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 №16
«Разработка, отладка и испытание делегатов в программах»

Минск 2024 г.

Лабораторная работа № 16

Тема работы: «Разработка, отладка и испытание делегатов в программах»

1 Цель работы

Сформировать умения разрабатывать делегаты и использовать их в программах.

2 Задание

Номер варианта соответствует номеру по списку в журнале.

Необходимо реализовать методы по вариантам из лабораторной работы № 4 и вызвать их с помощью делегатов.

3 Оснащение работы

ПК, среда Visual Studio 2019, MSword.

4 Основные теоретические сведения

Делегат - это особый тип. И объявляется он по особому:

```
delegate int MyDelegate (string x);
```

Тут все просто, есть ключевое слово **delegate**, а дальше сам делегат с именем MyDelegate, возвращаемым типом int и одним аргументом типа string. По факту же при компиляции кода в CIL — компилятор превращает каждый такой тип-делегат в **одноименный тип-класс** и все экземпляры данного типа-делегата по факту являются экземплярами соответствующих типов-классов. Каждый такой класс наследует тип MulticastDelegate от которого ему достаются методы Combine и Remove, содержит конструктор с двумя аргументами **target** (Object) и **methodPtr** (IntPtr), поле **invocationList** (Object), и три собственных метода **Invoke, BeginInvoke, EndEnvoke**. Объявляя новый тип-делегат мы сразу через синтаксис его объявления **жестко определяем сигнатуру допустимых методов**, которыми могут быть инициализированы экземпляры такого делегата. Это сразу влияет на сигнатуру автогенерируемых методов Invoke, BeginInvoke, EndEnvoke, поэтому эти методы и не наследуются от базового типа, а определяются для каждого типа-делегата отдельно.

Экземпляр же такого делегата стоит понимать, как **ссылку на конкретный метод или список методов**, который куда-то будет передан и скорее всего выполнен уже на той стороне. Причем клиент не сможет передать с методом значение аргументов с которыми он будет выполнен (если только мы этого ему не позволим), или поменять его сигнатуру. Но он сможет определить логику работы метода, то есть его тело.
Это удобно и безопасно для нашего кода так как мы знаем какой тип аргумента передать в делегат при выполнении и какой возвращаемый тип ожидать от делегата.

Если пофантазировать, то можно предоставить право передачи аргумента для делегатов клиентской стороне, например, создать метод с аргументом делегатом и аргументом который внутри нашего метода этому делегату будет передан, что позволит клиенту задавать еще и значение аргумента для метода в делегате. Например таким образом.

```
void MyFunc(myDelegate deleg, int arg){deleg.Invoke(arg);}
```

Создавая в коде экземпляр делегата его конструктору передается метод (подойдет и экземплярный и статический, главное, чтобы сигнатура метода совпадала с сигнатурой делегата). Если метод экземплярный, то в поле **target** записывается ссылка на экземпляр-владелец метода (он нужен нам, ведь если метод экземплярный, то это как минимум подразумевает работу с полями этого объекта target), а в **methodPtr** ссылка на метод. Если метод статический, то записываются в поля **target** и **methodPtr** будут записаны null и ссылка на метод соответственно. Инициализировать переменную делегата можно через создание экземпляра делегата:

```
MyDeleg x = new MyDeleg(MyFunc);
```

Или упрощенный синтаксис без вызова конструктора:

```
MyDeleg x = MyFunc;
```

Организовать передачу/получение экземпляра делегата можно по-разному. Так как делегат это в итоге всего лишь тип-класс, то можно свободно создавать поля, свойства, аргументы методов и т.д. конкретного типа делегата. Методы делегатов:
Invoke — синхронное выполнение метода который храниться в делегате.
BeginInvoke, EndEnvoke — аналогично но асинхронное.
Вызывать выполнение методов хранящихся в делегате можно и через упрощенный синтаксис:

```
delegInst.Invoke(argument);
```

это аналогично записи:

```
delegInst(argument);
```

А зачем делегату поле invocationList?

Поле **invocationList** имеет значение null для экземпляра делегата пока делегат хранит ссылку на один метод. Этот метод можно всегда перезаписать на другой приравняв через "=" переменной новый экземпляр делегата (или сразу нужного нам метода через упрощенный синтаксис). Но так же можно создать цепочку вызовов, когда делегат хранит ссылки на более чем один метод. Для этого нужно вызвать метод **Combine**:

```
MyDeleg first = MyFunc1;
MyDeleg second = MyFunc2;
first = (MyDeleg) Delegate.Combine(first, second);
```

Метод **Combine** возвращает ссылку на новый делегат в котором поля target и methodPtr пусты, но invocationList, который содержит две ссылки на делегаты: тот что был раньше в переменной first и тот что еще хранится в second. Надо понимать что добавив третий делегат через метод Combine и записав его результат в first, то метод вернет ссылку на новый делегат с полем invocationList в котором будет коллекция из трех ссылок, а делегат с двумя ссылками будет удален сборщиком мусора при следующем цикле очистки. При выполнении такого делегата все его методы будут выполнены по очереди. Если сигнатура делегата предполагает получение параметров то параметры будут для всех методов иметь одно значение. Если есть возвращаемое значение, то мы можем получить лишь значение последнего в списке метода. Метод **Remove** же в свою очередь производит поиск в списке делегатов по значению объекта-владельца и методу, и в случае нахождения удаляет первый совпавший.

```
Deleg first = first.Remove(MyFunc2);
```

Переопределенные для делегатов операторы **+=** и **-=** являются аналогами методов Combine и Remove:

```
first = (Deleg) Delegate.Combine(first, second);
```

аналогично следующей записи:

```
first += MyFunc2;
```

И соответственно:

```
first = first.Remove(MyFunc2);
```

аналогично следующей записи:

```
first -= MyFunc;
```

Стоит сказать что делегаты могут быть **обобщенными** (Generic), что является более правильным подходом к созданию отдельных делегатов для разных типов.

Также стоит упомянуть что **библиотека FCL уже содержит наиболее популярные типы делегатов (обобщенные и нет)**. Например делегат **Action<T>** представляет собой метод без возвращаемого значения но с аргументом, а **Fucn<T, TResult>** и с возвращаемым значением и аргументом.

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. Дайте определение понятию «Делегат».
2. Перечислите методы делегатов и дайте их характеристику.
3. Поясните понятие «Обобщенные делегаты». Назовите их функциональное назначение.

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.