Problem Set 2: Introduction to C#, Indexer & Properties

Problem 1
Build a class library “Person.dll” that implements the class **Person** that has the following fields:

- string firstName;
- string lastName;
- string middleName;
- string sSN;
- DateTime birthDate;
- string profession;

Note, in the new class **Person** you must override some of its inherited methods!

Problem 2
Build a class library “Stack.dll” that implements a generic container type **Stack**, which satisfies the following interface:

```csharp
public class Stack
{
    public Stack() { ... }
    public Boolean IsEmpty() { ... }
    public object Peek() { ... }
    public object Pop() { ... }
    public void Push( Object aObj ) { ... }
    public int Search( Object aObj ) { ... }
    public object this [int aIndex] { get; }  
}
```

This class implements a last-in-first-out stack of objects. The method Push puts an object on the top of the stack. The method Pop removes and returns the top object from the stack. The method Peek returns the top object without removing it. Search returns the stack index of an object if this object is an element of the stack, otherwise this function returns -1. Finally, the indexer returns the object stored at position aIndex in the Stack object. If aIndex does not denote a valid stack index, then the indexer throws the exception IndexOutOfRangeException.
Problem 3

Build a class library “MatrixLib.dll” that defines the namespace “MatrixLib”. This class library should provide two types “Vector” and “Matrix”. Both types should be implemented in different files. The type “Vector” should provide the following methods and constructors:

```csharp
namespace MatrixLib
{
    public class Vector
    {
        // creates a vector of size ASize, all elements are set to 0
        public Vector(int ASize) {...}
        // creates a vector using an array of integers
        public Vector(params int[] vals) {...}
        // returns a clone of the current Vector object
        public Vector Clone() {...}
        // Vector indexer, may raise IndexOutOfRangeException
        public int this[int index] { get {...} set {...} }
        // The readonly property Size denotes the actual size of the vector.
        public int Size { get {...} }
        // The function BuildDotProduct returns the \( \sum_{i} V_i \times W_i \), may raise ArithmeticException
        public int BuildDotProduct(Vector AVector) {...}
        // returns a string representation of a Vector object
        public override string ToString() {...}
    }
}
```

The type “Matrix” should provide the following methods and constructors:

```csharp
namespace MatrixLib
{
    public class Matrix
    {
        // creates a m,n matrix, all elements are set to 0
        public Matrix(int ARows, int AColumns) {...}
        // creates a matrix using initialized vectors of the same size
        public Matrix(params Vector[] vals) {...}
        // returns a clone of the current Matrix object
        public Matrix Clone() {...}
        // Matrix indexer, may raise IndexOutOfRangeException
        public int this[int ARow, int AColumn] { get {...} set {...} }
        // number of columns
        public int Columns { get {...} }
        // number of rows
        public int Rows { get {...} }
        // returns the product of this Matrix object and the aVector object with ResultVector[i] = \( \sum_{j} m_{ij}v_j \), may raise ArithmeticException
        public Vector MatrixTimesVector(Vector aVector) {...}
        // returns a string representation of a Matrix object
        public override string ToString() {...}
    }
}
```
Both the type Vector and the type Matrix provide at least one type-specific operation (e.g., dot product and multiplication of a matrix with a vector). A precise description of them can be found in any book on matrix algebra.

Define validation checks where it is necessary. Some operations (i.e., member functions) require specific input arguments! Use the standard exceptions as indicated above to signal that an operation has encountered an unexpected value.

Define two different constructors for both types. One constructor uses the dimensions of the object to be created, while the other one uses an array of values. This array has a variable length. However, we assume that when you create a Matrix object all vectors have the same size.

**Problem 4**

Implement the console application “TestMatrix.exe” that uses the class library “MatrixLib.dll”. In the Main function, define the necessary values (i.e., vectors and matrices) and write the appropriate code such that the console application “TestMatrix.exe” produces the following output:

```
First vector: (1.2,3.3)
Second vector: (1.2,3.4)
Dot product: 14

<1.2,3.3>*(4.5,6.2)<2.8,9.9>

<1.2,3.3>*(4.5,6.2)<2.8,9.9> = (8.4,58) = (26.62,98)
```

**Submission deadline:** Friday, September 23, 2005, 4:10 p.m.

**Submission procedure:** on paper in class (.cs files only) and electronically using the turnin-hw2 script, which is located in ~cs430x/public/bin. Please use the printout of the submission confirmation email as cover page and check the problems that you have solved.

In order to submit your homework solutions, go (using your CS UNIX account) into the directory that contains your solution (i.e., C#-source files and all related project files). In that directory run the command “~cs430x/public/bin/turnin-hw2“. After a successful submission, your will receive a confirmation email. Before the due date, you can resubmit your solutions as often as you like.

On the department’s Windows XP systems you can use the command csc to compile C#-programs. However, it is recommended to use Visual Studio .NET, because most assignments require some GUI work.