Verificarea şi Testarea Sistemelor de Calcul

Temă de casă

November 19, 2018

Titlu: *Testarea Empirică*

Profesor: Ş.l. Dr. Ing. Nicolae Enescu

Student: Voiculescu Ioan-Valentin

Facultate: Automatică, Calculatoare şi Electronică

Anul: IV

Specializarea: Calculatoare Română

Grupa: CR 4.H1 A

Contents

1 Codul Sursă[1]

1.1 Implementarea

1.1.1 EmpiricalTestingConsole.cs

```
  using   System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Lab1
{
    public static class EmpiricalTestingConsole
    {
        private static Output _output;
        static EmpiricalTestingConsole()
        {
            _output = new Output();
        }
        public static void Write(string[] lines)
        {
            _output.WriteData(lines);
        }
    }
}
```

1.1.2 Program.cs

```
  using   System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Lab1
{
    class Program
    {
        static void Main(string[] args)
        {
            EmpiricalTestingConsole.Write(new string[1] { "static void Main(string[] args)" });
            try
            {
                EmpiricalTestingConsole.Write(new string[1] { "try" });
                InputProcessing inputProcessing = new InputProcessing(new Input(), new Output());
                EmpiricalTestingConsole.Write(new string[1] { "InputProcessing inputProcessing = new InputProcessing(new Input(), new Output())" });
                EmpiricalTestingConsole.Write(new string[1] { "inputProcessing=" + inputProcessing.ToString() });
                OutputProcessing outputProcessing = new OutputProcessing(new Output());
                EmpiricalTestingConsole.Write(new string[1] { "OutputProcessing outputProcessing = new OutputProcessing(new Output())" });
                EmpiricalTestingConsole.Write(new string[1] { "outputProcessing=" + outputProcessing.ToString() });
                QuadraticEquation quadraticEquation = new QuadraticEquation(inputProcessing.GetData());
                EmpiricalTestingConsole.Write(new string[1] { "QuadraticEquation
                quadraticEquation = new
                QuadraticEquation(inputProcessing.
                GetData())" });
                EmpiricalTestingConsole.Write(new
                string[1] { "quadraticEquation=" +
                quadraticEquation.ToString() });
                outputProcessing.PutData(
                quadraticEquation.
                SolveWithRealSolutions());
                EmpiricalTestingConsole.Write(new
                string[1] { "outputProcessing.
                PutData(quadraticEquation.
                SolveWithRealSolutions())" });
            }
            catch (Exception e)
            {
                EmpiricalTestingConsole.Write(new
                string[1] { "catch (Exception e)"
                });
                Console.WriteLine(e.Message);
                EmpiricalTestingConsole.Write(new
                string[1] { "Console.WriteLine(e.
                Message)" });
            }
        }
    }
}
```

1.1.3 QuadraticEquation.cs

```
 using  System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Lab1
{
    public class QuadraticEquation
    {
        //ax^2 + bx + c = 0
        public Coefficients Coefficients { private set; get; }
        public double Delta
        {
            get
            {
                EmpiricalTestingConsole.Write(new string[2] { "public double Delta", "get" });
                return (double)(Math.Pow(Coefficients.B, 2) - 4 * Coefficients.A * Coefficients.C);
            }
        }
        public QuadraticEquation (Coefficients coefficients)
        {
            EmpiricalTestingConsole.Write(new string[1] { "public QuadraticEquation (Coefficients coefficients)" });
            EmpiricalTestingConsole.Write(new string[1] { "coefficients=" + coefficients.ToString() });
            EmpiricalTestingConsole.Write(new string[3] { "A=" + coefficients.A, "B=" + coefficients.B, "C=" + coefficients.C });
            if (coefficients != null)
            {
                EmpiricalTestingConsole.Write(new string[1] { "if (coefficients != null)" });
                Coefficients = coefficients;
                EmpiricalTestingConsole.Write(new
                   string[1] { "Coefficients =
                   coefficients" });
                EmpiricalTestingConsole.Write(new
                   string[1] { "Coefficients=" +
                   Coefficients.ToString() });
                EmpiricalTestingConsole.Write(new
                   string[3] { "A=" + Coefficients.A,
                   "B=" + Coefficients.B, "C=" +
                   Coefficients.C });
            }
            else
            {
                EmpiricalTestingConsole.Write(new
                   string[1] { "else" });
                throw new Exception("'coefficients'
                   cannot be null");
            }
        }
        public Solution SolveWithRealSolutions()
        {
            EmpiricalTestingConsole.Write(new string
               [1] { "public Solution
               SolveWithRealSolutions()" });
            Solution solution = null;
            EmpiricalTestingConsole.Write(new string
               [1] { "Solution solution = null" });
            if (Delta.Equals(0))
            {
                EmpiricalTestingConsole.Write(new
                   string[1] { "if (Delta.Equals(0))"
                   });
                solution = new Solution(
                    (double)(-Coefficients.B) / (
                       double)(2 * Coefficients.A),
                    (double)(-Coefficients.B) / (
                       double)(2 * Coefficients.A)
                    );
                EmpiricalTestingConsole.Write(new
                   string[4]
                {
                    "solution = new Solution(",
                    "(double)(-Coefficients.B) / (
                       double)(2 * Coefficients.A),",
                    "(double)(-Coefficients.B) / (
                       double)(2 * Coefficients.A)",
                    ")"
                });
                EmpiricalTestingConsole.Write(new string[1] { "solution=" + solution.ToString() });
                EmpiricalTestingConsole.Write(new string[2] { "solution.Root1=" + solution.Root1.ToString(), " solution.Root2=" + solution.Root2.ToString() });
            }
            if (Delta > 0)
            {
                EmpiricalTestingConsole.Write(new string[1] { " if(Delta > 0)" });
                solution = new Solution(
                    (double)(-Coefficients.B + Math.Sqrt(Delta)) / (double)(2 * Coefficients.A),
                    (double)(-Coefficients.B - Math.Sqrt(Delta)) / (double)(2 * Coefficients.A)
                    );
                EmpiricalTestingConsole.Write(new string[4]
                {
                    "solution = new Solution(",
                    "(double)(-Coefficients.B + Math.Sqrt(Delta)) / (double)(2 * Coefficients.A),",
                    "(double)(-Coefficients.B - Math.Sqrt(Delta)) / (double)(2 * Coefficients.A)",
                    ")"
                });
                EmpiricalTestingConsole.Write(new string[1] { "solution=" + solution.ToString() });
                EmpiricalTestingConsole.Write(new string[2] { "solution.Root1=" + solution.Root1.ToString(), " solution.Root2=" + solution.Root2.ToString() });
            }
            if (Delta < 0)
            {
                EmpiricalTestingConsole.Write(new string[1] { " if (Delta < 0)" });
                throw new Exception("the equation do not have a real solution");
            }
            return solution;
        }
    }
}
```

1.1.4 Coefficients.cs

```csharp
  using   System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Lab1
{
    public class Coefficients
    {
        public int A { private set; get; }
        public int B { private set; get; }
        public int C { private set; get; }
        public Coefficients(int coefficientA, int
            coefficientB, int coefficientC)
        {
            EmpiricalTestingConsole.Write(new string
                [1] {"public Coefficients(int
                coefficientA, int coefficientB, int
                coefficientC)"});
            EmpiricalTestingConsole.Write(new string
                [1] { "coefficientA=" + coefficientA.
                ToString() });
            EmpiricalTestingConsole.Write(new string
                [1] { "coefficientB=" + coefficientB.
                ToString() });
            EmpiricalTestingConsole.Write(new string
                [1] { "coefficientB=" + coefficientB.
                ToString() });
            if (coefficientA == 0)
            {
                EmpiricalTestingConsole.Write(new
                    string[1] { "if (coefficientA == 0)
                    " });
                throw new Exception("The 'a'
                    coefficient cannot equal 0");
            }
            A = coefficientA;
            EmpiricalTestingConsole.Write(new string
                [1] { "A = coefficientA" });
            EmpiricalTestingConsole.Write(new string
                [1] { "A=" + A.ToString()});
            B = coefficientB;
            EmpiricalTestingConsole.Write(new string
                [1] { "B = coefficientB" });
            EmpiricalTestingConsole.Write(new string
                [1] { "B=" + B.ToString() });
            C = coefficientC;
            EmpiricalTestingConsole.Write(new string
                [1] { "C = coefficientC" });
            EmpiricalTestingConsole.Write(new string
                [1] { "C=" + C.ToString() });
        }
    }
}
```

1.1.5 Solution.cs

```
 using  System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Lab1
{
    public class Solution
    {
        public double Root1 { private set; get; }
        public double Root2 { private set; get; }
        public Solution(double root1, double root2)
        {
            EmpiricalTestingConsole.Write(new string
                [1] { "public Solution(double root1,
                double root2)" });
            EmpiricalTestingConsole.Write(new string
                [2] { "root1=" + root1.ToString(), "
                root2=" + root2.ToString() });
            Root1 = root1;
            EmpiricalTestingConsole.Write(new string
                [1] { "Root1 = root1" });
            EmpiricalTestingConsole.Write(new string
                [1] { "Root1=" + Root1.ToString() });
            Root2 = root2;
            EmpiricalTestingConsole.Write(new string
                [1] { "Root2 = root2" });
            EmpiricalTestingConsole.Write(new string
                [1] { "Root2=" + Root2.ToString() });
        }
    }
}
```

1.1.6 DecimalRound.cs

```
using System;

namespace Lab1
{
    public static class Decimals
    {
        public static string[] Round(Solution solution, int numberDecimals)
        {
            EmpiricalTestingConsole.Write(new string[1] { "public static string[] Round(Solution solution, int numberDecimals)" });
            EmpiricalTestingConsole.Write(new string[1] { "solution=" + solution.ToString() });
            EmpiricalTestingConsole.Write(new string[2] { "solution.Root1=" + solution.Root1.ToString(), "solution.Root2=" + solution.Root2.ToString() });
            EmpiricalTestingConsole.Write(new string[1] { "numberDecimals=" + numberDecimals.ToString() });
            string[] roots;
            EmpiricalTestingConsole.Write(new string[1] { "string[] roots" });
            roots = new string[2]
            {
                (Math.Truncate(Math.Pow(10, numberDecimals) * solution.Root1) / (double) Math.Pow(10, numberDecimals)).ToString(),
                (Math.Truncate(Math.Pow(10, numberDecimals) * solution.Root2) / (double) Math.Pow(10, numberDecimals)).ToString(),
            };
            EmpiricalTestingConsole.Write(new string[5]
            {
                "roots = new string[2]",
                "{",
                "(Math.Truncate(Math.Pow(10, numberDecimals) * solution.Root1) /
                     (double) Math.Pow(10,
                    numberDecimals)).ToString(),",
                "(Math.Truncate(Math.Pow(10,
                    numberDecimals) * solution.Root2) /
                     (double) Math.Pow(10,
                    numberDecimals)).ToString(),",
                "};"

            });
            EmpiricalTestingConsole.Write(new string
                [2] { "roots[0]=" + roots[0], "roots
                [1]=" + roots[1] });
            return roots;
        }
    }
}
```

1.1.7 Constants.cs

```
  using   System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Lab1
{
    public static class Constants
    {
        public const int CoefficientAMinimumValue =
           -30;
        public const int CoefficientAMaximumValue =
           70;

        public const int CoefficientBMinimumValue =
           -50;
        public const int CoefficientBMaximumValue =
           10;

        public const int CoefficientCMinimumValue = 0;
        public const int CoefficientCMaximumValue =
           200;
    }
}
```

1.1.8 InputProcessing.cs

```
  using   System;
using System.Collections.Generic;
using System.Linq;
using System.Linq.Expressions;
using System.Text;
using System.Threading.Tasks;

namespace Lab1
{
    public class InputProcessing
    {
        private Input _input;
        private Output _output;
        public InputProcessing(Input input, Output
            output)
        {
            EmpiricalTestingConsole.Write(new string
                [1] { "public InputProcessing(Input
                input, Output output)" });
            EmpiricalTestingConsole.Write(new string
                [2] { "input=" + input.ToString(), "
                output=" + output.ToString() });
            _input = input;
            EmpiricalTestingConsole.Write(new string
                [1] { " _input = input" });
            EmpiricalTestingConsole.Write(new string
                [1] { "_input=" + _input.ToString() });
            _output = output;
            EmpiricalTestingConsole.Write(new string
                [1] { "_output = output" });
            EmpiricalTestingConsole.Write(new string
                [1] { "_output=" + _output.ToString()
                });
        }
        public Coefficients GetData()
        {
            EmpiricalTestingConsole.Write(new string
                [1] { "public Coefficients GetData()"
                });
            Coefficients coefficients = new
                Coefficients(
                 SetCoefficient('a', Constants.
                    CoefficientAMinimumValue, Constants
                    .CoefficientAMaximumValue),
                 SetCoefficient('b', Constants.
                    CoefficientBMinimumValue, Constants
                    .CoefficientBMaximumValue),
                 SetCoefficient('c', Constants.
                    CoefficientCMinimumValue, Constants
                    .CoefficientCMaximumValue)
                 );
            EmpiricalTestingConsole.Write(new string
                [5]
            {
                 "Coefficients coefficients = new
                    Coefficients(",
                 "SetCoefficient('a', Constants.
                    CoefficientAMinimumValue, Constants
                    .CoefficientAMaximumValue)",
                 "SetCoefficient('b', Constants.
                    CoefficientBMinimumValue, Constants
                    .CoefficientBMaximumValue)",
                 "SetCoefficient('c', Constants.
                    CoefficientCMinimumValue, Constants
                    .CoefficientCMaximumValue)",
                 ")"
            });
            EmpiricalTestingConsole.Write(new string
                [1] { "coefficients=" + coefficients.
                ToString() });
            EmpiricalTestingConsole.Write(new string
                [3]
            {
                 "coefficients.A=" + coefficients.A,
                 "coefficients.B=" + coefficients.B,
                 "coefficients.C=" + coefficients.C
            });
            return coefficients;
        }
        private int SetCoefficient(char c, int
            minimumValue, int maximumValue)
        {
            EmpiricalTestingConsole.Write(new string
                [1] { "private int SetCoefficient(char
                c, int minimumValue, int maximumValue)"
                 });
            EmpiricalTestingConsole.Write(new string
                [3]
            {
                 "c=" + c,
                 "minimumValue=" + minimumValue.
                    ToString(),
                 "maximumValue=" + maximumValue.
                    ToString()
            });
            int coefficient;
            EmpiricalTestingConsole.Write(new string
                [1] { "int coefficient" });
            if (_output != null)
            {
                 EmpiricalTestingConsole.Write(new
                    string[1] { "if (_output != null)"
                    });
                 string line = c + "=";
                 EmpiricalTestingConsole.Write(new
                    string[1] { "string line = c + " +
                    "=" });
                 EmpiricalTestingConsole.Write(new
                    string[1] { "line=" + line });
                 _output.WriteData(new string[1] {line
                    });
                 EmpiricalTestingConsole.Write(new
                    string[1] { "_output.WriteData(new
                    string[1] {line})" });
            }
            try
            {
                 EmpiricalTestingConsole.Write(new
                    string[1] { "try" });
                 coefficient = Int32.Parse(_input.
                    ReadDataByLine());
                 EmpiricalTestingConsole.Write(new
                    string[1] { "coefficient = Int32.
                    Parse(_input.ReadDataByLine())" });
                 EmpiricalTestingConsole.Write(new
                    string[1] { "coefficient=" +
                    coefficient });
            }
            catch (FormatException)
            {
                 EmpiricalTestingConsole.Write(new
                    string[1] { "catch (FormatException
                    )" });
                 throw new Exception(c+" is not integer
                    ");
            }
            if(!(coefficient>=minimumValue &&
                coefficient<=maximumValue))
            {
                EmpiricalTestingConsole.Write(new
                    string[1] { "if(!(coefficient>=
                    minimumValue && coefficient<=
                    maximumValue))" });
                throw new Exception(c + " out of range
                    ");
            }
            return coefficient;
        }
    }
}
```

1.1.9 OutputProcessing.cs

```
  using   System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Lab1
{
    public class OutputProcessing
    {
        private Output _output;
        public OutputProcessing(Output output)
        {
            EmpiricalTestingConsole.Write(new string[1] { "public OutputProcessing(Output output)" });
            _output = output;
            EmpiricalTestingConsole.Write(new string[1] { "_output = output" });
            EmpiricalTestingConsole.Write(new string[1] { "_output=" + output.ToString() });
        }
        public void PutData(Solution solution)
        {
            EmpiricalTestingConsole.Write(new string[1] { "public void PutData(Solution solution)" });
            EmpiricalTestingConsole.Write(new string[1] { "solution=" + solution.ToString() });
            EmpiricalTestingConsole.Write(new string[2] { "solution.Root1=" + solution.Root1.ToString(), "solution.Root2=" + solution.Root2.ToString() });
            string line1, line2;
            EmpiricalTestingConsole.Write(new string[1] { "string line1, line2" });
            string[] newSolution = Decimals.Round(solution, 2);
            EmpiricalTestingConsole.Write(new string[1] { "string[] newSolution = Decimals.Round(solution, 2)" });
            EmpiricalTestingConsole.Write(new string
                [2] { "newSolution[0]=" + newSolution
                [0], "newSolution[1]=" + newSolution[1]
                 });
            line1 = "x1=" + newSolution[0];
            EmpiricalTestingConsole.Write(new string
                [1] { "line1 = "+"x1 = "+" +
                newSolution[0]" });
            EmpiricalTestingConsole.Write(new string
                [1] { "line1=" + line1 });
            line2 = "x2=" + newSolution[1];
            EmpiricalTestingConsole.Write(new string
                [1] { "line2 = "+"x2 = "+" +
                newSolution[1]" });
            EmpiricalTestingConsole.Write(new string
                [1] { "line2=" + line2 });
            _output.WriteData(new string[2] {line1,
                line2});
            EmpiricalTestingConsole.Write(new string
                [1] { "_output.WriteData(new string[2]
                {line1, line2})" });
        }
    }
}
```

1.1.10 Input.cs

```csharp
using System;

namespace Lab1
{
    public class Input
    {
        public virtual string ReadDataByLine()
        {
            EmpiricalTestingConsole.Write(new string
                [1] { "public virtual string
                ReadDataByLine()"});
            return Console.ReadLine();
        }
    }
}
```

1.1.11 Output.cs

```csharp
using System;

namespace Lab1
{
    public class Output
    {
        public virtual void WriteData(string[] lines)
        {
            foreach (var line in lines)
            {
                Console.WriteLine(line);
            }
        }
    }
}
```

References

[1] `https://github.com/vioan12/Verificarea-si-Testarea-Sistemelor-de-Calcul`