звучания» и сделать так, чтобы данное свойство рассчитывалось автоматически (на основе свойства «музыкальные произведения»).
* Добавить в класс «Музыкальное произведение» свойство «Музыкальный носитель»

Создать один объект класса Коллекция, в данную коллекцию добавить два носителя - альбома, в первый альбом добавить два произведения, во второй – одно произведение. Вывести на экран музыкальные произведения одного из носителей и общую продолжительность звучания альбома.

в) добавить в класс Коллекция метод, осуществляющий поиск музыкального произведения по названию. Метод содержит входной параметр Название (String) и возвращает значение типа Музыкальное произведение.

Осуществить поиск в main(), вывести информацию по найденному произведению на экран, и отобразить на каком носителе оно содержится.

Вариант № 8

а) **Смоделировать структуру реестра городского жилья:**

|  |  |
| --- | --- |
| **Классы** | **Свойства** |
| Город | название (get, set) |
| Здание | название улицы (get, set)  номер дома (get, set)  базовая ежемесячная оплата за кв.м площади (get, set) |
| Помещение | номер (get, set)  площадь (get, set) |

Создать один объект класса Город, два здания, и три объекта - помещения, задать значения свойств, вывести на экран.

б) Связать между собой классы Город, Здание и Помещение, так, чтобы каждый город содержал здания, а каждое здание содержало помещения; также каждое помещение должно содержать информацию о том, к какому зданию оно относится.

* Добавить в класс Город множественное свойство «здания» (get) и методы «добавить здание» (add), «удалить здание» (remove).
* Добавить в класс Здание свойство «помещения» (get), а также методы «добавить помещение» (add), «удалить помещение» (remove). Добавить свойство «общая площадь» (get), рассчитываемое как суммарная площадь всех помещений в здании.
* Добавить в класс «Помещение» свойство «здание»

Создать один объект класса Город, два здания, в первое добавить два объекта – помещения, во вторую – одно помещение. Вывести на экран информацию о первом здании (значения всех свойств, включая общую площадь) и помещениях в этом здании.

в) добавить в класс Город метод, осуществляющий поиск здания по названию улицы и номеру дома. Метод содержит входные параметры Название улицы (String) и Номер дома (int или String) возвращает значение типа Здание. Осуществить поиск здания в main(), вывести найденное здание (значения всех свойств) на экран.

Вариант № 9

а) **Смоделировать структуру зоопарка:**

|  |  |
| --- | --- |
| **Классы** | **Свойства** |
| Зоопарк | название (get, set) |
| Вольер/клетка | номер (get, set)  размер (get, set)  макс. количество животных (get, set)  текущее количество животных (get, set) |
| Животное | название (get, set)  boolean хищник (get, set) |

Создать один объект класса Зоопарк, два объекта – Клетки, и три объекта - Животных, задать значения свойств, вывести на экран.

б) Связать между собой классы Зоопарк, Клетка и Животное, так, чтобы каждый зоопарк содержал клетки, в каждой из клеток содержатся животные, каждое животное «знает» в какой клетке оно содержится.

* Добавить в класс Зоопарк множественное свойство «клетки» (get) и методы «добавить клетку» (add), «удалить клетку» (remove).
* Добавить в класс Клетка свойство «животные» (get), а также методы «добавить животное» (add), «удалить животное» (remove). Удалить set для свойства «текущее количество животных» и сделать так, чтобы данное свойство рассчитывалось автоматически (на основе свойства «животные»).
* Добавить в класс «животное» свойство «клетка»

Создать один объект класса Зоопарк, в нем две клетки, в первую добавить два животных, во вторую одно животное. Вывести на экран животных, содержащихся в одной из клеток.

в) добавить в класс Зоопарк метод, осуществляющий поиск животного по названию. Метод содержит входной параметр Название (String) и возвращает значение типа Животное. Осуществить поиск в main(), вывести значения свойств найденного животного на экран.

Вариант № 10

а) **Смоделировать структуру автоматизированного банкомата:**

|  |  |
| --- | --- |
| **Классы** | **Свойства** |
| Банк | название (get, set) |
| Счет | номер (get, set)  PIN-код (get, set)  остаток (get, set) |
| Банкомат | идентификационный номер (get, set)  адрес (get, set) |

Создать один объект класса Банк, два объекта – Счета, и три объекта - Банкомата, задать значения свойств, вывести на экран.

б) Связать между собой классы Банк, Счет и Банкомат, так, чтобы каждый Банк содержал Счета и Банкоматы, каждый Счет содержал информацию, о том, в каком банке он содержится, каждый Банкомат – какой банк он обслуживает.

* Добавить в класс Банк множественное свойство «счета» (get) и методы «добавить счет» (add), «удалить счет» (remove)
* Добавить в класс Банк множественное свойство «банкоматы» (get) и метод «добавить банкомат» (add)
* Добавить в класс Счет свойство «банк» (get)
* Добавить в класс Банкомат свойство «банк» (get)
* Удалить из класса «Счет» set для свойства «остаток на счете». Добавить в класс Счет методы «пополнить счет (сумма)» и «снять со счета (сумма)»

Создать один объект класса Банк, два счета и три банкомата в нем. Вывести на экран состояние одного из счетов. Выполнить снятие определенной суммы с данного счета и снова вывести информацию о состоянии данного счета на экран.

в) Добавить в класс банкомат метод «снять деньги со счета» с параметрами «PIN-код» и «сумма». Данный метод обращается к банку, обслуживаемому данным банкоматом, производит поиск счета по данному PIN-коду и производит снятие указанной суммы (вызовом метода «снять со счета (сумма)» у найденного счета). В main() проверить работу созданного метода.

Приложение 1

Справочные материалы

Классы в C#

<https://docs.microsoft.com/ru-ru/dotnet/csharp/programming-guide/classes-and-structs/classes>

<https://docs.microsoft.com/ru-ru/dotnet/csharp/tour-of-csharp/classes-and-objects>

Объекты

<https://docs.microsoft.com/ru-ru/dotnet/csharp/programming-guide/classes-and-structs/objects>

Коллекции

<https://docs.microsoft.com/ru-ru/dotnet/api/system.collections?view=netframework-4.7.2>

<https://msdn.microsoft.com/ru-ru/library/ybcx56wz%28v=vs.120%29.aspx?f=255&MSPPError=-2147217396#BKMK_SimpleCollection>

ArrayList

<https://docs.microsoft.com/ru-ru/dotnet/api/system.collections.arraylist?redirectedfrom=MSDN&view=netframework-4.7.2>