Практическая работа №7

Исключения

Цель работы: изучить обработку исключений в языке C#.

Теоретическая часть

Исключениями, или исключительными ситуациями, обычно называются аномалии, которые могут возникать во время выполнения и которые трудно, а порой и вообще невозможно, предусмотреть во время программирования приложения.

В .NET существует структурированная обработка исключений, которая представляет собой методику, предназначенную для работы с исключениями, которые могут возникать на этапе выполнения.

Программирование со структурированной обработкой исключений подразумевает использование четырех следующих связанных между собой сущностей:

- тип класса, который представляет детали исключения;
- член, способный генерировать (`throw`) в вызывающем коде экземпляр класса исключения при соответствующих обстоятельствах;
- блок кода на вызывающей стороне, ответственный за обращение к члену, в котором может произойти исключение;
- блок кода на вызывающей стороне, который будет обрабатывать (или перехватывать (`catch`) исключение в случае его возникновения.

Блоки `try` инкапсулируют код, формирующий часть нормальных действий программы, которые потенциально могут столкнуться с серьезными ошибочными ситуациями.

Блоки `catch` инкапсулируют код, который обрабатывает ошибочные ситуации, происходящие в коде блока `try`. Это также удобное место для протоколирования ошибок.

Блоки `finally` инкапсулируют код, очищающий любые ресурсы или выполняющий другие действия, которые обычно нужно выполнить в конце блоков `try` или `catch`. Важно понимать, что этот блок выполняется независимо от того, сгенерировано исключение или нет.

Все определяемые на уровне пользователя и системы исключения в конечном итоге всегда наследуются от базового класса `System.Exception`, который, в свою очередь, наследуется от класса `System.Object`.

Если планируется создать действительно заслуживающий внимания специальный класс исключения, необходимо позаботиться о том, чтобы он соответствовал наилучшим рекомендациям .NET. Он должен:

- наследоваться от `ApplicationException`;
- сопровождаться атрибутом `[System.Serializable]`;
- иметь конструктор по умолчанию;
- иметь конструктор, который устанавливает значение унаследованного свойства `Message`;
- иметь конструктор для обработки "внутренних исключений";
- иметь конструктор для обработки сериализации типа.

Практическая часть

1) Разработать класс `Item`, который будет отображать информацию о товарах, а именно: артикул, наименование, цвет, стоимость.

2) Создать класс `Shop`, в котором товары будут хранится в виде приватного поля `List<Item> items`.
3) Разработать собственный класс исключения `ExistingItemCodeException`, которое будет генерироваться при попытке вставить товар с уже существующим артикулом. В классе исключения необходимо добавить свойство `Item`, в котором будет храниться информация о товаре, при вставке которого возникло исключение. Разработать конструктор `ExistingItemCodeException (Item item)`. В теле конструктора проинициализировать свойство `Data` информацией о товаре.
4) Разработать метод `AddItem`, который будет инкапсулировать вставку информации о новом товаре. Использовать блок `throw` для генерации исключения `ExistingItemCodeException`.
5) Сравнить различные случаи генерации исключения:

```
//1 case
try
     {
         throw new ExistingItemCodeException();
     }
     catch (ExistingItemCodeException ex)
     {
         throw ex;
     }
//2 case
try
     {
         throw new ExistingItemCodeException();
     }
     catch (ExistingItemCodeException ex)
     {
         throw;
     }
```