Национальный исследовательский университет «МИЭТ»
Центр дистанционного обучения

Направление: «Программная инженерия»

# «Программирование на C#»

Отчет по лабораторным работам

Вариант 1

Выполнила: Смирнова М.М
ПИН-31Д

https://github.com/MariaLink24/csharplabs/

# Лабораторная работа 1
# Классы, свойства, индексаторы. Одномерные, прямоугольные и ступенчатые массивы

**Person.cs**

```
using System;
class Person
   {
   private string firstName;
   private string lastName;
   private DateTime birthDate;

   public Person (){
    firstName = "Sharik";
    lastName = "Matroskin";
    birthDate = new DateTime(1980, 2, 3);
   }

   public Person (string fn, string ln, DateTime bd){
      firstName = fn;
      lastName = ln;
      birthDate = bd;
   }

   public String first
```

```
        {
            get { return firstName; }
            set { firstName = value; }
        }
public String last
    {
        get { return lastName; }
        set { lastName = value; }
    }

public DateTime date
    {
        get { return birthDate; }
        set { birthDate = value; }
    }
public int birthYear
    {
        get { return birthDate.Year; }
        set { birthDate = new DateTime(value, birthDate.Month, birthDate.Day); }
    }
public override string ToString()
    {
        return firstName + " " + lastName + " " + birthDate.ToString() + "\n";
    }
public virtual string ToShortString()
    {
        return  firstName + " " +  lastName + "\n";
    }
```

```
    }
```

**Student.cs**

```
using System;
class Student{
        private Person person;
        private Education education;
        int group;
        Exam[] examArr;

        public Person student {
              get { return person; }
            set { person = value; }
        }
         public Education educationType {
              get { return education; }
            set { education = value; }
        }

        public int groupNumber {
              get { return group; }
            set { group = value; }
        }

        public Exam[] exams {
              get { return examArr; }
            set { examArr = value; }
        }
```

```
    public double meanValue {
        get {
            double mean = 0;
            if (examArr != null){
              for (int i=0; i<examArr.Length; i++){
                    mean+=examArr[i].grade;
                }
              mean /= (double)examArr.Length;
            }
           return mean;
    }
     }
    public bool this[Education index] {
        get  {return education == index; }
    }

public Student(Person p, Education e, int g){
    person = p;
    education = e;
    group = g;
}
 public Student(){
    person = new Person();
    education = Education.Bachelor;
    group = 1;
}
```

```
    public void AddExams(Exam[] exms){
        if (examArr != null)
            for (int i=0; i<exms.Length; i++){
                examArr = exms;
            }
        else examArr = exms;

    }
    public override string ToString()
        {
            string exams = "";
            if (examArr.Length!=0)
                foreach(Exam e in examArr){
                    exams += e.ToString() + " ";
                }

            return person.first +  " " + person.last + " "  + person.birthYear.ToString() + " "
            + educationType.ToString() + " " + group.ToString() + " " + exams + "\n";
        }
    public virtual string ToShortString()
        {
            return person.first +  " " + person.last + " "  + person.birthYear.ToString() + " " +
educationType.ToString() + " " + group.ToString() + " " + meanValue.ToString() + "\n";
        }

}
```

**Exam.cs**

```
using System;
```

```
class Exam {
        public string subject;
        public int grade;
        public DateTime examDate;

     public Exam(string s, int g, DateTime d){
         subject = s;
         grade = g;
         examDate = d;
     }
      public Exam(){
         subject = "C# practice";
         grade = 5;
         examDate = new DateTime(2021, 6, 1);
     }
        public override string ToString()
        {
            return subject + " " + grade + " " + examDate.ToString() + "\n";
        }

    }
```

**Program.cs**

```
using System;
using System.Diagnostics;

public  enum Education {Specialist, Bachelor, SecondEducation}

class Program{
```

```
    static void Main(string[] args){
        Student rand_student = new Student();
        Console.WriteLine("данные студента: " + rand_student.ToShortString());

        Console.WriteLine(rand_student[Education.Bachelor]);
        Console.WriteLine(rand_student[Education.SecondEducation]);
        Console.WriteLine(rand_student[Education.Specialist]);

        Person justPerson = new Person("Kot", "Matroskin", new DateTime(1999, 1, 1));
        Student new_rand_student = new Student(justPerson, Education.Specialist, 2);
        Console.WriteLine("данные второго студента: " + new_rand_student.ToShortString());

        Exam cs = new Exam();
        Exam math = new Exam("math", 4, new DateTime(2021-5-6));
        Exam physics = new Exam("physics", 4, new DateTime(2021-5-7));
        Exam[] exams = new Exam[3];
        exams[0]= cs;
        exams[1]= math;
        exams[2]= physics;
        new_rand_student.AddExams(exams);
        Console.WriteLine("данные второго студента  с экзаменами: " +
new_rand_student.ToString());
        int n = 4;
        int m = 4;
        Exam[] exms1 = new Exam[n*m];
        for (int i = 0; i < n * m;i++)
              exms1[i] = new Exam();
        Exam[,] exms2 = new Exam[n,m];
```

```
for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            exms2[i,j] = new Exam();
    int sum = 0;
    int k = 0;
    for (; sum < 4 * 4; k++)
        sum += k;
Exam[][] exms3 = new Exam[n][];
for (int i = 0; i < n; i++)
    {
        exms3[i] = new Exam[i];
        for (int j = 0; j < i; j++)
            exms3[i][j] = new Exam();
    }
Stopwatch timer = new Stopwatch();


  timer.Start();
    for(int i=0;i<n*m;i++)
    {
        exms1[i].subject = "russian";
        exms1[i].grade = 5;
        exms1[i].examDate = new DateTime();
    }
    timer.Stop();
    TimeSpan timeTaken = timer.Elapsed;
    string time1 = timeTaken.ToString(@"m\:ss\.fff");
    timer.Start();
    for(int i=0;i<n;i++)
```

```
            for(int j=0;j<m;j++)
            {
                exms2[i, j].subject = "russian";
                exms2[i, j].grade = 5;
                exms2[i, j].examDate = new DateTime();
            }
        timer.Stop();
        TimeSpan timeTaken2 = timer.Elapsed;
        string time2 = timeTaken2.ToString(@"m\:ss\.fff");
        timer.Start();
        for(int i=0;i< n;i++)
            for(int j=0;j< i;j++)
            {
                exms3[i][j].subject = "russian";
                exms3[i][j].grade = 5;
                exms3[i][j].examDate = new DateTime();
            }
        TimeSpan timeTaken3 = timer.Elapsed;
        string time3 = timeTaken3.ToString(@"m\:ss\.fff");
        timer.Stop();
        Console.WriteLine("одномерный: " + time1+"\n");
        Console.WriteLine("двумерный прямоугольный: " + time2 + "\n");
        Console.WriteLine("двумерный ступенчатый: " + time3 + "\n");

    }
}
```

# Лабораторная работа 2
# Наследование. Исключения. Интерфейсы. Итераторы и блоки итераторов

**Person.cs**

```
using System;

interface IDateAndCopy {
    object DeepCopy();
    DateTime Date {get; set;}
}

class Person: IDateAndCopy
    {
    protected string firstName;
    protected string lastName;
    protected DateTime birthDate;


    public Person (){
     firstName = "Sharik";
     lastName = "Matroskin";
     birthDate = new DateTime(1980, 2, 3);
    }


    public Person (string fn, string ln, DateTime bd){
        firstName = fn;
        lastName = ln;
        birthDate = bd;
```

```
}

DateTime IDateAndCopy.Date { get; set; }

public String first
      {
         get { return firstName; }
         set { firstName = value; }
      }
public String last
   {
      get { return lastName; }
      set { lastName = value; }
   }

public DateTime date
   {
      get { return birthDate; }
      set { birthDate = value; }
   }

public int birthYear
   {
      get { return birthDate.Year; }
      set { birthDate = new DateTime(value, birthDate.Month, birthDate.Day); }
   }
```

```
public override string ToString()
   {
      return firstName + " " + lastName + " " + birthDate.ToString() + "\n";
   }
public virtual string ToShortString()
   {
      return  firstName + " " +  lastName + "\n";
   }


public override  bool Equals(object obj){
     Person personObject = obj as Person;


     return obj != null &&
     this.firstName == personObject.firstName &&
        this.lastName == personObject.lastName &&
        this.birthDate == personObject.birthDate;
   }
   public static bool operator ==(Person p1, Person p2)
   {
      return p1.Equals(p2);
   }
   public static bool operator !=(Person p1, Person p2)
   {
      return !p1.Equals(p2);
   }
   public override int GetHashCode()
   {
```

```
        return this.firstName.GetHashCode()+this.lastName.GetHashCode()
+this.birthDate.GetHashCode();

    }

    public virtual object DeepCopy()

    {

        Person copiedPerson = new Person();

        copiedPerson.lastName = (string)lastName.Clone();

        copiedPerson.firstName = (string)firstName.Clone();

        copiedPerson.birthDate = birthDate;

        return copiedPerson;

    }


}
```

**Student.cs**

```
using System;

using System.Collections;

class Student : Person, IDateAndCopy, IEnumerable{


    private Person person;

    private Education education;

    private int group;

    private ArrayList examArr;

    private ArrayList testArr;



     public Education educationType {

        get { return education; }

        set { education = value; }
```

```
}



public int groupNumber {
      get { return group; }
    set {
         if (value <= 100 || value > 599)
        {
            throw new ArgumentOutOfRangeException("err: boundaries exceeded [100; 599)");
        }
        groupNumber = value;
        }
}

public System.Collections.ArrayList exams {
      get { return  (System.Collections.ArrayList)examArr; }
    set { examArr = (System.Collections.ArrayList)value; }
}
 public System.Collections.ArrayList tests {
      get { return  (System.Collections.ArrayList)testArr; }
    set { testArr = (System.Collections.ArrayList)value; }
}
public Person p {
     get
    {
        return new Person((string)firstName.Clone(), (string)lastName.Clone(), birthDate);
    }
    set
```

```
    {
        firstName = (string)value.first.Clone();
        lastName = (string)value.last.Clone();
        birthDate = value.date;
    }
  }

  public double meanValue {
      get {
         double mean = 0;
         if (examArr != null){
           foreach(Exam e in examArr){
                mean+=e.grade;
              }
           mean /= (double)examArr.Count;
         }
        return mean;
  }
  }
  public bool this[Education index] {
      get  {return education == index; }
  }

public Student(Person p, Education e, int g){
  person = p;
  education = e;
  group = g;
  examArr = new System.Collections.ArrayList();
```

```
    testArr = new System.Collections.ArrayList();
}
 public Student(){
    Person person = new Person();
    education = Education.Bachelor;
    group = 1;
    examArr = new System.Collections.ArrayList();
    testArr = new System.Collections.ArrayList();

}

public void AddExams(ArrayList exms){
       if(exms != null){
          foreach (Exam e in exms){
            examArr.Add(e.DeepCopy());

          }
       }
   }

   public void AddTests(ArrayList tsts){
       if (tsts != null) {
          foreach (Test t in tsts){
             testArr.Add(t.DeepCopy());
          }
       }
   }
   public override string ToString()
```

```
        {
            string result;
            result = person.ToString() + " "  +
              educationType.ToString() +
             " " + group.ToString() + " ";

          if (examArr!=null) {
            foreach(Exam e in examArr)
            result = result + e.ToString() + " ";
          }
           if (testArr!=null) {
            foreach(Test t in testArr)
            result = result + t.ToString() + " ";
           }
            return result;

           }
      public override string ToShortString()
        {
            return person.first + " " + person.last+ " " + person.date.ToString() + " "
+educationType.ToString() + " " + group.ToString() + " " + meanValue.ToString() + "\n";
        }
        public override object DeepCopy()
      {
        Student copiedStudent = new Student(person, education, group);
        copiedStudent.AddExams(this.examArr);
        copiedStudent.AddTests(this.testArr);
```

```
    return copiedStudent;
}



public IEnumerable getAllExamsAndTest() {
if (examArr!=null){
    foreach (var exam in examArr)
        yield return exam;
}
if (testArr!=null){
    foreach (var test in testArr)
        yield return test;
}
}

public IEnumerable getSuccessfulExams(int k) {
    if(examArr!=null){
    foreach (var exam in examArr) {
        Exam ex = (Exam) exam;
        if (ex.grade > k)
            yield return exam;
    }
    }
}
public IEnumerable getDoneTestsAndExams(){
    if(examArr!=null){
        foreach(var exam in examArr){
            Exam ex = (Exam) exam;
```

```
    if (ex.grade > 2)
        yield return exam;
  }
    }
    if(testArr!=null){
    foreach(var test in testArr){
        Test t = (Test) test;
    if (t.isPassed)
        yield return test;
  }
    }
  }



public IEnumerable getDoneTests(){
   if(examArr!=null){
    foreach(var exam in examArr){
        Exam ex = (Exam) exam;


    if(testArr!=null){
    foreach(var test in testArr){
        Test t = (Test) test;
    if (t.isPassed && t.subject==ex.subject && ex.grade>2)
        yield return test;
  }
    }
}
```

```
        }
    }

     public IEnumerator GetEnumerator()
        {
         return new StudenEnumerator(this);
    }

}
```

**StudentEnumerator.cs**

```
using System;
using System.Collections;
using System.Linq;

class StudenEnumerator : IEnumerator{

Student student;
string currSubj;
int position = -1;
ArrayList  subjs;
public StudenEnumerator(Student s){
    student = s;
}
 public object Current {
    get {
      return currSubj;
    }
  }
```

```
public bool MoveNext(){
    subjs = getSameSubjects();
    if (position == subjs.Count - 1) {
      Reset();
      return false;
    }
    position++;
    return true;
}
public void Reset(){
    position = -1;


}
public ArrayList getExamNames(){
    ArrayList subjs=null;
            for (int i = 0; i < student.exams.Count; i++)
            {
              Exam ex = (Exam) student.exams[i];
              subjs.Add(ex.subject);
            }



            return subjs;
}
public ArrayList getTestsNames(){
    ArrayList subjs=null;
            for (int i = 0; i < student.tests.Count; i++)
            {
```

```
            Test t = (Test) student.tests[i];
            subjs.Add(t.subject);
          }

          return subjs;
}
public ArrayList getSameSubjects(){
   var exams = getExamNames();
   var tests = getTestsNames();
var elements = System.Linq.Enumerable.Intersect(exams.ToArray(), tests.ToArray()).ToArray();
ArrayList result = new ArrayList(elements);
return result;
}



}
```

**Exam.cs**

```
using System;
class Exam : IDateAndCopy {
        public string subject;
        public int grade;
        public DateTime examDate;

   public DateTime Date { get => throw new NotImplementedException(); set => throw new
NotImplementedException(); }

   public Exam(string s, int g, DateTime d){
         subject = s;
```

```
    grade = g;
    examDate = d;
  }
  public Exam(){
    subject = "C# practice";
    grade = 5;
    examDate = new DateTime(2021, 6, 1);
  }
  DateTime IDateAndCopy.Date
    {
       get { return examDate; }
       set { examDate = value; }

    }
    public override string ToString()
    {
       return subject + " " + grade.ToString() + " " + examDate.ToString() + "\n";
    }
    public object DeepCopy()
    {
       Exam copiedExam = new Exam();
       copiedExam.examDate = examDate;
       copiedExam.grade = grade;
       copiedExam.subject = (string)subject.Clone();
       return (object)copiedExam;
    }

}
```

**Program.cs**

```
using System;
using System.Collections;
using System.Diagnostics;

public  enum Education {Specialist, Bachelor, SecondEducation}

class Program{
    static void Main(string[] args){

        Person p1 = new Person();
        Person p2 = new Person();
        // Console.WriteLine("ссылки на p1 p2 равны: " + Object.ReferenceEquals(p1, p2)+"\n");
        // Console.WriteLine("равны объекты p1 p2: " + p1.Equals(p2)+"\n");
        // Console.WriteLine("p1 hash: " + p1.GetHashCode()+"\n");
        // Console.WriteLine("p2 hash: " + p2.GetHashCode()+"\n");

        Student s1 = new Student(p1, Education.Bachelor, 1);
        Console.WriteLine("свойство person для student : " + s1.p +"\n");

        Exam cs = new Exam();
        Exam math = new Exam("math", 4, new DateTime(2021-5-6));
        Exam physics = new Exam("physics", 4, new DateTime(2021-5-7));
        Exam[] exams = new Exam[3];
        exams[0]= cs;
        exams[1]= math;
        exams[2]= physics;
        ArrayList examsArrayList = new ArrayList();
```

```
examsArrayList.AddRange(exams);
s1.AddExams(examsArrayList);

Test t = new Test();
Test[] tests = new Test[1];
tests[0] = t;
ArrayList testsArrayList = new ArrayList();
testsArrayList.AddRange(tests);
s1.AddTests(testsArrayList);
Student s2 = (Student)s1.DeepCopy();
s1.first =  "Kot";
Console.WriteLine("оригинальный student : " + s1.ToString() +"\n");
Console.WriteLine("скопированный student : " + s2.ToString() +"\n");
Console.WriteLine("оригинальный student : " + s1.first +"\n");



try
   {
      s1.groupNumber = 600;
   }
   catch (ArgumentOutOfRangeException e)
   {
      Console.WriteLine(e.Message);
   }

   foreach (var task in s1.getAllExamsAndTest())
      Console.WriteLine(task.ToString());
```

```
            foreach (var task in s1.getSuccessfulExams(3))
                Console.WriteLine(task.ToString());

             foreach (var task in s1.getDoneTestsAndExams())
                Console.WriteLine(task.ToString());

            foreach (var task in s1.getDoneTests())
                Console.WriteLine(task.ToString());




    }
}
```

**Test.cs**

```
using System;

class Test{
    public string subject;
    public bool isPassed;

    public Test(){
        subject = "test subject";
        isPassed = true;
    }
     public Test(string s, bool i){
        subject = s;
```

```
        isPassed = i;
    }
     public override string ToString()
        {
            return subject + " " + isPassed.ToString()  + "\n";
        }
     public object DeepCopy()
        {
            Test copiedTest = new Test();
            copiedTest.subject = (string)subject.Clone();
            copiedTest.isPassed = isPassed;
            return (object)copiedTest;
        }
}
```

# Лабораторная работа 3. Варианты первого уровня Универсальные типы. Классы-коллекции. Методы расширения класса System.Linq.Enumerable

**Person.cs**

```
using System;
using System.Collections.Generic;
using System.Diagnostics.CodeAnalysis;
interface IDateAndCopy {
    object DeepCopy();
    DateTime Date {get; set;}
}
class Person: IDateAndCopy, IComparable, IComparer<Person>
    {
    protected string firstName;
    protected string lastName;
    protected DateTime birthDate;

    public Person (){
     firstName = "Sharik";
     lastName = "Matroskin";
     birthDate = new DateTime(1980, 2, 3);
    }

    public Person (string fn, string ln, DateTime bd){
        firstName = fn;
        lastName = ln;
        birthDate = bd;
```

```
}

DateTime IDateAndCopy.Date { get; set; }

public String first
      {
         get { return firstName; }
         set { firstName = value; }
      }
public String last
   {
      get { return lastName; }
      set { lastName = value; }
   }

public DateTime date
   {
      get { return birthDate; }
      set { birthDate = value; }
   }

public int birthYear
   {
      get { return birthDate.Year; }
      set { birthDate = new DateTime(value, birthDate.Month, birthDate.Day); }
   }
```

```
public override string ToString()
   {
      return firstName + " " + lastName + " " + birthDate.ToString() + "\n";
   }
public virtual string ToShortString()
   {
      return  firstName + " " +  lastName + "\n";
   }


public override  bool Equals(object obj){
     Person personObject = obj as Person;


     return obj != null &&
     this.firstName == personObject.firstName &&
        this.lastName == personObject.lastName &&
        this.birthDate == personObject.birthDate;
   }
   public static bool operator ==(Person p1, Person p2)
   {
      return p1.Equals(p2);
   }
   public static bool operator !=(Person p1, Person p2)
   {
      return !p1.Equals(p2);
   }
   public override int GetHashCode()
   {
```

```
        return this.firstName.GetHashCode()+this.lastName.GetHashCode()
+this.birthDate.GetHashCode();

    }

    public virtual object DeepCopy()

    {

        Person copiedPerson = new Person();

        copiedPerson.lastName = (string)lastName.Clone();

        copiedPerson.firstName = (string)firstName.Clone();

        copiedPerson.birthDate = birthDate;

        return copiedPerson;

    }

     public int CompareTo(object obj)

{

    Person p = obj as Person;

        return this.last.CompareTo(p.last);




}


public int Compare([AllowNull] Person x, [AllowNull] Person y)

{

     if (x.date < y.date)

        return 1;

    else if (x.date > y.date)

        return -1;

    else

        return 0;

}
```

```
    }
```

**Student.cs**

```
using System;
using System.Collections;
using System.Collections.Generic;

class Student : Person, IDateAndCopy, IEnumerable{

        private Person person;
        private Education education;
        private int group;
        private List<Exam> examArr;
        private List<Test> testArr;



         public Education educationType {
              get { return education; }
            set { education = value; }
        }



        public int groupNumber {
              get { return group; }
            set {
                 if (value <= 100 || value > 599)
                {
                    throw new ArgumentOutOfRangeException("err: boundaries exceeded [100; 599)");
```

```
        }
        groupNumber = value;
        }
}

public List<Exam> exams {
      get { return  (List<Exam>)examArr; }
    set { examArr = (List<Exam>)value; }
}
 public List<Test> tests {
      get { return  (List<Test>)testArr; }
    set { testArr = (List<Test>)value; }
}
public Person p {
     get
    {
        return new Person((string)firstName.Clone(), (string)lastName.Clone(), birthDate);
    }
    set
    {
        firstName = (string)value.first.Clone();
        lastName = (string)value.last.Clone();
        birthDate = value.date;
    }
}

 public double meanValue {
      get {
```

```
            double mean = 0;
            if (examArr != null){
              foreach(Exam e in examArr){
                    mean+=e.grade;
                }
              mean /= (double)examArr.Count;
            }
           return mean;
    }
     }
    public bool this[Education index] {
        get  {return education == index; }
    }

public Student(Person p, Education e, int g){
    person = p;
    education = e;
    group = g;
    examArr = new List<Exam>();
    testArr = new List<Test>();
}
 public Student(){
    person = new Person();
    education = Education.Bachelor;
    group = 1;
    examArr = new List<Exam>();
    testArr = new List<Test>();
```

```
}

public void AddExams(List<Exam> exms){
    if(exms != null){
        foreach (Exam e in exms){
          examArr.Add((Exam)e.DeepCopy());


        }
    }
  }

  public void AddTests(List<Test> tsts){
    if (tsts != null) {
        foreach (Test t in tsts){
          testArr.Add((Test)t.DeepCopy());
        }
    }
  }
  public override string ToString()
     {
        string result;
        result = person.ToString() + " "  +
          educationType.ToString() +
         " " + group.ToString() + " ";

    if (examArr!=null) {
        foreach(Exam e in examArr)
        result = result + e.ToString() + " ";
```

```
        }
         if (testArr!=null) {
            foreach(Test t in testArr)
            result = result + t.ToString() + " ";
         }
            return result;


         }
      public override string ToShortString()
         {
            return person.first + " " + person.last+ " " + person.date.ToString() + " "
+educationType.ToString() + " " + group.ToString() + " " + meanValue.ToString() + "\n";
         }
        public override object DeepCopy()
      {
         Student copiedStudent = new Student(person, education, group);
         copiedStudent.AddExams(this.examArr);
         copiedStudent.AddTests(this.testArr);

         return copiedStudent;
      }



      public IEnumerable getAllExamsAndTest() {
      if (examArr!=null){
         foreach (var exam in examArr)
            yield return exam;
      }
```

```
if (testArr!=null){
    foreach (var test in testArr)
        yield return test;
}
}

public IEnumerable getSuccessfulExams(int k) {
    if(examArr!=null){
    foreach (var exam in examArr) {
        Exam ex = (Exam) exam;
        if (ex.grade > k)
            yield return exam;
    }
    }
}
public IEnumerable getDoneTestsAndExams(){
    if(examArr!=null){
        foreach(var exam in examArr){
            Exam ex = (Exam) exam;
        if (ex.grade > 2)
            yield return exam;
    }
        }
        if(testArr!=null){
        foreach(var test in testArr){
            Test t = (Test) test;
        if (t.isPassed)
            yield return test;
```

```
        }
            }
        }



    public IEnumerable getDoneTests(){
      if(examArr!=null){
        foreach(var exam in examArr){
            Exam ex = (Exam) exam;


        if(testArr!=null){
        foreach(var test in testArr){
            Test t = (Test) test;
        if (t.isPassed && t.subject==ex.subject && ex.grade>2)
            yield return test;
      }
            }
       }
     }
  }

   public IEnumerator GetEnumerator()
     {
      return new StudenEnumerator(this);
  }

}
```

**StudentCollection.cs**

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;

class StudentCollection  {

private static List<Student> studentList;

public double maxMean {
   get {return studentList.Max(s=> s.meanValue);}
}

public IEnumerable<Student> specialists {
   get {return studentList.Where(s=> s.educationType ==Education.Specialist);}
}

public StudentCollection(){
   studentList = new List<Student>();
}
public void AddDefaults(){
   studentList.Add(new Student());
}
public void AddStudents(Student[] students){
   for (int i=0; i<students.Length; i++){
      studentList.Add(students[i]);
   }
```

```
}

public override string ToString()
   {
      string res = "";
      if (studentList!=null){
         foreach(Student s in studentList){
            res += s.ToString() + " ";
         }
      }
      return res;
   }

public virtual string ToShortString(){
    string res = "";
      if (studentList!=null){
         foreach(Student s in studentList){
            res += s.ToShortString() + " " + s.exams.Count + " " + s.tests.Count + " ";
         }
      }
      return res;
}
public void compareByMeanValue(){
   studentList.Sort((IComparer<Student>) new StudentComparator());

}

public void compareByBirthdate(){
```

```
        studentList.Sort( (IComparer<Student>) new Person());


}

public void compareByLastName(){
    studentList.Sort(delegate (Student x, Student y)
          {
             return x.last.CompareTo(y.last);
          });
}

public IOrderedEnumerable<IGrouping<double, Student>> AverageMarkGroup(double value){
var queryLastNames =
        from student in studentList
        group student by student.meanValue==value into newGroup
        select newGroup;
return (IOrderedEnumerable<IGrouping<double, Student>>)queryLastNames;
}



}
```

**StudentComparator.cs**

```
using System;
using System.Collections.Generic;
using System.Diagnostics.CodeAnalysis;
```

```
class StudentComparator : IComparer<Student>
{
    public int Compare([AllowNull] Student x, [AllowNull] Student y)
    {

        if (x.meanValue > y.meanValue)
        {
            return 1;
        }
        else if (x.meanValue < y.meanValue)
        {
            return -1;
        }

        return 0;

    }




}
```

**StudentEnumerator.cs**

```
using System;
using System.Collections;
using System.Linq;

class StudenEnumerator : IEnumerator{
```

```
Student student;
string currSubj;
int position = -1;
ArrayList  subjs;
public StudenEnumerator(Student s){
    student = s;
}
 public object Current {
    get {
      return currSubj;
    }
  }
public bool MoveNext(){
    subjs = getSameSubjects();
    if (position == subjs.Count - 1) {
      Reset();
      return false;
    }
    position++;
    return true;
}
public void Reset(){
    position = -1;

}
public ArrayList getExamNames(){
    ArrayList subjs=null;
            for (int i = 0; i < student.exams.Count; i++)
```

```
        {
          Exam ex = (Exam) student.exams[i];
          subjs.Add(ex.subject);
        }


        return subjs;
}
public ArrayList getTestsNames(){
   ArrayList subjs=null;
        for (int i = 0; i < student.tests.Count; i++)
        {
          Test t = (Test) student.tests[i];
          subjs.Add(t.subject);
        }

        return subjs;
}
public ArrayList getSameSubjects(){
   var exams = getExamNames();
   var tests = getTestsNames();
var elements = System.Linq.Enumerable.Intersect(exams.ToArray(), tests.ToArray()).ToArray();
ArrayList result = new ArrayList(elements);
return result;
}



}
```

**Exam.cs**

```
using System;

class Exam : IDateAndCopy {

        public string subject;

        public int grade;

        public DateTime examDate;


    public DateTime Date { get => throw new NotImplementedException(); set => throw new
NotImplementedException(); }


    public Exam(string s, int g, DateTime d){

         subject = s;

         grade = g;

         examDate = d;

     }

      public Exam(){

         subject = "C# practice";

         grade = 5;

         examDate = new DateTime(2021, 6, 1);

     }

      DateTime IDateAndCopy.Date

        {

            get { return examDate; }

            set { examDate = value; }


        }

        public override string ToString()

        {
```

```
        return subject + " " + grade.ToString() + " " + examDate.ToString() + "\n";
    }
    public object DeepCopy()
    {
        Exam copiedExam = new Exam();
        copiedExam.examDate = examDate;
        copiedExam.grade = grade;
        copiedExam.subject = (string)subject.Clone();
        return (object)copiedExam;
    }

}
```

**Test.cs**

```
using System;

class Test{
    public string subject;
    public bool isPassed;

    public Test(){
        subject = "test subject";
        isPassed = true;
    }
     public Test(string s, bool i){
        subject = s;
        isPassed = i;
    }
      public override string ToString()
```

```
        {
            return subject + " " + isPassed.ToString()  + "\n";
        }
      public object DeepCopy()
        {
            Test copiedTest = new Test();
            copiedTest.subject = (string)subject.Clone();
            copiedTest.isPassed = isPassed;
            return (object)copiedTest;
        }
}
```

**TestCollections.cs**

```
using System;
using System.Collections.Generic;
using System.Text;

class TestCollections{
   private  List<Person> plist;
   private List<string> slist;
   private Dictionary<Person, Student> dperson;
   private Dictionary<string, Student> dstring;

 static public Student generate(int n)
        {
            return new Student();
        }
```

```
public TestCollections(int n)
{
 plist = new List<Person>();
 slist = new List<string>();
 dperson = new Dictionary<Person, Student>();
 dstring = new Dictionary<string, Student>();

 for (int i = 0; i <= n; i++)
   {
     StringBuilder str_build = new StringBuilder();
     Random random = new Random();
     str_build.Append(Convert.ToChar(Convert.ToInt32(Math.Floor(25 *
random.NextDouble())) + 65));
     Person randPerson = new Person(str_build.ToString(), str_build.ToString(), new
DateTime());
     Student randStudent = new Student(randPerson, Education.Bachelor, 1);

     StringBuilder str_build2 = new StringBuilder();
     Random random2 = new Random();
     str_build2.Append(Convert.ToChar(Convert.ToInt32(Math.Floor(25 *
random2.NextDouble())) + 65));
     Person randPerson2 = new Person(str_build2.ToString(), str_build2.ToString(), new
DateTime());
     Student randStudent2 = new Student(randPerson2, Education.Bachelor, 1);

     plist.Add(randPerson);
     slist.Add(str_build.ToString());
     dperson.Add(randPerson, randStudent);
     dstring.Add(str_build.ToString(), randStudent2);
   }
```

```
    }

    public void findElementInList()
    {
        Person randomPerson = new Person("ewfv", "erwfv", new DateTime());
        int start1 = Environment.TickCount;
        if (plist.Contains(plist[0]))
        {
            int end1 = Environment.TickCount - start1;
            Console.WriteLine("plist содержит 1й элемент {0}, время поиска  {1}", plist[1], end1);
        }
        int start2 = Environment.TickCount;

         if (plist.Contains(plist[plist.Count/2]))
        {
            int end2 = Environment.TickCount - start2;
            Console.WriteLine("plist содержит центральный элемент {0}, время поиска  {1}",
plist[plist.Count/2], end2);
        }
        int start3 = Environment.TickCount;

         if (plist.Contains(plist[plist.Count-1]))
        {
            int end3 = Environment.TickCount - start3;
            Console.WriteLine("plist содержит последний элемент {0}, время поиска  {1}",
plist[plist.Count-1], end3);
        }
        int start4 = Environment.TickCount;
```

```
            if (plist.Contains(randomPerson))
            {
                int end4 = Environment.TickCount - start4;
                Console.WriteLine("plist содержит элемент не из коллекции {0}, время поиска  {1}", randomPerson, end4);
            }

        }

        public void findElemetKeyDictionary(){
            Person p = new Person("hfjekw", "fkwe", new DateTime());
            Student randomSt = new Student(p, Education.SecondEducation, 2);
            int start1 = Environment.TickCount;
            if (dperson.ContainsKey(dperson[plist[0]]))
            {
                int end1 = Environment.TickCount - start1;
                Console.WriteLine("dperson содержит key 1 элемента {0}, время поиска  {1}", dperson[plist[0]], end1);
            }
            int start2 = Environment.TickCount;
            if (dperson.ContainsKey(dperson[plist[plist.Count/2]]))
            {
                int end2 = Environment.TickCount - start2;
                Console.WriteLine("dperson содержит key центрального элемента {0}, время поиска {1}", dperson[plist[plist.Count/2]], end2);
            }
            int start3 = Environment.TickCount;
            if (dperson.ContainsKey(dperson[plist[plist.Count-1]]))
```

```
        {
            int end3 = Environment.TickCount - start3;
            Console.WriteLine("dperson содержит key последнего элемента {0}, время поиска {1}", dperson[plist[plist.Count-1]], end3);
        }
        int start4 = Environment.TickCount;
        if (dperson.ContainsKey(randomSt))
        {
            int end4 = Environment.TickCount - start4;
            Console.WriteLine("dperson содержит key элемента не из коллекции {0}, время поиска  {1}", randomSt, end4);
        }

    }

    public void findEdlemetValueDictionary(){
        Student randomSt = new Student();
        int start1 = Environment.TickCount;
        if (dperson.ContainsValue(dperson[plist[0]]))
        {
            int end1 = Environment.TickCount - start1;
            Console.WriteLine("dperson содержит value 1 элемента {0}, время поиска  {1}", dperson[plist[0]], end1);
        }
        int start2 = Environment.TickCount;
        if (dperson.ContainsValue(dperson[plist[plist.Count/2]]))
        {
            int end2 = Environment.TickCount - start2;
            Console.WriteLine("dperson содержит value центрального элемента {0}, время поиска {1}", dperson[plist[plist.Count/2]], end2);
```

```
        }
        int start3 = Environment.TickCount;
        if (dperson.ContainsValue(dperson[plist[plist.Count-1]]))
        {
            int end3 = Environment.TickCount - start3;
            Console.WriteLine("dperson содержит value последнего элемента {0}, время поиска {1}", dperson[plist[plist.Count-1]], end3);
        }
        int start4 = Environment.TickCount;
        if (dperson.ContainsValue(randomSt))
        {
            int end4 = Environment.TickCount - start4;
            Console.WriteLine("dperson содержит value элемента не из коллекции {0}, время поиска  {1}", randomSt, end4);
        }


    }
  }
```

**Program.cs**

```
using System;
using System.Collections;
using System.Diagnostics;
using System.Collections.Generic;

public  enum Education {Specialist, Bachelor, SecondEducation}

class Program{
   static void Main(string[] args){
```

```
Person p1 = new Person("Fyodr", "Uncle", new DateTime(2000, 11, 4));
Person p2 = new Person();

Student s1 = new Student(p1, Education.Bachelor, 1);

Exam cs = new Exam();
Exam math = new Exam("math", 4, new DateTime(2021-5-6));
Exam physics = new Exam("physics", 4, new DateTime(2021-5-7));
Exam[] exams = new Exam[3];
exams[0]= cs;
exams[1]= math;
exams[2]= physics;
List<Exam> examsList = new List<Exam>();
examsList.AddRange(exams);
s1.AddExams(examsList);

Test t = new Test();
Test[] tests = new Test[1];
tests[0] = t;
List<Test> testsList = new List<Test>();
testsList.AddRange(tests);
s1.AddTests(testsList);

Student s2 = new Student();
Exam ex1 = new Exam();
Exam[] exams2 = new Exam[1];
exams2[0]= ex1;
```

```
List<Exam> examsList2 = new List<Exam>();
examsList2.AddRange(exams);
s1.AddExams(examsList2);


StudentCollection stcl = new StudentCollection();
Student[] st = new Student[2];
st[0] = s1;
st[1] = s2;
stcl.AddStudents(st);

Console.WriteLine(stcl.ToString());
stcl.compareByMeanValue();
Console.WriteLine("compareByMeanValue " + stcl.ToShortString());
stcl.compareByLastName();
Console.WriteLine("compareByLastName " + stcl.ToShortString());
stcl.compareByBirthdate();
Console.WriteLine("compareByBirthdate " + stcl.ToShortString());

Console.WriteLine("max mean value: " + stcl.maxMean);
Console.WriteLine("specialists: " + stcl.specialists.ToString());

// var grouped = stcl.AverageMarkGroup(4);
// Console.WriteLine("grouped: ", grouped.ToString());

TestCollections tc = new TestCollections(3);
tc.findElementInList();
tc.findElemetKeyDictionary();
```

```
        tc.findEdlemetValueDictionary();


    }
}
```

# Лабораторная работа 4. Варианты второго уровня Универсальные типы. Классы-коллекции. Методы расширения класса System.Linq.Enumerable

**Person.cs**

```
using System;
using System.Collections.Generic;
using System.Diagnostics.CodeAnalysis;
interface IDateAndCopy {
    object DeepCopy();
    DateTime Date {get; set;}
}
class Person: IDateAndCopy, IComparable, IComparer<Person>
    {
    protected string firstName;
    protected string lastName;
    protected DateTime birthDate;

    public Person (){
     firstName = "Sharik";
     lastName = "Matroskin";
     birthDate = new DateTime(1980, 2, 3);
    }

    public Person (string fn, string ln, DateTime bd){
        firstName = fn;
        lastName = ln;
        birthDate = bd;
```

```
}

DateTime IDateAndCopy.Date { get; set; }

public String first
      {
         get { return firstName; }
         set { firstName = value; }
      }
public String last
   {
      get { return lastName; }
      set { lastName = value; }
   }

public DateTime date
   {
      get { return birthDate; }
      set { birthDate = value; }
   }

public int birthYear
   {
      get { return birthDate.Year; }
      set { birthDate = new DateTime(value, birthDate.Month, birthDate.Day); }
   }
```

```
public override string ToString()
   {
      return firstName + " " + lastName + " " + birthDate.ToString() + "\n";
   }
public virtual string ToShortString()
   {
      return  firstName + " " +  lastName + "\n";
   }


public override  bool Equals(object obj){
     Person personObject = obj as Person;


     return obj != null &&
     this.firstName == personObject.firstName &&
        this.lastName == personObject.lastName &&
        this.birthDate == personObject.birthDate;
   }
   public static bool operator ==(Person p1, Person p2)
   {
      return p1.Equals(p2);
   }
   public static bool operator !=(Person p1, Person p2)
   {
      return !p1.Equals(p2);
   }
   public override int GetHashCode()
   {
```

```
        return this.firstName.GetHashCode()+this.lastName.GetHashCode()
+this.birthDate.GetHashCode();

    }

    public virtual object DeepCopy()

    {

        Person copiedPerson = new Person();

        copiedPerson.lastName = (string)lastName.Clone();

        copiedPerson.firstName = (string)firstName.Clone();

        copiedPerson.birthDate = birthDate;

        return copiedPerson;

    }

     public int CompareTo(object obj)

{

    Person p = obj as Person;

        return this.last.CompareTo(p.last);




}


public int Compare([AllowNull] Person x, [AllowNull] Person y)

{

     if (x.date < y.date)

        return 1;

    else if (x.date > y.date)

        return -1;

    else

        return 0;

}
```

```
    }
```

**Student.cs**

```
using System;
using System.Collections;
using System.Collections.Generic;

class Student : Person, IDateAndCopy, IEnumerable{

        private Person person;
        private Education education;
        private int group;
        private List<Exam> examArr = new List<Exam>();
        private List<Test> testArr = new List<Test>();



         public Education educationType {
              get { return education; }
            set { education = value; }
        }



        public int groupNumber {
              get { return group; }
            set {
                 if (value <= 100 || value > 599)
                {
                    throw new ArgumentOutOfRangeException("err: boundaries exceeded [100; 599)");
```

```
        }
        groupNumber = value;
        }
}

public List<Exam> exams {
      get { return  (List<Exam>)examArr; }
    set { examArr = (List<Exam>)value; }
}
 public List<Test> tests {
      get { return  (List<Test>)testArr; }
    set { testArr = (List<Test>)value; }
}
public Person p {
     get
    {
        return new Person((string)firstName.Clone(), (string)lastName.Clone(), birthDate);
    }
    set
    {
        firstName = (string)value.first.Clone();
        lastName = (string)value.last.Clone();
        birthDate = value.date;
    }
}

 public double meanValue {
      get {
```

```
            double mean = 0;
            if (examArr != null){
              foreach(Exam e in examArr){
                    mean+=e.grade;
                }
              mean /= (double)examArr.Count;
            }
           return mean;
    }
     }
    public bool this[Education index] {
        get  {return education == index; }
    }

public Student(Person p, Education e, int g){
    person = p;
    education = e;
    group = g;
    examArr = new List<Exam>();
    testArr = new List<Test>();
}
 public Student(){
    person = new Person();
    education = Education.Bachelor;
    group = 1;
    examArr = new List<Exam>();
    testArr = new List<Test>();
```

```
}

public void AddExams(List<Exam> exms){
    if(exms != null){
        foreach (Exam e in exms){
          examArr.Add((Exam)e.DeepCopy());


        }
    }
  }

  public void AddTests(List<Test> tsts){
    if (tsts != null) {
        foreach (Test t in tsts){
          testArr.Add((Test)t.DeepCopy());
        }
    }
  }
  public override string ToString()
     {
        string result;
        result = person.ToString() + " "  +
          educationType.ToString() +
         " " + group.ToString() + " ";

    if (examArr!=null) {
        foreach(Exam e in examArr)
        result = result + e.ToString() + " ";
```

```
        }
         if (testArr!=null) {
            foreach(Test t in testArr)
            result = result + t.ToString() + " ";
         }
            return result;


         }
      public override string ToShortString()
         {
            return person.first + " " + person.last+ " " + person.date.ToString() + " "
+educationType.ToString() + " " + group.ToString() + " " + meanValue.ToString() + "\n";
         }
        public override object DeepCopy()
      {
         Student copiedStudent = new Student(person, education, group);
         copiedStudent.AddExams(this.examArr);
         copiedStudent.AddTests(this.testArr);

         return copiedStudent;
      }



      public IEnumerable getAllExamsAndTest() {
      if (examArr!=null){
         foreach (var exam in examArr)
            yield return exam;
      }
```

```
if (testArr!=null){
    foreach (var test in testArr)
        yield return test;
}
}

public IEnumerable getSuccessfulExams(int k) {
    if(examArr!=null){
    foreach (var exam in examArr) {
        Exam ex = (Exam) exam;
        if (ex.grade > k)
            yield return exam;
    }
    }
}
public IEnumerable getDoneTestsAndExams(){
    if(examArr!=null){
        foreach(var exam in examArr){
            Exam ex = (Exam) exam;
        if (ex.grade > 2)
            yield return exam;
    }
        }
        if(testArr!=null){
        foreach(var test in testArr){
            Test t = (Test) test;
        if (t.isPassed)
            yield return test;
```

```
        }
            }
        }



     public IEnumerable getDoneTests(){
       if(examArr!=null){
          foreach(var exam in examArr){
             Exam ex = (Exam) exam;


          if(testArr!=null){
          foreach(var test in testArr){
             Test t = (Test) test;
          if (t.isPassed && t.subject==ex.subject && ex.grade>2)
             yield return test;
       }
          }
     }
   }
}


 public IEnumerator GetEnumerator()
   {
    return new StudenEnumerator(this);
}


  public void SortBySubject()
```

```
    {
        examArr.Sort(delegate (Exam x, Exam y)
        {
            return x.subject.CompareTo(y.subject);
        });
    }
    public void SortByGrade()
    {
        examArr.Sort(new ExamComparer());

    }
    public void SortByDate()
    {
        examArr.Sort(new ExamComparer());
    }

}
```

**StudentCollection.cs**

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;

class StudentCollection<TKey>  {

private static List<Student> studentList;
private static Dictionary<TKey, Student> studentDictionary = new Dictionary<TKey, Student>();
private static KeySelector<TKey> key;
```

```
public double maxMean {
    get {return studentList.Max(s=> s.meanValue);}
}

public IEnumerable<Student> specialists {
    get {return studentList.Where(s=> s.educationType ==Education.Specialist);}
}

public StudentCollection(){
    studentList = new List<Student>();
}

public StudentCollection(KeySelector<TKey> k){
  key = k;
}
public void AddDefaults(){
    studentList.Add(new Student());
}
public void AddStudents(Student[] students){
    for (int i=0; i<students.Length; i++){
        studentList.Add(students[i]);
    }
}

public override string ToString()
```

```
    {
        string res = "";
        if (studentList!=null){
            foreach(Student s in studentList){
                res += s.ToString() + " ";
            }
        }
        return res;
    }

public virtual string ToShortString(){
     string res = "";
        if (studentList!=null){
            foreach(Student s in studentList){
                res += s.ToShortString() + " " + s.exams.Count + " " + s.tests.Count + " ";
            }
        }
        return res;
}
public void compareByMeanValue(){
    studentList.Sort((IComparer<Student>) new StudentComparator());

}

public void compareByBirthdate(){
        studentList.Sort( (IComparer<Student>) new Person());

}
```

```
public void compareByLastName(){
    studentList.Sort(delegate (Student x, Student y)
        {
            return x.last.CompareTo(y.last);
        });
}

public IOrderedEnumerable<IGrouping<double, Student>> AverageMarkGroup(double value){
var queryLastNames =
        from student in studentList
        group student by student.meanValue==value into newGroup
        select newGroup;
return (IOrderedEnumerable<IGrouping<double, Student>>)queryLastNames;
}



}
```

**StudentComparator.cs**

```
using System;
using System.Collections.Generic;
using System.Diagnostics.CodeAnalysis;

class StudentComparator : IComparer<Student>
{
   public int Compare([AllowNull] Student x, [AllowNull] Student y)
```

```
    {

        if (x.meanValue > y.meanValue)
        {
            return 1;
        }
        else if (x.meanValue < y.meanValue)
        {
            return -1;
        }

        return 0;

    }




}
```

**StudentEnumerator.cs**

```
using System;
using System.Collections;
using System.Linq;

class StudenEnumerator : IEnumerator{

Student student;
string currSubj;
int position = -1;
```

```
ArrayList  subjs;
public StudenEnumerator(Student s){
    student = s;
}
 public object Current {
    get {
      return currSubj;
    }
  }
public bool MoveNext(){
    subjs = getSameSubjects();
    if (position == subjs.Count - 1) {
      Reset();
      return false;
    }
    position++;
    return true;
}
public void Reset(){
    position = -1;

}
public ArrayList getExamNames(){
    ArrayList subjs=null;
            for (int i = 0; i < student.exams.Count; i++)
            {
              Exam ex = (Exam) student.exams[i];
              subjs.Add(ex.subject);
```

```
        }


        return subjs;
}
public ArrayList getTestsNames(){
   ArrayList subjs=null;
        for (int i = 0; i < student.tests.Count; i++)
        {
          Test t = (Test) student.tests[i];
          subjs.Add(t.subject);
        }

        return subjs;
}
public ArrayList getSameSubjects(){
   var exams = getExamNames();
   var tests = getTestsNames();
var elements = System.Linq.Enumerable.Intersect(exams.ToArray(), tests.ToArray()).ToArray();
ArrayList result = new ArrayList(elements);
return result;
}



}
```

**Exam.cs**

```
using System;
using System.Collections.Generic;
```

```
class Exam : IDateAndCopy, IComparable, IComparer<Exam> {
        public string subject;
        public int grade;
        public DateTime examDate;


    public DateTime Date { get => throw new NotImplementedException(); set => throw new
NotImplementedException(); }


    public Exam(string s, int g, DateTime d){
         subject = s;
         grade = g;
         examDate = d;
     }
      public Exam(){
         subject = "C# practice";
         grade = 5;
         examDate = new DateTime(2021, 6, 1);
     }
      DateTime IDateAndCopy.Date
       {
           get { return examDate; }
           set { examDate = value; }


       }
       public override string ToString()
       {
           return subject + " " + grade.ToString() + " " + examDate.ToString() + "\n";
       }
```

```
        public object DeepCopy()
        {
            Exam copiedExam = new Exam();
            copiedExam.examDate = examDate;
            copiedExam.grade = grade;
            copiedExam.subject = (string)subject.Clone();
            return (object)copiedExam;
        }

        public int CompareTo(object obj){
            return subject.CompareTo(((Exam)obj).subject);
        }

         public int Compare(Exam x, Exam y)
    {
         if (x.grade < y.grade)
            return 1;
        else if (x.grade > y.grade)
            return -1;
        else
            return 0;
    }

    }
```

**ExamComparer.cs**

```
using System;
using System.Collections.Generic;
using System.Linq;
```

```
using System.Text;
using System.Threading.Tasks;

    class ExamComparer : IComparer<Exam>

    {
        public int Compare(Exam x, Exam y)
        {
            if (x.examDate == y.examDate)
                return 0;
            else
                if (x.examDate < y.examDate)
                return -1;
            else
                return 1;
        }
    }
```

**KeySelector.cs**

```
    delegate TKey KeySelector<TKey>(Student st);
```

**Test.cs**

```
using System;

class Test{
    public string subject;
    public bool isPassed;
```

```
    public Test(){
        subject = "test subject";
        isPassed = true;
    }
     public Test(string s, bool i){
        subject = s;
        isPassed = i;
    }
      public override string ToString()
        {
            return subject + " " + isPassed.ToString()  + "\n";
        }
     public object DeepCopy()
        {
            Test copiedTest = new Test();
            copiedTest.subject = (string)subject.Clone();
            copiedTest.isPassed = isPassed;
            return (object)copiedTest;
        }
}
```

**TestCollections.cs**

```
using System;
using System.Collections.Generic;
using System.Text;
    public delegate KeyValuePair<TKey, TValue> GenerateElement<TKey, TValue>(int j);
    public delegate bool FindElement<TKey>(TKey key);

class TestCollections<TKey,TValue>{
```

```
  private  List<TKey> plist;
  private List<string> slist;
  private Dictionary<TKey, TValue> dperson;
  private Dictionary<string, TValue> dstring;


static public Student generate(int n)
      {
         return new Student();
      }



      public TestCollections(int n, GenerateElement<TKey,TValue> generator)
      {
       plist = new List<TKey>();
       slist = new List<string>();
       dperson = new Dictionary<TKey, TValue>();
       dstring = new Dictionary<string, TValue>();
      KeyValuePair<TKey, TValue> res;
       for (int i = 0; i <= n; i++)
         {
            res = generator(i);
            StringBuilder str_build = new StringBuilder();
            Random random = new Random();
            str_build.Append(Convert.ToChar(Convert.ToInt32(Math.Floor(25 *
random.NextDouble())) + 65));
            Person randPerson = new Person(str_build.ToString(), str_build.ToString(), new
DateTime());
            Student randStudent = new Student(randPerson, Education.Bachelor, 1);
```

```
            StringBuilder str_build2 = new StringBuilder();

            Random random2 = new Random();

            str_build2.Append(Convert.ToChar(Convert.ToInt32(Math.Floor(25 *
random2.NextDouble())) + 65));

            Person randPerson2 = new Person(str_build2.ToString(), str_build2.ToString(), new
DateTime());

            Student randStudent2 = new Student(randPerson2, Education.Bachelor, 1);



            plist.Add(res.Key);

            slist.Add(str_build.ToString());

            dperson.Add(res.Key, res.Value);

            dstring.Add(str_build.ToString(), res.Value);

        }


    }


    public  bool FindInlist(TKey key)

    {

       return plist.Contains(key);

    }

    public bool FindInslist(string key)

    {

        return slist.Contains(key);

    }

    public bool FindKeyInDict(TKey key)

    {

         return dperson.ContainsKey(key);

    }

    public bool FindKeyInsDict(string key)
```

```
        {
          return  dstring.ContainsKey(key);
        }
        public bool FindVInDict(TValue key)
        {
           return dperson.ContainsValue(key);
        }

    }
```

**Program.cs**

```
using System;
using System.Collections;
using System.Diagnostics;
using System.Collections.Generic;

public  enum Education {Specialist, Bachelor, SecondEducation}

class Program{
    static void Main(string[] args){

        Person p1 = new Person("Fyodr", "Uncle", new DateTime(2000, 11, 4));
        Person p2 = new Person();

        Student s1 = new Student(p1, Education.Bachelor, 1);

        Exam cs = new Exam();
        Exam math = new Exam("math", 4, new DateTime(2021-5-6));
```

```
Exam physics = new Exam("physics", 4, new DateTime(2021-5-7));
Exam[] exams = new Exam[3];
exams[0]= cs;
exams[1]= math;
exams[2]= physics;
List<Exam> examsList = new List<Exam>();
examsList.AddRange(exams);
s1.AddExams(examsList);

Test t = new Test();
Test[] tests = new Test[1];
tests[0] = t;
List<Test> testsList = new List<Test>();
testsList.AddRange(tests);
s1.AddTests(testsList);



s1.SortBySubject();
foreach (var e in s1.exams)
{
   Console.WriteLine(e);
}
s1.SortByGrade();
foreach (var e in s1.exams)
{
   Console.WriteLine(e);
}
s1.SortByDate();
```

```
foreach (var e in s1.exams)
{
    Console.WriteLine(e);
}
```

```
Student s2 = new Student();
Exam ex1 = new Exam();
Exam[] exams2 = new Exam[1];
exams2[0] = ex1;
List<Exam> examsList2 = new List<Exam>();
examsList2.AddRange(exams);
s1.AddExams(examsList2);
```

```
StudentCollection<Student> stcl = new StudentCollection<Student>();
Student[] st = new Student[2];
st[0] = s1;
st[1] = s2;
stcl.AddStudents(st);
```

```
Console.WriteLine(stcl.ToString());
```

```
Console.WriteLine("max mean value: " + stcl.maxMean);
Console.WriteLine("specialists: " + stcl.specialists.ToString());
```

```
        var grouped = stcl.AverageMarkGroup(4);
        Console.WriteLine("grouped: ", grouped.ToString());


        GenerateElement<Person, Student> generator = x =>
        {
            Person p = new Person("Yellow", "White", new DateTime(2000, 11, 4));
            Student s = new Student(p, Education.Bachelor, 1);
            return new KeyValuePair<Person, Student>(p, s);
          };
        TestCollections<Person, Student> tc = new TestCollections<Person, Student>(3, generator);



    }
}
```

# Лабораторная работа 5. Варианты второго уровня Делегаты. События

**Person.cs**

```
using System;
using System.Collections.Generic;
using System.Diagnostics.CodeAnalysis;
interface IDateAndCopy {
    object DeepCopy();
    DateTime Date {get; set;}
}
class Person: IDateAndCopy, IComparable, IComparer<Person>
    {
    protected string firstName;
    protected string lastName;
    protected DateTime birthDate;

    public Person (){
     firstName = "Sharik";
     lastName = "Matroskin";
     birthDate = new DateTime(1980, 2, 3);
    }

    public Person (string fn, string ln, DateTime bd){
        firstName = fn;
        lastName = ln;
        birthDate = bd;
```

```
}

DateTime IDateAndCopy.Date { get; set; }

public String first
      {
         get { return firstName; }
         set { firstName = value; }
      }
public String last
   {
      get { return lastName; }
      set { lastName = value; }
   }

public DateTime date
   {
      get { return birthDate; }
      set { birthDate = value; }
   }

public int birthYear
   {
      get { return birthDate.Year; }
      set { birthDate = new DateTime(value, birthDate.Month, birthDate.Day); }
   }
```

```
public override string ToString()
   {
      return firstName + " " + lastName + " " + birthDate.ToString() + "\n";
   }
public virtual string ToShortString()
   {
      return  firstName + " " +  lastName + "\n";
   }


public override  bool Equals(object obj){
     Person personObject = obj as Person;


     return obj != null &&
     this.firstName == personObject.firstName &&
        this.lastName == personObject.lastName &&
        this.birthDate == personObject.birthDate;
   }
   public static bool operator ==(Person p1, Person p2)
   {
      return p1.Equals(p2);
   }
   public static bool operator !=(Person p1, Person p2)
   {
      return !p1.Equals(p2);
   }
   public override int GetHashCode()
   {
```

```
        return this.firstName.GetHashCode()+this.lastName.GetHashCode()
+this.birthDate.GetHashCode();

    }

    public virtual object DeepCopy()

    {

        Person copiedPerson = new Person();

        copiedPerson.lastName = (string)lastName.Clone();

        copiedPerson.firstName = (string)firstName.Clone();

        copiedPerson.birthDate = birthDate;

        return copiedPerson;

    }

     public int CompareTo(object obj)

{

    Person p = obj as Person;

        return this.last.CompareTo(p.last);




}


public int Compare([AllowNull] Person x, [AllowNull] Person y)

{

     if (x.date < y.date)

        return 1;

    else if (x.date > y.date)

        return -1;

    else

        return 0;

}
```

```
    }
```

**Student.cs**

```
using System;
using System.Collections;
using System.Collections.Generic;

class Student : Person, IDateAndCopy, IEnumerable{

        private Person person;
        private Education education;
        private int group;
        private List<Exam> examArr = new List<Exam>();
        private List<Test> testArr = new List<Test>();



         public Education educationType {
              get { return education; }
            set { education = value; }
        }



        public int groupNumber {
              get { return group; }
            set {
                 if (value <= 100 || value > 599)
                {
                    throw new ArgumentOutOfRangeException("err: boundaries exceeded [100; 599)");
```

```
        }
        groupNumber = value;
        }
}

public List<Exam> exams {
      get { return  (List<Exam>)examArr; }
    set { examArr = (List<Exam>)value; }
}
 public List<Test> tests {
      get { return  (List<Test>)testArr; }
    set { testArr = (List<Test>)value; }
}
public Person p {
     get
    {
        return new Person((string)firstName.Clone(), (string)lastName.Clone(), birthDate);
    }
    set
    {
        firstName = (string)value.first.Clone();
        lastName = (string)value.last.Clone();
        birthDate = value.date;
    }
}

 public double meanValue {
      get {
```

```
            double mean = 0;
            if (examArr != null){
              foreach(Exam e in examArr){
                    mean+=e.grade;
                }
              mean /= (double)examArr.Count;
            }
           return mean;
    }
     }
    public bool this[Education index] {
        get  {return education == index; }
    }

public Student(Person p, Education e, int g){
    person = p;
    education = e;
    group = g;
    examArr = new List<Exam>();
    testArr = new List<Test>();
}
 public Student(){
    person = new Person();
    education = Education.Bachelor;
    group = 1;
    examArr = new List<Exam>();
    testArr = new List<Test>();
```

```
}

public void AddExams(List<Exam> exms){
    if(exms != null){
        foreach (Exam e in exms){
          examArr.Add((Exam)e.DeepCopy());


        }
    }
  }

  public void AddTests(List<Test> tsts){
    if (tsts != null) {
        foreach (Test t in tsts){
          testArr.Add((Test)t.DeepCopy());
        }
    }
  }
  public override string ToString()
     {
        string result;
        result = person.ToString() + " "  +
          educationType.ToString() +
         " " + group.ToString() + " ";

    if (examArr!=null) {
        foreach(Exam e in examArr)
        result = result + e.ToString() + " ";
```

```
        }
         if (testArr!=null) {
            foreach(Test t in testArr)
            result = result + t.ToString() + " ";
         }
            return result;


         }
      public override string ToShortString()
         {
            return person.first + " " + person.last+ " " + person.date.ToString() + " "
+educationType.ToString() + " " + group.ToString() + " " + meanValue.ToString() + "\n";
         }
        public override object DeepCopy()
      {
         Student copiedStudent = new Student(person, education, group);
         copiedStudent.AddExams(this.examArr);
         copiedStudent.AddTests(this.testArr);

         return copiedStudent;
      }



      public IEnumerable getAllExamsAndTest() {
      if (examArr!=null){
         foreach (var exam in examArr)
            yield return exam;
      }
```

```
if (testArr!=null){
    foreach (var test in testArr)
        yield return test;
}
}

public IEnumerable getSuccessfulExams(int k) {
    if(examArr!=null){
    foreach (var exam in examArr) {
        Exam ex = (Exam) exam;
        if (ex.grade > k)
            yield return exam;
    }
    }
}
public IEnumerable getDoneTestsAndExams(){
    if(examArr!=null){
        foreach(var exam in examArr){
            Exam ex = (Exam) exam;
        if (ex.grade > 2)
            yield return exam;
    }
        }
        if(testArr!=null){
        foreach(var test in testArr){
            Test t = (Test) test;
        if (t.isPassed)
            yield return test;
```

```
        }
            }
        }



     public IEnumerable getDoneTests(){
       if(examArr!=null){
          foreach(var exam in examArr){
             Exam ex = (Exam) exam;


          if(testArr!=null){
          foreach(var test in testArr){
             Test t = (Test) test;
          if (t.isPassed && t.subject==ex.subject && ex.grade>2)
             yield return test;
       }
          }
     }
   }
}


 public IEnumerator GetEnumerator()
   {
    return new StudenEnumerator(this);
}


  public void SortBySubject()
```

```
    {
        examArr.Sort(delegate (Exam x, Exam y)
        {
            return x.subject.CompareTo(y.subject);
        });
    }
    public void SortByGrade()
    {
        examArr.Sort(new ExamComparer());

    }
    public void SortByDate()
    {
        examArr.Sort(new ExamComparer());
    }

}
```

**StudentCollection.cs**

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using testlab;



public delegate void StudentListHandler<TKey>(object source,
StudentListHandlerEventArgs<TKey> args);
```

```
class StudentCollection<TKey>  {



public string name { get; set; }
private static List<Student> studentList;
private static Dictionary<TKey, Student> studentDictionary = new Dictionary<TKey, Student>();
private static KeySelector<TKey> key;
public event StudentListHandler<TKey> StudentChanged;




    public double maxMean {
    get {return studentList.Max(s=> s.meanValue);}
}

public IEnumerable<Student> specialists {
    get {return studentList.Where(s=> s.educationType ==Education.Specialist);}
}

public StudentCollection(){
    studentList = new List<Student>();
}

public StudentCollection(KeySelector<TKey> k){
  key = k;
}

    private KeySelector<TKey> _selector;
```

```
    public void AddDefaults()
    {
        Student s = new Student();
        StudentChanged(this, new StudentListHandlerEventArgs<TKey>(name, Update.Add,
_selector(s)));
        studentDictionary.Add(_selector(s), s);
    }
    public void AddStudents(params Student[] a)
    {

        foreach (Student x in a)
        {
            StudentChanged(this, new StudentListHandlerEventArgs<TKey>(name, Update.Add,
_selector(x)));
            studentDictionary.Add(_selector(x), x);
        }
    }

    public void Remove(int j)
    {

        studentDictionary.RemoveAt(j);
    }




    public override string ToString()
    {
```

```
        string res = "";
        if (studentList!=null){
            foreach(Student s in studentList){
                res += s.ToString() + " ";
            }
        }
        return res;
    }

public virtual string ToShortString(){
     string res = "";
        if (studentList!=null){
            foreach(Student s in studentList){
                res += s.ToShortString() + " " + s.exams.Count + " " + s.tests.Count + " ";
            }
        }
        return res;
}
public void compareByMeanValue(){
    studentList.Sort((IComparer<Student>) new StudentComparator());

}

public void compareByBirthdate(){
       studentList.Sort( (IComparer<Student>) new Person());

}
```

```
public void compareByLastName(){
    studentList.Sort(delegate (Student x, Student y)
        {
            return x.last.CompareTo(y.last);
        });
}

   public IOrderedEnumerable<IGrouping<double, Student>> AverageMarkGroup(double value){
var queryLastNames =
        from student in studentList
        group student by student.meanValue==value into newGroup
        select newGroup;
return (IOrderedEnumerable<IGrouping<double, Student>>)queryLastNames;
}

   public bool Replace(Student a, Student b)
   {
      TKey k = default(TKey);
      if (studentDictionary.ContainsValue(a))
      {
         foreach (KeyValuePair<TKey, Student> x in studentDictionary)
         {
            if (x.Value == a)
            {
               k = x.Key;
               break;
            }
         }
```

```
            StudentChanged(this, new StudentListHandlerEventArgs<TKey>(name, Update.Replace,
k));
            studentDictionary[k] = b;
            return true;
        }
        else return false;
    }



}
```

**StudentComparator.cs**

```
using System;
using System.Collections.Generic;
using System.Diagnostics.CodeAnalysis;

class StudentComparator : IComparer<Student>
{
    public int Compare([AllowNull] Student x, [AllowNull] Student y)
    {

        if (x.meanValue > y.meanValue)
        {
            return 1;
        }
        else if (x.meanValue < y.meanValue)
        {
```

```
            return -1;
        }

        return 0;

    }




}
```

**StudentEnumerator.cs**

```
using System;
using System.Collections;
using System.Linq;

class StudenEnumerator : IEnumerator{

Student student;
string currSubj;
int position = -1;
ArrayList  subjs;
public StudenEnumerator(Student s){
    student = s;
}
 public object Current {
    get {
      return currSubj;
    }
```

```
  }
public bool MoveNext(){
    subjs = getSameSubjects();
    if (position == subjs.Count - 1) {
      Reset();
      return false;
    }
    position++;
    return true;
}
public void Reset(){
    position = -1;


}
public ArrayList getExamNames(){
    ArrayList subjs=null;
            for (int i = 0; i < student.exams.Count; i++)
            {
              Exam ex = (Exam) student.exams[i];
              subjs.Add(ex.subject);
            }



            return subjs;
}
public ArrayList getTestsNames(){
    ArrayList subjs=null;
            for (int i = 0; i < student.tests.Count; i++)
```

```
            {
              Test t = (Test) student.tests[i];
              subjs.Add(t.subject);
            }

            return subjs;
}
public ArrayList getSameSubjects(){
    var exams = getExamNames();
    var tests = getTestsNames();
var elements = System.Linq.Enumerable.Intersect(exams.ToArray(), tests.ToArray()).ToArray();
ArrayList result = new ArrayList(elements);
return result;
}



}
```

**StudentListHandlerEventArgs.cs**

```
using System;
namespace testlab
{
    public enum Update { Add, Replace, Delete };

    public class StudentListHandlerEventArgs<TKey> : EventArgs
    {

        public string name { get; set; }
        public Update Type { get; set; }
```

```
        public TKey ChangedKey { get; set; }


        public StudentListHandlerEventArgs(string n, Update T, TKey C)
        {
            name = n;
            Type = T;
            ChangedKey = C;
        }
        public override string ToString()
        {
            return name + "\n" + Type.ToString() + "\n" + ChangedKey.ToString();
        }
    }
}
```

**Journal.cs**

```
using System;
namespace testlab
{
    public class Journal
    {
   List<JournalEntry> changes = new List<JournalEntry>();


        public void handler(object sender, StudentListHandlerEventArgs<string> e)
        {
```

```
            changes.Add(new JournalEntry(e.NameOfCollection, e.Reason, e.PropertyName,
e.RegisterNumber));
        }


        public override string ToString()
        {
            string text = "";
            for (int i = 0; i < changes.Count; i++)
                text += changes[i] + "\n";
            return text;
        }
    }
}
```

**JournalEntry.cs**

```
using System;

namespace testlab
{
    public class JournalEntry
    {
        string NameOfCollection { get; set; }
        Update U { get; set; }
        string PropertyName { get; set; }
        int RegisterNumber { get; set; }
        public JournalEntry(string nameOfCollection, Update update, string propertyName, int
registredNumber)
        {
            NameOfCollection = nameOfCollection; U = update; PropertyName = propertyName;
RegisterNumber = registredNumber;
```

```
        }
        public override string ToString()
        {
            return "NameOfCollection: " + NameOfCollection + ", Update: " + Update + ",
PropertyName: " + PropertyName + ", RegistredNumber: " + RegisterNumber;
        }
    }
}
```

**Program.cs**

```
using System;
using System.Collections;
using System.Diagnostics;
using System.Collections.Generic;
using testlab;

public  enum Education {Specialist, Bachelor, SecondEducation}

class Program{
    static void Main(string[] args){

        Person p1 = new Person("Fyodr", "Uncle", new DateTime(2000, 11, 4));
        Person p2 = new Person();

        Student s1 = new Student(p1, Education.Bachelor, 1);
        Student s2 = new Student();
```

```
Person p3 = new Person();
Person p4 = new Person();

Student s3 = new Student(p3, Education.Bachelor, 1);
Student s4 = new Student(p4, Education.Specialist, 2);


StudentCollection<Student> stcl = new StudentCollection<Student>();
StudentCollection<Student> stcl2 = new StudentCollection<Student>();


Journal j1 = new Journal();
Journal j2 = new Journal();

stcl.StudentChanged += j1.handler;
stcl2.StudentChanged += j2.handler;


Student[] st = new Student[2];
st[0] = s1;
st[1] = s2;
stcl.AddStudents(st);

Student[] st2 = new Student[2];
st2[0] = s3;
st2[1] = s4;
stcl2.AddStudents(st2);
```

```
        stcl.Remove(0);
        stcl.Replace(s2, s3);

        Console.WriteLine(j1);
        Console.WriteLine(j2);
    }
}
```

# Лабораторная работа 5. Варианты второго уровня Сериализация. Взаимодействие управляемого и неуправляемого кода

Matrix.cs

```
using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;


namespace testlab

{

   public class Matrix

   {


      double[] A;

      public Matrix() { }

      public Matrix(int n)

      {

         A = new double[n];


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

            A[i] = (n + 1) - i;

      }


      public double[] Solve(double[] right)

      {


         int n = A.Length;
```

```
double[] x = new double[n];
x[0] = 1.0 / A[0];
double[] y = new double[n];
y[0] = x[0];
for (int k = 1; k <= n - 1; k++)
{
    double Fk = 0;
    for (int i = 0; i < k; i++)
        Fk += A[k - i] * x[i];

    double Gk = 0;
    for (int i = 0; i < k; i++)
        Gk += A[i + 1] * y[i];

    double rk = 1.0 / (1.0 - Fk * Gk);

    double sk = -rk * Fk;

    double tk = -rk * Gk;
    double[] x_old = new double[k];
    for (int i = 0; i < k; i++)
        x_old[i] = x[i];
    double[] y_old = new double[k];
    for (int i = 0; i < k; i++)
        y_old[i] = y[i];
    x[0] = x_old[0] * rk + 0 * sk;
    for (int i = 1; i < k; i++)
        x[i] = x_old[i] * rk + y_old[i - 1] * sk;
```

```
    x[k] = 0 * rk + y_old[k - 1] * sk;
    y[0] = x_old[0] * tk + 0 * rk;
    for (int i = 1; i < k; i++)
        y[i] = x_old[i] * tk + y_old[i - 1] * rk;
    y[k] = 0 * tk + y_old[k - 1] * rk;
}


double[,] matrix_a = new double[n, n];
double[,] matrix_b = new double[n, n];
double[,] matrix_c = new double[n, n];
double[,] matrix_d = new double[n, n];


for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
    {
        matrix_a[i, j] = (i >= j) ? x[i - j] : 0;
        matrix_b[i, j] = (j >= i) ? y[n - 1 - j + i] : 0;
        matrix_c[i, j] = (i > j) ? y[i - 1 - j] : 0;
        matrix_d[i, j] = (j > i) ? x[n - 1 - j + 1 + i] : 0;
    }

double[,] matrix_ab = new double[n, n];
double[,] matrix_cd = new double[n, n];

for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
```

```
    {
        double sum_ab = 0;
        double sum_cd = 0;
        for (int k = 0; k < n; k++)
        {
            sum_ab += matrix_a[i, k] * matrix_b[k, j];
            sum_cd += matrix_c[i, k] * matrix_d[k, j];
        }
        matrix_ab[i, j] = sum_ab;
        matrix_cd[i, j] = sum_cd;
    }

double[,] matrix_res = new double[n, n];

for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
        matrix_res[i, j] = (1.0 / x[0]) * (matrix_ab[i, j] - matrix_cd[i, j]);



double[] answer = new double[n];

for (int i = 0; i < n; i++)
{
    answer[i] = 0;
    for (int j = 0; j < n; j++)
        answer[i] += matrix_res[i, j] * right[j];
}
```

```
            return answer;
        }

        public override string ToString()
        {
            string text = "";
            for (int i = 0; i < A.Length; i++)
                text += A[i] + " ";
            return text;
        }
    }
}
```

TimeItem.cs

```
using System;
namespace testlab
{

    public class TimeItem
    {

        public int n { get; set; }
        public int k { get; set; }
        public double CsTime { get; set; }
        public double CppTime { get; set; }
        public double Factor { get { return CsTime / CppTime; } }

        public TimeItem() { }
```

```
        public TimeItem(int n, int k, double csTime, double cppTime)
        {
            this.n = n; this.k = k; this.CsTime = csTime; this.CppTime = cppTime;
        }

        public override string ToString()
        {
            return "Порядок: " + n + "; Повторы: " + k + "; Время C#: " + CsTime + "; Время C++: "
+ CppTime + "; коэф: " + Factor;
        }
    }
}
```

TimeList.cs

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

namespace testlab
{
    public class TimeList
    {
        public List<TimeItem> items = new List<TimeItem>();

        public void Add(TimeItem timeItem)
```

```
{
    items.Add(timeItem);
}

public static TimeList Load(string fileName)
{
    var serializer = new XmlSerializer(typeof(TimeList));

    using (TextReader reader = new StreamReader(fileName))
    {
        return (serializer.Deserialize(reader) as TimeList);
    }
}

public void Save(string fileName)
{
    XmlSerializer serializer = new XmlSerializer(typeof(TimeList));

    using (TextWriter writer = new StreamWriter(fileName))
    {
        serializer.Serialize(writer, this);
    }

}
public override string ToString()
{
    for (int i = 0; i < items.Count; i++)
        text += (i + 1) + ". " + items[i].ToString() + "\n";
```

```
            return text;
        }
    }
}
```

Program.cs

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Runtime.InteropServices;
using System.Text;
using System.Threading.Tasks;

namespace testlab
{
    class Program
    {
        const string DllPath = " ./testlab/Lab_5_C.dll";

        [DllImport(DllPath)]
        static extern int Version();

        [DllImport(DllPath)]
        static extern double FirstMethod_C(int n, int k);
```

```
[DllImport(DllPath)]
private static unsafe extern void SecondMethod_C(int n, double[] matrix_array, double[] right_array, double* answer_array);

static void Main(string[] args)
{

    Console.WriteLine();

    string fileName = "timeList.xml";
    TimeList timeList = new TimeList();

    int n = 100, k = 5;
    timeList.Add(new TimeItem(n, k, FirstMethod(n, k), FirstMethod_C(n, k)));

    n = 200; k = 1;
    Console.WriteLine("Запуск в c++ dll...");
    double cppTime = FirstMethod_C(n, k);
    Console.WriteLine("Время выполнения", n, k, cppTime);
    Console.WriteLine("Запуск в c#...");
    double csTime = FirstMethod(n, k);
    Console.WriteLine("Время выполнения 1", n, k, csTime);
    csTime = FirstMethod(n, k);
    Console.WriteLine("Время выполнения 2", n, k, csTime);
    csTime = FirstMethod(n, k);
    Console.WriteLine("Время выполнения 3", n, k, csTime);
    timeList.Add(new TimeItem(n, k, csTime, cppTime));
```

```
        Console.WriteLine(fileName);
        timeList.Save(fileName);
        Console.WriteLine();

        double[] matrix_array = new double[3] { 4, 3, 2 };
        double[] right_array = new double[3] { 10, 20, 30 };
        double[] answer_array = new double[3];
        Console.WriteLine("Передача данных c++: M = ({0}; {1}; {2}), R = ({3}; {4}; {5})",
matrix_array[0], matrix_array[1], matrix_array[2], right_array[0], right_array[1], right_array[2]);

            fixed (double* answer_array_pointer = answer_array)
                SecondMethod_C(3, matrix_array, right_array, answer_array_pointer);

        Console.Write("Ответ ");
        for (int i = 0; i < answer_array.Length; i++)
            Console.Write("{0:F3}" + (i < answer_array.Length - 1 ? "; " : ""), answer_array[i]);
        Console.WriteLine(")");
        Console.WriteLine();

        Console.WriteLine(fileName);
        TimeList timeList_2 = TimeList.Load(fileName);
        Console.WriteLine(timeList_2.ToString());


    }

    static double FirstMethod(int n, int k)
    {
```

```
            Stopwatch timer = new Stopwatch();
            timer.Restart();
            Matrix matrix = new Matrix(n);
            double[] right = new double[n];
            for (int i = 0; i < n; i++)
                right[i] = (i + 1) * 10;
            for (int i = 0; i < k; i++)
                matrix.Solve(right);
            timer.Stop();
            return timer.Elapsed.TotalSeconds;
        }
    }
}
```