#9 Делегаты, события и лямбда-выражения.

Задание:

1. Используя класс-коллекцию из предыдущей работы (#8), познакомьтесь с делегатами и событиями:
   https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/delegates/
   https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/events/
   a. Разработайте свой класс **EventArgs**, содержащий какую-то полезную нагрузку;
   b. Добавьте в свою иерархию 2 различных события с вашим кастомным классом **EventArgs**. Осуществите переопределение метода-триггера события, если необходимо;
   c. Добавьте в класс-коллекцию методы, реагирующий на события иерархии;
   d. Осуществите отписку от событий в деструкторе класса-коллекции.
   e. Продемонстрируйте работу событий и методов-подписчиков.
2. Изучите стандартные типы делегатов **Action**, **Func**, **Predicate**:
   https://docs.microsoft.com/en-us/dotnet/api/system.action?view=netframework-4.8
   https://docs.microsoft.com/en-us/dotnet/api/system.func-1?view=netframework-4.8
   https://docs.microsoft.com/en-us/dotnet/api/system.predicate-1?view=netframework-4.8
3. Добавьте в класс-коллекцию методы, принимающие параметрами каждый из стандартных типов делегатов, например:
   a. **Action** – выполнение полезной нагрузки с каждым объектом коллекции;
   b. **Func** – функция-маппер, преобразующая объект коллекции к новому типу;
   c. **Predicate** – поиск объектов, удовлетворяющих условию.
4. Используйте **lambda**-синтаксис везде, где это возможно:
   https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/statements-expressions-operators/lambda-expressions
5. Используйте **try-catch** для обработки потенциально опасных мест кода.

Повышенный уровень:

1. Познакомьтесь с паттерном проектирования «Фабричный метод»: https://ru.wikipedia.org/wiki/Фабричный_метод_(шаблон_проектирования)
2. Создайте статический класс **LoggerFactory** с методом **GetLogger**, который возвращает объект наследника **ILogger**, в зависимости от значения параметра *loggerType*.
3. Реализуйте *loggerType* с использованием перечисления **LoggerType** со значениями: **File** и **Console**.
4. Реорганизуйте вашу архитектуру таким образом, чтобы из секции логгеров из вашей библиотеки был доступен лишь **LoggerFactory**, **ILogger** и **LoggerType**.
5. Конкретные реализации логгера (**FileLogger** и **ConsoleLogger**) по-прежнему остаются *синглтонами*.
6. Подключите вашу библиотеку к проекту текущей лабораторной работы.
7. Осуществляйте логгирование, используя ваш кастомный логгер.
   Note: вы можете переопределить формат сообщений, выводимых логгером, при необходимости.

Вопросы:

1. Что такое делегаты? Каково их предназначение? Каким типом данных является делегат: ссылочным или значимым?
2. Как создать делегат?
3. Как можно присвоить делегату адрес метода?
4. Какими способами можно вызвать делегат? Возможно ли присвоить делегату сразу несколько разных методов?
5. Что такое событие? Какой синтаксис объявления события?
6. Как события связаны с делегатами? Что такое “***object*** *sender*”?
7. Что такое ковариантность и контравариантность делегатов? В чем их преимущества?
8. В чем разница между **Func** и **Action**?

Повышенный уровень:

1. В чем заключается принцип работы паттерна «*фабричный метод*»?

2. Как можно реализовать паттерн «*одиночка*»? Как он реализуется в .NET? Какие ограничения должны быть наложены на класс, чтобы он мог называться «одиночкой»?