Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»
Филиал
«Минский радиотехнический колледж»

Учебный предмет
«Конструирование программ и языки программирования»

Инструкция

по выполнению лабораторной работы №9
«Разработка, создание объектов и использование их в программах»

Минск 2024 г.

Лабораторная работа № 9

Тема работы: «Разработка, создание объектов и использование их в программах»

1 Цель работы

Сформировать умения разрабатывать классы, создавать объекты и использовать их в программах

2 Задание

Необходимо реализовать спроектированные классы из лабораторной работы №8, для каждого класса создать по три объекта.

3 Оснащение работы

ПК, среда Visual Studio 2019, MSword.

4 Основные теоретические сведения

Класс является типом данных, определяемым пользователем. Он должен представлять собой одну логическую сущность, например, являться моделью реального объекта или процесса. *Элементами* класса являются *данные* и *функции*, предназначенные для их обработки.

Описание класса содержит *ключевое слово* class, за которым следует его *имя*, а далее в фигурных скобках — *тело* класса, то есть *список* его элементов. Кроме того, для класса можно задать его базовые классы (предки) и ряд необязательных атрибутов и спецификаторов, определяющих различные характеристики класса:

```
[атрибуты] [спецификаторы] class имя_класса [: предки]
    тело_класса
```

Обязательными являются только *ключевое слово* class, имя и тело класса. *Тело класса* — это *список* описаний его элементов, заключенный в *фигурные скобки*. *Список* может быть пустым, если *класс* не содержит ни одного элемента. Таким образом, простейшее описание класса может выглядеть так:

```
class Demo {}
```

Класс можно описывать непосредственно внутри пространства имен или внутри другого класса. В последнем случае *класс* называется *вложенным*.

- *Константы* класса хранят неизменяемые значения, связанные с классом.
- *Методы* реализуют вычисления или другие действия, выполняемые классом или экземпляром.
- *Свойства* определяют характеристики класса в совокупности со способами их задания и получения, то есть методами записи и чтения.
- *Конструкторы* реализуют действия по *инициализации экземпляров* или класса в целом.
- *Деструкторы* определяют действия, которые необходимо выполнить до того, как объект будет уничтожен.

- *Индексаторы* обеспечивают возможность доступа к элементам класса по их порядковому номеру.
- *Операции* задают действия с объектами с помощью знаков операций.
- *События* определяют уведомления, которые может генерировать класс.
- *Типы* — это типы данных, внутренние по отношению к классу.

Данные, содержащиеся в классе, могут быть переменными или константами и задаются в соответствии с правилами, рассмотренными в разделах "Переменные" и "Именованные константы". Переменные, описанные в классе, называются полями класса. При описании элементов класса можно также указывать атрибуты и спецификаторы, задающие различные характеристики элементов.

Синтаксис описания элемента данных:

```
[ атрибуты ] [ спецификаторы ] [ const ] тип имя [ = начальное_значение ]
```

По умолчанию элементы класса считаются закрытыми (private). Для полей класса этот вид доступа является предпочтительным. Все *методы класса* имеют непосредственный *доступ* к его закрытым полям.

Поля, описанные со спецификатором static, *а также константы существуют в единственном экземпляре для всех объектов класса, поэтому к ним обращаются не через имя экземпляра, а через имя класса. Если класс содержит только статические элементы, экземпляр класса создавать не требуется. Именно этим фактом мы пользовались во всех предыдущих листингах*.

Обращение к полю класса выполняется с помощью *операции доступа* (точка). Справа от точки задается имя поля, слева — имя экземпляра для обычных полей или *имя класса* для статических.

Поля со спецификатором readonly предназначены только для чтения. Установить *значение* такого поля можно либо при его описании, либо в конструкторе.

Метод — это *функциональный элемент* класса, который реализует вычисления или другие действия, выполняемые классом или экземпляром. Методы определяют поведение класса.

Метод представляет собой законченный фрагмент кода, к которому можно обратиться *по* имени. Он описывается один раз, а вызываться может столько раз, сколько необходимо. Один и тот же метод может обрабатывать различные данные, переданные ему в качестве аргументов.

Синтаксис метода:

```
[ атрибуты ] [ спецификаторы ] тип имя_метода ( [ параметры ] )
    тело_метода
```

Первая строка представляет собой *заголовок* метода. *Тело метода*, задающее действия, выполняемые методом, чаще всего представляет собой блок.

Тип определяет, *значение* какого типа вычисляется с помощью метода. Часто употребляется термин "метод возвращает *значение*". Если метод не возвращает никакого значения, в его заголовке задается тип void, а оператор return отсутствует.

Параметры используются для обмена информацией с методом. *Параметр* представляет собой локальную переменную, которая при вызове метода принимает *значение* соответствующего аргумента. *Область действия* параметра — весь метод.

Метод, не возвращающий значение, вызывается отдельным оператором, а метод, возвращающий значение, — в составе выражения в правой части оператора присваивания.

Параметры, описываемые в заголовке метода, определяют множество значений *аргументов*, которые можно передавать в метод. *Список* аргументов при вызове как бы накладывается на *список* параметров, поэтому они должны попарно соответствовать друг другу. Для каждого параметра должны задаваться его *тип* и *имя*.

При вызове метода выполняются следующие действия:

1. Вычисляются выражения, стоящие на месте аргументов.
2. Выделяется память под параметры метода в соответствии с их типом.
3. Каждому из параметров сопоставляется соответствующий аргумент (аргументы как бы накладываются на параметры и замещают их).
4. Выполняется тело метода.
5. Если метод возвращает значение, оно передается в точку вызова; если метод имеет тип void, управление передается на оператор, следующий после вызова.

Существуют два способа передачи параметров: по значению и по ссылке.

При передаче по значению метод получает копии значений аргументов, и операторы метода работают с этими копиями. Доступа к исходным значениям аргументов у метода нет, а, следовательно, нет и возможности их изменить.

При передаче по ссылке (по адресу) метод получает копии адресов аргументов, он осуществляет доступ к ячейкам памяти по этим адресам и может изменять исходные значения аргументов, модифицируя параметры.

В C# для обмена данными между вызывающей и вызываемой функциями предусмотрено четыре типа параметров:

- параметры-значения;
- параметры-ссылки — описываются с помощью ключевого слова ref ;
- выходные параметры — описываются с помощью ключевого слова out ;
- параметры-массивы — описываются с помощью ключевого слова params.

Ключевое слово предшествует описанию типа параметра. Если оно опущено, параметр считается параметром-значением. Параметр-массив может быть только один и должен располагаться последним в списке, например:

```
public int Calculate( int a, ref int b, out int c, params int[] d ) …
```

Параметр-значение описывается в заголовке метода следующим образом:

тип имя

Пример заголовка метода, имеющего один параметр-значение целого типа:

```
void P( int x )
```

Признаком параметра-ссылки является ключевое слово ref перед описанием параметра:

```
ref тип имя
```

Пример заголовка метода, имеющего один параметр-ссылку целого типа:

```
void P( ref int x )
```

При вызове метода в область параметров копируется адрес аргумента, и метод через него имеет доступ к ячейке, в которой хранится аргумент. Метод работает непосредственно с переменной из вызывающей функции и, следовательно, может ее изменить, поэтому если в методе требуется изменить значения параметров, они должны передаваться только по ссылке.

Внимание

При вызове метода на месте параметра-ссылки может находиться только ссылка на инициализированную переменную точно того же типа. Перед именем параметра указывается ключевое слово ref.

Каждый *объект* содержит свой экземпляр полей класса. Методы находятся в памяти в единственном экземпляре и используются всеми объектами совместно, поэтому необходимо обеспечить работу методов нестатических экземпляров с полями именно того объекта, для которого они были вызваны. Для этого в любой нестатический метод автоматически передается скрытый *параметр* this, в котором хранится *ссылка* на вызвавший функцию экземпляр.

В явном виде *параметр* this применяется для того, чтобы возвратить из метода ссылку на вызвавший *объект*, а также для идентификации поля в случае, если его имя совпадает с именем параметра метода.

Конструктор предназначен для инициализации объекта. Он вызывается автоматически при создании объекта класса с помощью *операции* new. Имя конструктора совпадает с именем класса. Ниже перечислены свойства конструкторов

- Конструктор *не возвращает значение*, даже типа void.
- Класс может иметь *несколько конструкторов* с разными параметрами для разных видов инициализации.
- Если программист не указал ни одного конструктора или какие-то поля не были инициализированы, полям *значимых типов* присваивается нуль, полям ссылочных типов — значение null.
- Конструктор, вызываемый без параметров, называется *конструктором по умолчанию*.

Часто бывает удобно задать в классе несколько конструкторов, чтобы обеспечить возможность инициализации объектов разными способами. Все конструкторы должны иметь разные сигнатуры.

Если один из конструкторов выполняет какие-либо действия, а другой должен делать то же самое плюс еще что-нибудь, удобно вызвать первый конструктор из второго.

Существует второй тип конструкторов — *статические конструкторы*, или *конструкторы класса*. *Конструктор* экземпляра инициализирует данные экземпляра, *конструктор* класса — данные класса.

Статический *конструктор* не имеет параметров, его нельзя вызвать явным образом. Система сама определяет момент, в который требуется его выполнить.

Свойства служат для организации доступа к полям класса. Как правило, свойство связано с закрытым полем класса и определяет методы его получения и установки. *Синтаксис* свойства:

```
[ атрибуты ] [ спецификаторы ] тип имя_свойства
{
    [ get код_доступа ]
    [ set код_доступа ]
}
```

Значения спецификаторов для свойств и методов аналогичны. Чаще всего свойства объявляются со спецификатором public. *Код доступа* представляет собой блоки операторов, которые выполняются при получении (get) или установке (set) свойства. Может отсутствовать либо часть get, либо set, но не обе одновременно.

Если отсутствует часть set, свойство доступно только для чтения (*read-only*), если отсутствует часть get, свойство доступно только для записи (write-only).

5. Порядок выполнения работы

1. Выделить ключевые моменты задачи.
2. Построить алгоритм и теоретическую объектную модель решения задачи.
3. Запрограммировать полученные алгоритмы и объектную модель.

6. Форма отчета о работе

Лабораторная работа № ___

Номер учебной группы _____

Фамилия, инициалы учащегося _____

Дата выполнения работы _____

Тема работы: _____

Цель работы: _____

Оснащение работы: _____

Результат выполнения работы: _____

7. Контрольные вопросы и задания

1. Класс – это…
2. Перечислите элементы класса.
3. Синтаксис объявления класса.
4. Метод – это…
5. Параметр – это…
6. Перечислите способы передачи параметров. В чем их суть?

8. Рекомендуемая литература

1. Рихтер, Дж. CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 на языке C# / Дж. Рихтер. СПб. : Изд-во Питер, 2021. 896 с.

2. Прайс, М. Дж. C# 10 и .NET 6. Современная кросс-платформенная разработка / М. Дж. Прайс. СПб : Изд-во Питер, 2023. 848 с.

3. Васильев, А.Н. Программирование на C# для начинающих. Особенности языка / А.Н. Васильев. М. : Эксмо, 2022. 528 с.

4. Фримен, А. ASP.NET Core 3 с примерами на C# для профессионалов / А. Фримен. СПб. : Изд-во Вильямс, 2021. 1184 с.