**Задание 2. Наследование и полиморфизм**

2.1. Создать иерархию классов Person-Student-Teacher. Каждый класс – в своей сборке. В каждом классе должны быть свойства, а также виртуальная функция Print и переопределенная функция ToString(). Основная программа создает массив объектов Person или их наследников, после чего выдает его на экран. У каждого Teacher должен быть список Students, которыми он руководит, у каждого Student - Teacher, который им руководит.

**Замечание.** В процессе реализации возникнет такая ошибка как циклическая зависимость сборок: сборка Student зависит от сборки Teacher и наоборот. Для устранения этой ошибки рекомендуется создать класс Student без поля Teacher, после чего создать производный класс StudentWithAdvisor с полем Teacher в отдельной сборке.

2.2. Для классов Person-Student-Teacher реализовать и оттестировать ToString(), Equals(), GetHashCode().

2.3. Для классов Person-Student-Teacher реализовать статические методы RandomPerson, RandomStudent, RandomTeacher, которые возвращают случайного из некоторого статического массива.

2.4. С помощью is, as, GetType определить, сколько в массиве персон, студентов и преподавателей и перевести всех студентов на следующий курс.

2.5. Для классов Person-Student-Teacher реализовать глубокое клонирование, определив виртуальный метод Clone(). Клон должен возвращать точную копию по значению и типу. Проиллюстрировать Clone на примере контейнера персон - должны создаваться клоны объекты ровно тех типов, которые содержатся в исходном контейнере.

(нет) 2.6. Используя метод GetType() класса Student и метод BaseType() класса Type, вывести всех предков класса Student (написать общий метод)

**Замечание 1.** Свойства ("умные" поля) определяются так:

private int age;

public int Age {

get { return age; }

set { if (value<0) value = 0; age = value; }

}

Здесь value - переменная, неявно объявленная в каждом сеттере. Свойства отличаются от полей тем, что при доступе на чтение и запись можно совершать дополнительные действия. Обычная практика - проверка в сеттере значения на допустимость и его исправление или генерация исключения.

**Замечание 2.** В конструкторе потомка следует вызывать конструктор предка в списке инициализации:

Student(...): base(...) {}

**Замечание 3.** Виртуальные функции следует объявлять с ключевым словом virtual в предке и с ключевым словом override в потомках. Виртуальную функцию следует вызывать через переменную базового класса:

Person p = new Student(...);

p.Print();

**Замечание 4.** Функции ToString() и Equals() определены в базовом классе Object как виртуальные.

**Замечание 5.** p is Student возвращает True если в p - студент или производный класс. p as Student преобразует тип p к Student, а если это невозможно, возвращает null.

**Замечание 6.** Для сравнения на точное совпадение типа используется GetType:

if (p.GetType()==typeof(Student))

**Задание 3. Перегрузка операторов**

3.0. Для классов Person-Student-Teacher реализовать ==, !=

3.1. Создать структуру Complex с перегруженными операциями, а также с возможностью приведения типа double->complex. Должны быть реализованы также ToString(), Equals(), ==, !=. Сравнить производительность в случае реализации Complex как класса и как структуры.

3.2. Создать класс Frac с перегруженными операциями + - \* / , а также с возможностью приведения типа Frac->double. Должны быть реализованы также ToString(), Equals(), ==, !=. Вычислить значение полинома в точке. Все коэффициенты и x должны иметь тип Frac. Сравнить производительность в случае реализации Frac как класса и как структуры.

3.3. Используя класс Frac, реализовать метод Гаусса для решения системы линейных уравнений из n уравнений с n неизвестными и с рациональными коэффициентами, заданными типом Frac. Найти точное решение, представляющее собой List<Frac>.

**Замечание 1.** Операции реализуются как статические методы:

public static bool operator==(Person p1, Person p2) {return p1.Equals(p2);}

**Замечание 2.** Операции приведения типа определяются так:

public static explicit operator double(Frac f) {...}

explicit означает явное приведение типа, вместо него может стоять implicit - неявное приведение типа.

**Задание 4. Индексаторы**

4.1. Создать класс, реализующий битовый массив на основе обычного, используя индексные свойства.

4.2. Создать класс ассоциативного массива, используя два списка List - список ключей и список значений. Основная операция: x = d[K] (на чтение) и d[K] = V (на запись)

Индексные свойства создаются следующим образом:

private Dictionary<string,int> a;

public int this[string s] {

get { return a[s]; }

set { a[s] = value; }

}

**Задание 5 на интерфейсы**

5.1. Проверить, как работает явная и неявная реализация методов интерфейса. Для этого **явно** реализовать в классах Person и Student интерфейс

interface IPrintable {

void Print();

}

и вызвать этот метод, используя переменную типа интерфейс. Заметим, что старый метод Print также можно оставить.

5.2. В классе Student реализовать интерфейс IComparable<Student>. Воспользовавшись Array.Sort, отсортировать массив студентов.

5.3. Реализовать интерфейс IComparer<Student> в классе StudentComparer, вложенном в Student. Параметром его конструктора должен выступать критерий сортировки. Реализовать в классе Student несколько статических свойств по количеству критериев сортировки (например, Student.SortByGroup). Воспользовавшись второй формой Array.Sort, отсортировать массив студентов по разным критериям.

5.4. Реализовать в классах Person-Student-Teacher интерфейс ICloneable и проиллюстрировать его использование.

5.5. Реализовать в классе Person интерфейс IDisposable и убедиться в корректности его работы в операторе using

Интерфейс IDisposable имеет вид:

interface IDisposable {

void Dispose();

}

Он используется для детерминированного освобождения ресурсов в стиле C++, метод Dispose играет роль деструктора. Если класс My поддерживает интерфейс IDisposable, то его можно использовать в операторе using:

using (My m = new My())

{

} *// в конце гарантированно вызовется Dispose*

В методе Dispose() класса Person достаточно выдавать диагностическое сообщение о том, что метод Dispose() вызван.

5.6. Создать контейнер Persons, который можно было бы использовать в foreach. Для этого поместить в него поле List<Person> list и метод Add(Person p), а также реализовать интерфейс IEnumerable<Person>, используя в методе GetEnumerator конструкцию yield return.

Пример реализации:

public class CustomContainer: IEnumerable<int>

{

public IEnumerator<int> GetEnumerator()

{

for (int i = 0; i < 10; i++)

yield return i;

}

}

Заполнить контейнер Persons персонами и студентами, используя его метод Add. После этого воспользоваться методом foreach по данному контейнеру, выдавая всех персон и студентов в контейнере. Перед каждой персоной или студентом должен выводиться порядковый номер во внутреннем списке.

(нет) 5.7. Реализовать обобщенную функцию MinIndex, возвращающую пару (индекс, МинимальныйЭлемент) в массиве элементов типа T. Для этого наложить в секции where ограничений на параметры обобщения условие where T: IComparable<T>. Результат возвращать в виде Tuple<int,T>. Если в массиве нет элементов, то индекс минимального равен -1, а минимальный равен default(T) - значению по умолчанию для типа T.

**Задание 6 на yield**

6.1. Используя yield, реализовать и оттестировать метод-генератор n членов арифметической прогрессии с первым элементом a0 и шагом d. Для вывода реализовать метод Print, использующий yield и выводящий любую последовательность.

6.2. Реализовать метод-генератор бесконечной последовательности значений val. Для тестирования реализовать метод

IEnumerable<T> TakeN<T>(IEnumerable<T> seq, int n)

который возвращает n первых членов бесконечной последовательности.

6.3. Реализовать метод-генератор бесконечной последовательности случайных чисел.

6.4. Реализовать метод-генератор бесконечной арифметической прогрессии.

6.4а. Реализовать метод-генератор бесконечной последовательности чисел Фибоначчи.

6.5. Реализовать метод-генератор бесконечной последовательности циклически повторяющихся элементов последовательности seq

6.6. Реализовать метод, принимающий две последовательности одинаковой длины и возвращающий последовательность пар в виде Tuple<T,T1>

6.7. Реализовать метод, возвращающий по последовательности целых отфильтрованную последовательность, состоящую только из четных элементов исходной последовательности

6.8. Реализовать метод, возвращающий по последовательности целых значений преобразованную последовательность квадратов этих значений (последовательность, полученную произвольной функцией преобразования).

6.9. Реализовать метод, принимающий две последовательности элементов одного типа и возвращающий последовательность чередующихся элементов исходных последовательностей

6.10. Реализовать метод, возвращающий по последовательности значений последовательность пар рядом стоящих значений:

1 2 3 4 5 -> (1,2) (2,3) (3,4) (4,5)

**(нет) Задание 7 на файлы и потоки**

7.1. Сохранить в текстовых файлах русско-английский тест в различных кодировках и затем считать его из этого файла:

а) в однобайтовых: MS DOS, Koi-8, Windows

б) в Unicode: Utf-8, Utf-16

7.2. В текстовом файле записаны вещественные числа (на каждой строчке - несколько, разделены несколькими пробелами, в формате 3.14 2.597) и другие лексемы (не числа). Найти сумму чисел, игнорируя неверные лексемы.

7.3. Создать типизированный файл целых, затем модифицировать его, возведя все элементы в квадрат.

7.4. Для данной папки рекурсивно выдать список её файлов и подпапок.

**(нет) Задание 8 на сериализацию**

Для выполнения заданий следует подключить к основной сборке следующие сборки: System.Runtime.Serialization System.Runtime.Serialization.Formatters.Soap

8.1. Используя BinaryFormatter и атрибут [Serializable], сериализовать на диск в бинарном формате объекты классов Student и Teacher, после чего десериализовать. Убедиться, что сериализация работает корректно (для этого десериализовать и вывести на экран повторно). Обратить особое внимание, что класс Teacher содержит поле List<Student>. Попробовать провести сериализацию, используя SoapFormatter. Что не работает? Как это можно исправить (какое поле убрать/не сериализовать)?

8.2. Используя SoapFormatter и атрибут [Serializable], сериализовать на диск в XML формате связный список из нескольких

class Node

{

public int data;

public Node next;

// конструктор

}

Затем десериализовать его и убедиться, что все данные сохранились. Посмотреть содержимое XML-файла и понять, за счет чего десериализуется связный список (как восстанавливаются ссылки). Можно ли в этом случае использовать BinaryFormatter?

8.3. Используя класс XMLSerializer, сериализовать-десериализовать объекты класса Student в XML-формате. Использовать атрибуты [XmlAttribute] для полей, которые необходимо сериализовать. Чем отличается XML-представление от SoapFormatter? Понять ограничения этого сериализатора.

8,4. Используя SoapFormatter, создать структуру, поддерживающую граф объектов и сериализовать-десериализовать граф. Подумать над представлением графа в виде списка инцидентных вершин, не используя List<T>.

8.5. Используя DataContractSerializer, его методы ReadObject и WriteObject и атрибуты [DataContract] для класса и [DataMember] для открытых полей или свойств, сериализовать на диск в XML-формате объекты класса Student. Вызвав конструктор DataContractSerializer в виде new DataContractSerializer (typeof (Student), new Type[ ]{typeof (SeniorStudent)}), сериализовать-десериализовать также потомков Student типа SeniorStudent.

#### Задание 13 на рефлексию

13.1. Используя механизм отражения, "расшифровать" все классы, содержащиеся в Student.dll. Создать объект класса Student, вызвать его метод, свойство (на чтение и запись). dll не подключать к проекту статически, использовать Assembly.LoadFrom().

13.2. Для данного типа (Student) вывести цепочку всех его предков и все интерфейсы, им реализуемые.

13.3. Используя механизм отражения, динамически создать объект класса Dictionary<string,int>, динамически наполнить его несколькими значениями и динамически вызвать метод (свойство), возвращающее значение по ключу.

13.4. Создать программу автоподключения плагинов к программе. Плагин представляет собой dll, содержащую класс, удовлетворяющий некоторому интерфейсу с одним методом. Все плагины бросаются в папку с основной программой. Программа должна анализировать все классы в dll в текущей папке, отбирать те, которые реализуют интерфейс, создавать по одному объекту каждого такого класса и вызывать метод интерфейса для этого объекта.

**Задание 15 на потоки (Threads)**

[Материалы по параллельному программированию в .NET на MSDN](http://msdn.microsoft.com/ru-ru/library/dd460693%28v=vs.110%29.aspx)

15.1. Создать несколько потоков, выводящих на консоль в цикле свой hashcode. Посмотреть порядок вывода. Добавить Sleep(1), Sleep(10), Sleep(0). Посмотреть изменения в порядке вывода. Закоментировать Sleep() и установить приоритеты потокам. Посмотреть изменения в порядке вывода.

15.2. Написать программу, демонстрирующую работу t.Join(): основной поток должен дожидаться окончания работы 2-х потоков, потом продолжать работу. Конструктивно: дополнительные потоки должны предоставлять данные, необходимые для дальнейшей работы основного потока.

15.3. Продемонстрировать работу критических секций на примере банкомата.

15.4. Используя критические секции, реализовать потокобезопасный класс Стек. Операции Push и Pop должны блокироваться одним объектом. Проиллюстрировать корректность работы стека, выполняя операции Push и Pop случайным образом в разных потоках. Продемонстрировать, что обычный класс стека не является потокобезопасным.

15.5. Продемонстрировать ситуацию с возникновением DeadLock.

15.6. Продемонстрировать асинхронную работу методов с использованием пула потоков (Task.Run). Операции для пула потоков должны быть достаточно длительными.

**Задание 16 на асинхронное программирование**

16.1. Продемонстрировать работу async и await

16.2а) Продемонстрировать работу именованного мьютекса для синхронизации действий между несколькими процессами.

16.2б) (нет) Продемонстрировать работу именованного семафора для синхронизации действий внутри одного процесса.

16.2в) Продемонстрировать работу ManualResetEvent в ситуации когда несколько потоков дожидаются возникновения события.

16.2г) Продемонстрировать работу AutoResetEvent в ситуации когда несколько потоков работают, попеременно посылая друг другу команды на продолжение.

16.3. Продемонстрировать выполнение асинхронных операций, используя:

а) модель ожидания

б) модель опроса

в) модель с обратным вызовом

г) модель с продолжением (continuation)