Программирование на C#

Семинар №6

Модуль №2

Тема:

Решение задач на тему иерархии типов и полиморфизм

Вспоминаем Полиморфизм

```
class A    {
    public void PrintA() { Console.Write("A"); }
}
class B : A    {
    public void PrintA() { Console.Write("B"); }
}
class Program  {
    static void Main()      {
            A objA;
            objA = new B();
            objA.PrintA();
     }
}
```

Происходит не подмена, а сокрытие

```
> A
```

Модификатор подмены метода базового класса

```
> B
```

```
class A    {
   public virtual void PrintA()
       { Console.Write("A"); }
}
class B : A    {
   public override void PrintA()
       { Console.Write("B"); }
}
class Program    {
   static void Main()      {
           A objA;
           objA = new B();
           objA.PrintA();
       }
    }
```

Замещение базового метода

Self01: Не Читатель, а Писатель

Необходимо разработать консольное приложение, работающее со следующими классами:

- Класс **Writer**, содержащий следующие элементы:
  - Виртуальный метод **void Write(string s),** выводящий строку s на консоль;

- Класс **CamelWriter**, наследник класса **Writer**, содержащий следующие элементы:
  - Переопределенный метод **void Write(string s),** выводящий строку s на консоль, заменив в ней все большие буквы на маленькие, сделав все буквы после пробелов большими и удалив пробелы; Например, qWe rTy -> QweRty

- Класс **CipheredWriter**, наследник класса **Writer**, содержащий следующие элементы:
  - Закрытое поле **int shift**;
  - Конструктор **CipheredWriter(int shift),** инициализирующий поле значением аргумента. При значении аргумента 0 выбросить **ArgumentException**, сообщив о небезопасном шифровании;
  - Переопределенный метод **void Write(string s),** выводящий строку **s** на консоль, циклически сдвинув код каждого символа на **shift**; Пример, shift = 1 xyz -> yza

В основной программе создать массив ссылок типа **Writer** и связать их с объектами каждого из трёх реализованных классов. Организовать два режима работы:

- Получать от пользователя тестовые строки и выводить на экран результат их преобразования каждым из экземпляров из массива отдельно.
- Получать от пользователя тестовые строки и каждую обрабатывать конвейером (т.е. последовательно) из объектов массива.

Self02: Круги и Прямоугольники (1)

Необходимо реализовать следующие классы:

1. Класс **Geometry** - описывает геометрическую фигуру с центром в точке **(X,Y)**. Содержит два вещественных поля для координат точки с доступом только для чтения и методы, допускающие переопределение в наследниках:
    1. **GetSquare()** - метод возвращающий площадь фигуры;
    2. **IsPointInside(Geometry geom)** - метод возвращает **true**, если точка **(X,Y)** объекта класса, к которому применяется вызов метода, лежит строго внутри передаваемой в качестве параметра фигуры. При площади фигуры **geom**, равной нулю, выбрасывает исключение **ArgumentException**.
2. Класс **Circle** - наследник класса **Geometry**, описывает круг с радиусом **radius** и центром в точке **(X,Y)**;
3. переопределяет метод **GetSquare()** для вычисления площади круга.
4. Класс **Rectangular** - наследник класса **Geometry**, описывает прямоугольник, у которого левый верхний угол лежит в точке **(X,Y)** и известны две стороны:
    1. **sideA** и **sideB**;
    2. переопределяет метод **GetSquare()** для вычисления площади прямоугольника.

В каждом классе все поля должны быть инкапсулированы, для доступа к значениям необходимо использовать свойства. Также для всех наследников класса **Geometry** нужно переопределить метод **ToString(),** чтобы он формировал строку с полной информацией о состоянии объекта.

В основной программе случайным образом инициировать массив длиной **10** из кругов и прямоугольников. Координаты точек должны лежать в интервале **[-50; 51)**, а их длины сторон – **[0; 10).** Отсортировать фигуры по возрастанию их площадей. Вывести информацию о сгенерированных объектах в консоль. Далее попросить пользователя описать одну из геометрических фигур на выбор. Создать данный объект. Определить, с какими объектами коллекции пересекается пользовательская фигура и вывести информацию об этих объектах и их площадь. Предусмотрите проверку корректности ввода, цикл повтора решения, обработку возникших исключений.

Self02: Круги и Прямоугольники (2)

Иерархию классов не забудьте разместить в библиотеке и подключить ее к консольному приложению.

В основной программе обрабатываем исключения, проверяем корректность ввода данных и организуем повторение решения.

Комментарии к коду обязательны.

Модельное решение задачи доступно по ссылке:
(https://replit.com/@olgamaksimenkova/CirclesAndRectangles)

Self03: Классы-изгои (1)

Необходимо разработать приложение, содержащее следующие классы:

1. Класс **Rogue**, содержащий следующие элементы:
    1. Закрытое поле **int health** и свойство доступа к нему (только для чтения);
    2. Приватное поле **int damage** и свойство доступа к нему (только для чтения);
    3. Приватное поле **bool poisoned** и свойство доступа к нему (только для чтения);
    4. Конструктор **Rogue(int health, int damage)**, инициализирующий поля переданными значениями и поле **poisoned** значением **false** . При значениях параметров **0** или меньше выбрасывается **ArgumentOutOfRangeException** с подходящими параметрами;
    5. Виртуальный метод **void Attack(Rogue target)**, наносящий здоровью цели случайный урон из диапазона **[damage -3 ; damage+3)** (минимальный урон **0**). Если **Rogue** мёртв, выбрасывается **Exception**. Если цель уничтожена, выбрасывается **ArgumentException**;
2. Класс **Thief** , наследник класса **Rogue**, содержащий следующие элементы:
    1. Метод **void Attack(Mage target)** переопределен так, что **Thief** атакует цель дважды; *Вызвать базовую версию метода можно с помощью ключевого слова base*
    2. Конструктор, соответствующий базовому;
3. Класс **Assassin**, наследник класса **Rogue**, содержащий следующие элементы:
    1. Метод **void Attack(Rogue target)** переопределен так, что в добавок к базовому поведению цель становится отравленной и получает **1** единицу урона во время каждой своей атаки;
    2. Конструктор, соответствующий базовому;

В основной программе создать два массива типа **Rogue** одинаковых размеров (количество элементов массивов - целое натуральное число, не превосходящее пяти - получить от пользователя), заполнить их экземплярами трёх реализованных классов. Объекты **Rogue** из разных массивов по очереди атакуют случайных живых **Rogue** из другого массива, пока все объекты **Rogue** из одного из массивов не будут уничтожены. Вывести информацию о выживших на экран (переопределив метод **ToString()** в реализованных классах).

Self03: Классы-изгои (2)

Иерархию классов не забудьте разместить в библиотеке и подключить ее к консольному приложению.

В основной программе обрабатываем исключения, проверяем корректность ввода данных и организуем повторение решения.

Комментарии к коду обязательны.

Модельное решение задачи доступно по ссылке: https://replit.com/@olgamaksimenkova/RougeClasses