# Обектно-ориентирано Програмиране в C#

Цел на упражнението

Това упражнение запознава студентите с основни понятия в ООП в C#. Разширява познанията им в областта на ООП относно създаване на клас, обект, атрибут, метод, интерфейс, структура, както и работа с колекции (ArrayList, List<T>, LinkedList<T>, Queue<T>, Stack<T>, Dictionary<T, V>).

**1. Задачи за изпълнение**

**1.1. Клас.** Съвкупност от променливи и методи. Служи за представяне на реални обекти от света.

**Задача 1:** Създаване на клас

class MyCar

{

int speed;

string model;

public void ShowInfo();

}

**1.2. Обект.** В контекста на обектно-ориентираното програмиране, под обект се разбира съвкупност от елементи и тяхното поведение.

**Задача 2:** Създаване на обект

using System;

public class Car

{

public string color = "red";

}

Car frmCar = new Car();

private void button1\_Click(object sender, EventArgs e)

{

MessageBox.Show(frmCar.color+"");

}

**1.3. Атрибут.** Това е свойство на класа.

**Задача 3:** Създаване на атрибут

using System;

class Model

{

public string name;

}

Model m1 = new Model();

private void button1\_Click(object sender, EventArgs e)

{

m1.name = "Ferrari";

MessageBox.Show(m1.name + "");

}

**1.4. Метод.** Това е част от една програма, която решава конкретен проблем.

**Задача 4:** Създаване на метод в C#

using System;

Rectangle c = new Rectangle();

c.SetSize(2, 3);

private void button1\_Click(object sender, EventArgs e)

{

MessageBox.Show(c.Area().ToString());

}

class Rectangle

{

private int width;

private int height;

public void SetSize(int width, int height)

{

this.width = width;

this.height = height;

}

public double Area()

{

return this.width \*this.height \* Math.PI;

}

}

**1.5. Модификатор**. Това са запазени думи в обектно-ориентираното програмиране, които задават правата за достъп до класове, методи и други елементи на кода.

Таблица 10.1. Модификатори в C#

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | ***Class*** | ***Current assembly*** | ***Derived types*** | ***Derived types in current assembly*** | ***Entire program*** |
| public | + | + | + | + | + |
| protected | + | o | + | + | o |
| internal | + | + | o | O | o |
| private | + | o | o | O | o |
| protected internal | + | + | + | + | o |
| private protected | + | o | o | + | o |

**Задача 5:** Модификатор за достъп

using System;

Rectangle c = new Rectangle();

class Rectangle

{

private int width;

private int height;

public void SetSize(int width, int height)

{

this.width = width;

this.height = height;

}

public double Area()

{

return this.width \*this.height \* Math.PI;

}

}

private void button1\_Click(object sender, EventArgs e)

{

c.SetSize(2, 3);

MessageBox.Show(c.Area().ToString());

}

**1.6. Наследяване.** Класът наследник взема всички свойства и методи на класа-родител.

**Задача 6:** Наследяване в C#

using System;

Student student = new Student();

class Base

{

public string name = "Base";

protected int id = 1;

private bool isDefined = true;

}

class Student : Base

{

public void info()

{

MessageBox.Show("This is Student class");

MessageBox.Show("Members inherited");

MessageBox.Show(this.name);

MessageBox.Show(this.id.ToString());

MessageBox.Show("this.isDefined" + " is a field that cannot be displayed because it is private");

}

}

private void button1\_Click(object sender, EventArgs e)

{

student.info();

}

**1.7. Конструктор.** Блок от инструкции, който се изпълнява за инициализация на един обект при неговото създаване.

**Задача 7:** Конструктор в C#

using System;

Car car= new Car();

Car car1=new Car("Ferrari");

class Car

{

public Car()

{

MessageBox.Show("Car is created");

}

public Car(string car1)

{

MessageBox.Show("Car {car1} is created");

}

}

**Задача 8:** Верижно извикване на конструктори

using System;

class Rectangle

{

public Rectangle(int width, int height)

{

MessageBox.Show("Rectangle w={"+ width +"}, h={"+ height +"} is created");

}

public Rectangle() : this(1, 1) { }

}

private void button1\_Click(object sender, EventArgs e)

{

new Rectangle(2, 3);

new Rectangle();

}

}

**1.8. Метод Object.ToString.** Това е основният метод за форматиране в .NET Framework. Той преобразува обект в неговото низово представяне, така че да е подходящ за показване.

**Задача 9:** Метод ToString

using System;

class Car

{

public override string ToString()

{

return "This is Car class";

}

}

private void button1\_Click(object sender, EventArgs e)

{

Car c = new Car();

Object o = new Object();

MessageBox.Show(o.ToString());

MessageBox.Show(c.ToString());

MessageBox.Show(c+"");

}

**1.9. Инициализация на обект.** Преди да се използват елементите на класа, трябва да се инициализират променливите с указател към обект.

**Задача 10:** Инициализация на обект

using System;

class Car

{

public Car() {}

public string Model { set; get; }

public string Version { set; get; }

public override string ToString()

{

return Model+" is a " + Version;

}

}

private void button1\_Click(object sender, EventArgs e)

{

Car u = new Car { Model = "Ferrari", Version = " Ferrari Sergio" };

MessageBox.Show(u + " " );

}

**1.10. Статичен метод**

**Задача 11:** Статичен метод

using System;

namespace StaticMethod{

class User

{

static int Id = 100;

public static void ShowInfo()

{

MessageBox.Show ("This is User class");

MessageBox.Show ("The Id is: {"+Id+"}");

}

}

private void button1\_Click(object sender, EventArgs e)

{

User.ShowInfo();

}

**}**

**1.11. Структура.** Структурите представляват съвкупност от полета с данни. Приличат много на класовете, но са типове по стойност. Най-често се разполагат в стека. Предават се по стойност. Унищожават се при излизане от обхват. За разлика от структурите класовете са типове по референция и се разполагат в динамичната памет. Създаването и унищожаването им е по-бавно. При правилна употреба заместването на класове със структури може значително да увеличи производителността. Не се препоръчва в структурите да има методи с логика - трябва да съдържат само данни.

**Задача 12:**

using System;

namespace StuctUser

{

struct User

{

public string FirstName;

public string LastName;

public int Age;

public void DisplayInfo()

{

MessageBox.Show("FirstName:"+ FirstName+ " LastName: "+LastName+" Age: "+Age);

}

}

private void button1\_Click(object sender, EventArgs e)

{

User user;

user.FirstName = "Lilian";

user.LastName = "Petrov";

user.Age = 20;

user.DisplayInfo();

}

}

**1.12. Интерфейс IComparable.** Този интерфейс е реализиран от типове, чиито стойности могат да бъдат подредени или сортирани. Той изисква типовете за внедряване да дефинират един метод, CompareTo (Object), който показва дали позицията на текущия екземпляр в реда за сортиране е преди, след или на същата позиция, като на втория обект от същия тип.

**Задача 13:** Реализиране на интерфейса IComparable

using System;

using System.Collections.Generic;

namespace CompareBikesEx

{

class Bikes : IComparable<Bikes>

{

public string Name { get; set; }

public int Price { get; set; }

public int CompareTo(Bikes other)

{

if (this.Price < other.Price)

{

return 1;

}

else if (this.Price > other.Price)

{

return -1;

}

else

{

return 0;

}

}

public override string ToString()

{

return "Name: " + this.Name + " Price: " + this.Price;

}

}

private void button1\_Click(object sender, EventArgs e)

{

int count = 0;

var bikes = new List<Bikes>();

bikes.Add(new Bikes() { Name = "yamaha", Price = 900 });

bikes.Add(new Bikes() { Name = "honda", Price = 800 });

bikes.Add(new Bikes() { Name = "hero", Price = 700 });

bikes.Add(new Bikes() { Name = "pulsar", Price = 600 });

bikes.Add(new Bikes() { Name = "suzuki", Price = 500 });

bikes.Sort();

foreach (var element in bikes)

{

MessageBox.Show("Element "+count+" "+element);

count++;

}

MessageBox.Show("Number of elements: "+count);

}

}

**1.13. Колекции:** ArrayList, List<T>, LinkedList<T>, Queue<T>, Stack<T>, Dictionary<T, V>

**Задача 14:** ArrayList

sing System;

using System.Collections;

namespace Collections

{

class \_ArrayList

{

ArrayList list = new ArrayList();

private void button1\_Click(object sender, EventArgs e)

{

list.Add(0.5); // добавя се в списък обект от тип double

list.Add(1); // добавя се в списък обект от тип int

list.AddRange(new string[] { "Hi", "!" }); // добавя се в списък обект от тип string

// итериране

foreach (object o in list)

{

listBox1.Items.Add(o + "");

}

// изтриване на първия елемент

list.RemoveAt(0);

// обръщане на елементите от списъка

list.Reverse();

// получаване на елемент по индекс

listBox1.Items.Add(list[0]);

// итерация върху стойности

for (int i = 0; i < list.Count; i++)

{

listBox1.Items.Add(list[i]);

}

}

}

}

**Задача 15:** List<T>

using System;

using System.Collections.Generic;

namespace Collections

{

class \_List

{

List<int> numbers = new List<int>() { 1, 2, 3, 100 };

class Student

{

public string Name { get; set; }

}

private void button1\_Click(object sender, EventArgs e){

numbers.Add(6); // добавяне на елемент

numbers.AddRange(new int[] { 7, 8, 9 });

numbers.Insert(0, 99); // поставяне на числото 99 на първо място в списъка

numbers.RemoveAt(1); // премахване на втори елемент

foreach (int i in numbers)

{

listBox1.Items.Add(i.ToString());

}

List<Student> student = new List<Student>(3);

student.Add(new Student() { Name = "Tod" });

student.Add(new Student() { Name = "Bill" });

foreach (Student s in student)

{

listBox1.Items.Add(s.Name);

}

}

}

**Задача 16:** LinkedList<T>

using System;

using System.Collections.Generic;

namespace Collections

{

class \_LinkedList

{

LinkedList<int> numbers = new LinkedList<int>();

class Person

{

public string Name { get; set; }

}

private void button1\_Click(object sender, EventArgs e){

numbers.AddLast(10); // вмъква се възел със стойност 10 на последно място

//поради липса на възли, последния става първи

numbers.AddFirst(20); // вмъква се възел със стойност 20 на първо място

numbers.AddAfter(numbers.Last, 30); // вмъква се след последния възел нов възел със стойност 30

// списъка от възли има следната последователност: 20, 10, 30

foreach (int i in numbers)

{

listBox1.Items.Add(i);

}

LinkedList<Person> persons = new LinkedList<Person>();

// добавя се person към списъка и се взема обект LinkedListNode<Person>, в който се съхранява името на Tod

LinkedListNode<Person> tod = persons.AddLast(new Person() { Name = "Tod" });

persons.AddLast(new Person() { Name = "Jack" });

persons.AddFirst(new Person() { Name = "Joe" });

listBox1.Items.Add(tod.Previous.Value.Name); // връща предишната стойност на възел

listBox1.Items.Add(tod.Next.Value.Name); // връща следващата стойност на възел

}

}

**Задача 17:** Queue<T>

using System;

using System.Collections.Generic;

namespace Collections

{

class \_Queue

{

Queue<int> numbers = new Queue<int>();

class Person

{

public string Name { get; set; }

}

private void button1\_Click(object sender, EventArgs e){

numbers.Enqueue(3); // опашка 3

numbers.Enqueue(5); // опашка 3, 5

numbers.Enqueue(8); // опашка 3, 5, 8

//получава се първия елемент от опашката

int queueElement = numbers.Dequeue(); //обръшане на 5 и 8

listBox1.Items.Add(queueElement);

Queue<Person> persons = new Queue<Person>();

persons.Enqueue(new Person() { Name = "Tod" });

persons.Enqueue(new Person() { Name = "Jack" });

persons.Enqueue(new Person() { Name = "John" });

// получава се първият елемент без да се извлича

Person pp = persons.Peek();

listBox1.Items.Add(pp.Name);

listBox1.Items.Add("В опашката има "+ persons.Count +" души" );

// В опашката са Tod, Jack, John

foreach (Person p in persons)

{

listBox1.Items.Add(p.Name);

}

// Извличане на първият елемент от опашката - Tod

Person person = persons.Dequeue(); // В опашката са Jack, John

listBox1.Items.Add(person.Name);

}

}

**Задача 18:** Stack<T>

using System;

using System.Collections.Generic;

namespace Collections

{

class \_Stack

{

Stack<int> numbers = new Stack<int>();

class Person

{

public string Name { get; set; }

}

private void button1\_Click(object sender, EventArgs e){

numbers.Push(3); // добавен е в стека 3

numbers.Push(5); // добавени са в стека 5, 3

numbers.Push(8); // добавени са в стека 8, 5, 3

//тъй като 8 е последният въведен елемент, той се появява

int stackElement = numbers.Pop(); // в стека са 5, 3

listBox1.Items.Add(stackElement);

Stack<Person> persons = new Stack<Person>();

persons.Push(new Person() { Name = "Tod" });

persons.Push(new Person() { Name = "Jack" });

persons.Push(new Person() { Name = "John" });

foreach (Person p in persons)

{

listBox1.Items.Add(p.Name);

}

// първият елемент в стека

Person person = persons.Pop(); // в стека са Jack, John

listBox1.Items.Add(person.Name);

}

}

**Задача 19:** Dictionary<T, V>

Dictionary<int, string> countries = new Dictionary<int, string>(5);

private void button1\_Click(object sender, EventArgs e)

{

countries.Add(1, "Bulgaria");

countries.Add(3, "Malta");

countries.Add(2, "Belgium");

countries.Add(4, "Bolivia");

countries.Add(5, "Cuba");

foreach (KeyValuePair<int, string> keyValue in countries)

{

listBox1.Items.Add(keyValue.Key + " - " + keyValue.Value);

}

// получаване на елемент по ключ

string country = countries[4];

//промяна на обекта

countries[4] = "Spain";

//изтриване по ключ

countries.Remove(2);

listBox1.Items.Add("\n\tСлед промяната\n\t");

foreach (KeyValuePair<int, string> keyValue in countries)

{

listBox1.Items.Add(keyValue.Key + " - " + keyValue.Value);

}

}

Задача 1. *Разработете самостоятелно C# програма, която да декларира 3 класа: базов клас описващ професия, с име profession и класове teacher и doctor (учител и лекар), които да наследяват клас професия. Базовият клас съдържа полета, които се инициализират и достъпват чрез свойства (за четене и запис): „област”, „специалност”, „образование”, „заплата“. Базовият клас съдържа също метод: за пресмятане на ставка, т.е. заплащане за 1 час (брой работни часове е параметър, който се подава отвън). Класът учител съдържа допълнително данни (полета) за предмет и клас, които се инициализират и достъпват чрез свойства. Класът лекар съдържа 2 полета, указващи пациент, амболаторен лист и рецепта, които се също се инициализират и достъпват чрез свойства. Създайте код, който да използват създадените класове. Да се създаде меню за избор на транспортно средство и да се създадт обекти от съответния клас.*

# създаване на Unit Test в C#

**1. Тестване на софтуер – TDD и Unit тест**

* **TDD тест**

Съществуват различни подходи за тестване на софтуер, един от тях е Test Driven Development (TDD) или разработка чрез тестове. Този подход за тестване е създаден от Кент Бек през 2003, като основната му цел е да опрости дизайна. Характерно за този подход е, че първо се пишат тестови варианти, които да съответстват на изходния програмен код, който трябва да покрият.

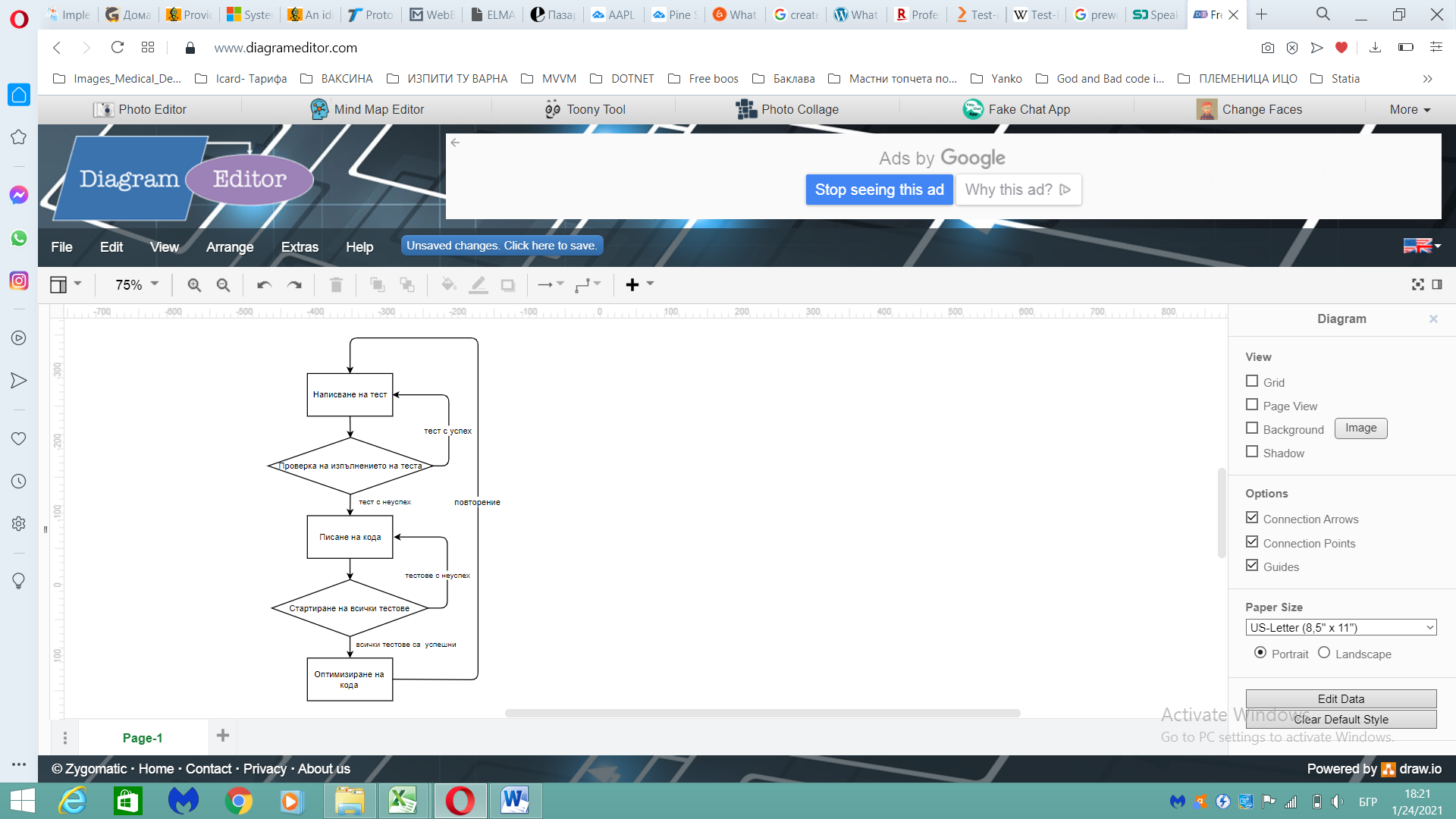
**1.1. Недостатъци на TDD**

* Трудно се прилагат при програми използващи БД или такива, които зависят от специфични, мрежови настройки;
* Недобре написани тестове обикновено завършват с неуспех;
* Добре е архитектурата, дизайна и стратегията на тестване да са добре обмислени, за да няма нежелани последствия от тестването. Промяна в теста на по-късен етап е трудна;
* Възможност за припокриване на тестовете по различни причини, като недобре написани тестове, поради недобро познаване на стратегията на TDD;
* Тестовете може да бъдат изтрити по грешка в процеса на разработка;
* Писането на код става непосредствено след създаване на модулните тестове и е желателно те да се пишат от едни и същи програмисти. Това от своя страна ще гарантира, коректна проверка на входните данни;
* При голям брой успешно проведени тестове, има вероятност да се стигне до погрешно констатиране на сигурността, а така и до недостатъчна реакция от AQ екипа.

**1.2. Предимства на TDD**

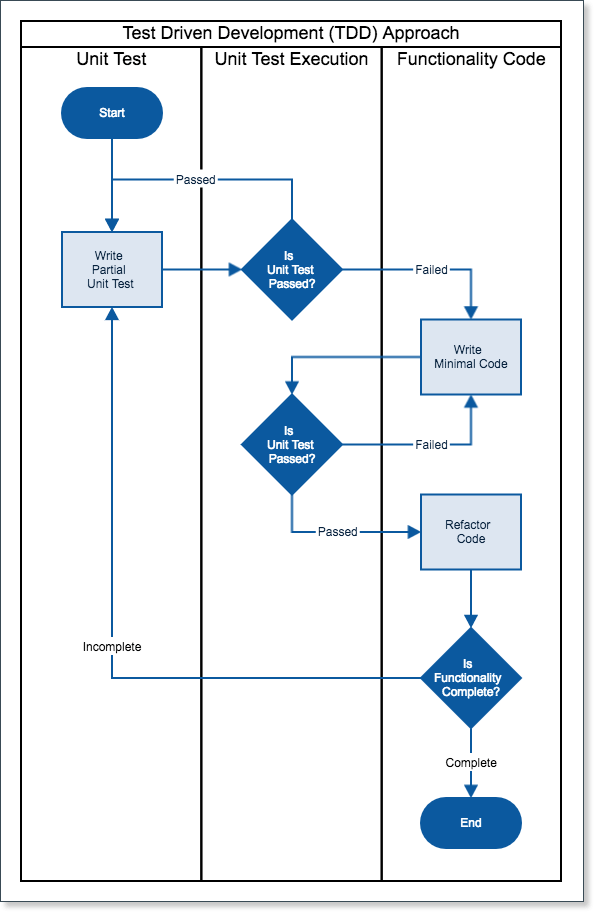
* Опростяване на кода;
* При по-сложен дизайн, например използване на СШП, те трябва да се напишат съобразно тях, като генерират този дизайн.

**1.3. Работен процес и описание на TDD**



**Фиг.11.1.** Работен процес на метода TDD

* Запис на единичен частичен тест.
* Изпълнение на модулния тест с цел да бъде счупен.
* Написване на минимален код за функционалността за тестване на единичен частичен тест.
* Стартиране на модулни тест:
* Ако e успешен, рефакториране на кода без да се променя поведението и без да се провали теста отново.
* Ако не е успешен преминаване към т.3 за да пренаписване/ минимална промяна на кода

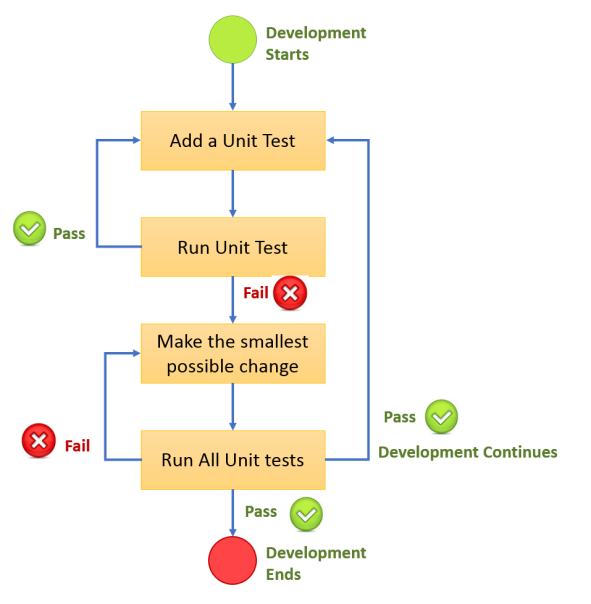
**[](https://i2.wp.com/novicedeveloper.com/wp-content/uploads/2018/11/Test-Driven-Development-TDD-Approach.png?ssl=1)**

**Фиг.11.2.** Test-Driven Development (TDD) Approach – Process Flow Diagram

Test Driven Development (TDD) е концепция на TDD за написване и корекция на неуспешни тестове, преди да написване на същинския код (преди разработката). Това помага да се избегне дублиране на код, тъй като се пише малко количество код наведнъж, за да се преминат тестовете. TDD е процес на разработване и стартиране на автоматизиран тест преди реалното развитие на приложението. TDD понякога се нарича още Test First Development.

* **Unit Тест**

Друг вид тестване е Unit теста. Този тест спада към функционалните тестове. Основната му идея е да се напишат тестове, които упражняват възможно най-малката „единица“ код. Единичните тестове обикновено се пишат на същия език за програмиране като изходния код на самото приложение и се пишат, за да се използва директно този код. Всъшност Unit тестовете най-общо казано са код, който тества друг код.



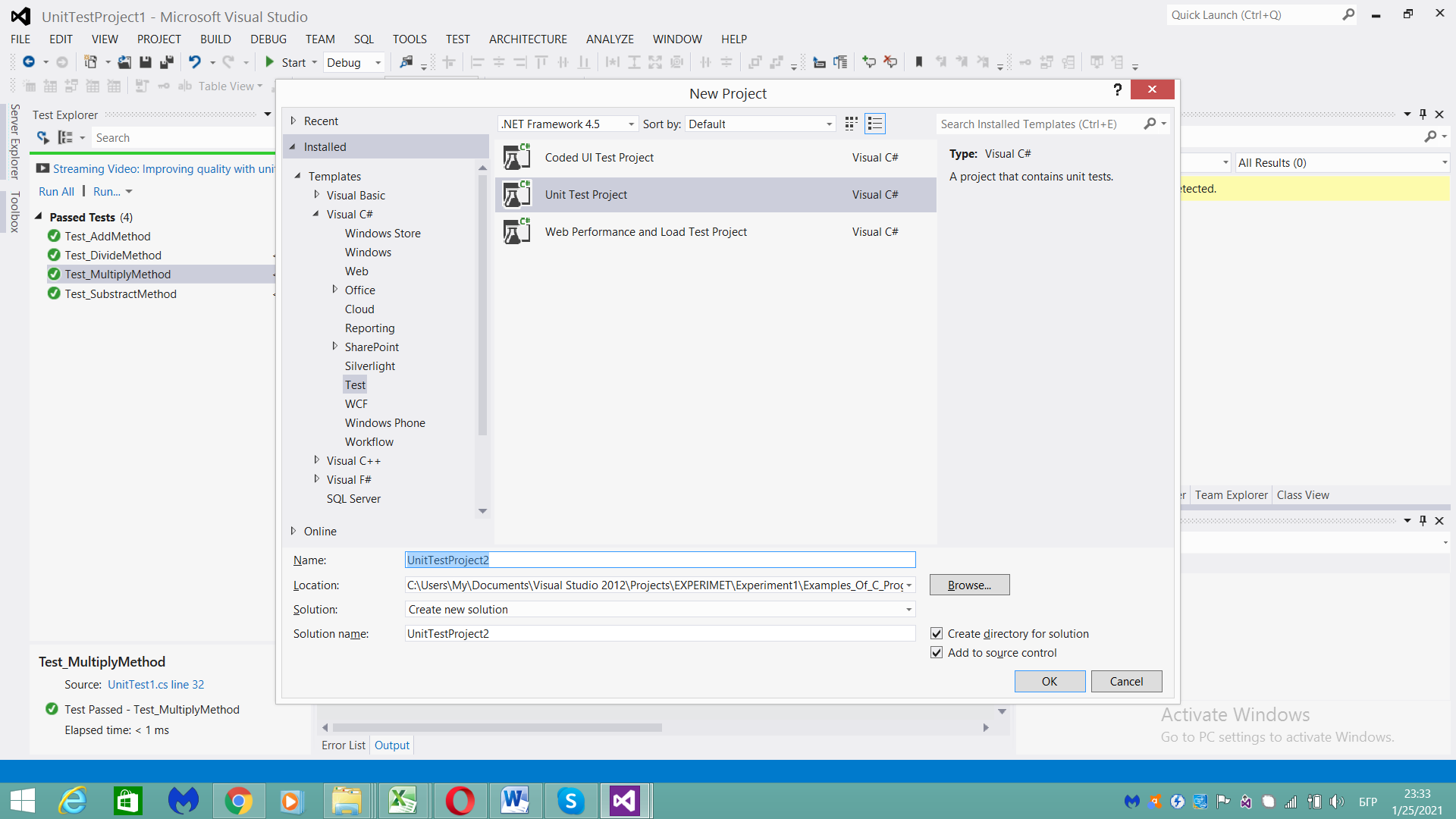
**Фиг.11.3.** Диаграма на работния процес на Unit тест

**2. Задачи за изпълнение:**

Задача 1. *Да се създаде Unit Test за тестване на 4 функции свързани с пресмятане на 4 те основни аритметични операции (събиране, изваждане, умножение и деление) на две променливи. Стъпките за изпълнение на задачата са 3: създаване на Unit Test Project, създаване на клас BasicMaths и тестване на създадения Unit Test.*

Постъпково примерно решение:

1.Създаване на нов проект. Избор на Test/ Unit Test Project (фиг. 11.4)



**Фиг.11.4.** Екран за избор на Unit Test Project

2. Написване на кода за тестване на всяка от 4-те функции

using System;

using Microsoft.VisualStudio.TestTools.UnitTesting;

namespace UnitTestProject1

{

[TestClass]

public class UnitTest1

{

[TestMethod]

public void Test\_AddMethod()

{

BasicMaths bm = new BasicMaths();

double res = bm.Add(10, 10);

Assert.AreEqual(res, 20);

}

[TestMethod]

public void Test\_SubstractMethod()

{

BasicMaths bm = new BasicMaths();

double res = bm.Substract(10, 10);

Assert.AreEqual(res, 0);

}

[TestMethod]

public void Test\_DivideMethod()

{

BasicMaths bm = new BasicMaths();

double res = bm.Divide(10, 5);

Assert.AreEqual(res, 2);

}

[TestMethod]

public void Test\_MultiplyMethod()

{

BasicMaths bm = new BasicMaths();

double res = bm.Multiply(2, 10);

Assert.AreEqual(res, 20);

}

}

}

3. Добавяне на клас BasicMaths включващ функции – събиране, изваждане, умножение и деление.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace UnitTestProject1

{

public class BasicMaths

{

public double Add(double num1, double num2)

{

return num1 + num2;

}

public double Substract(double num1, double num2)

{

return num1 - num2;

}

public double Divide(double num1, double num2)

{

return num1 / num2;

}

public double Multiply(double num1, double num2)

{

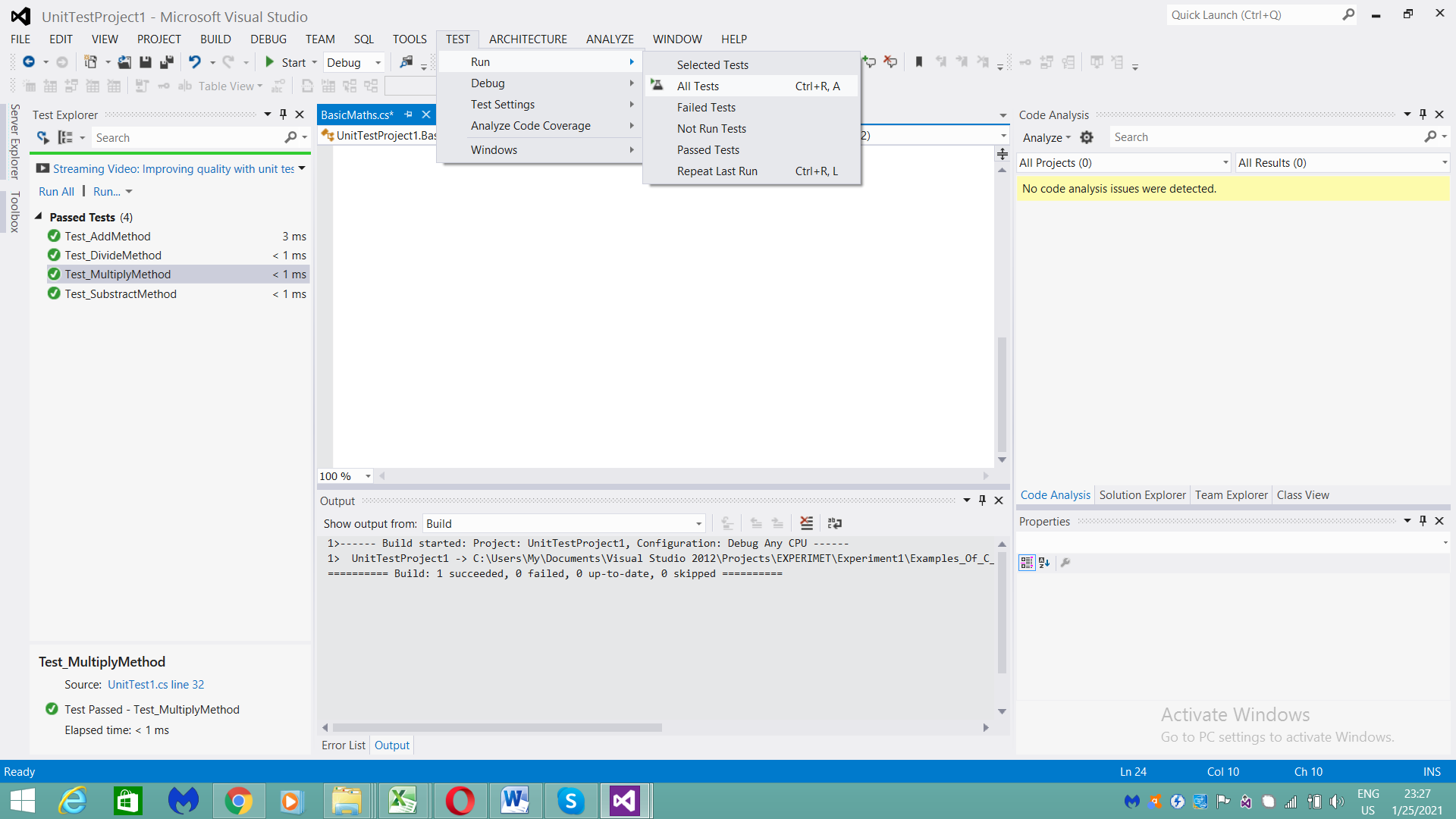
return num1 \* num2;

}

}

}

4. Стартиране на Unit Test



**Фиг.11.5.** Екран за избор за стартиране на Unit Test

5. Анализ на направения тест

От панела Test Explorer от фиг.11.5. се вижда, че Unit теста е успешен, това е обузначено със зелен цвят, като е добавено и времето за изпълнение на теста за всяка отделана функция.

При неуспешен тест е необходимо, той да се коригира и отново да се изпълни.

2. Задания за самостоятелна работа

Задача 1. Да се създаде Unit Test към задача Квадратно уравнение.

Задача 2. Да се създаде Unit Test към задача Ваксинация за валидиране на user name и password (чрез регулярни изрази). Изискванията за валидиране към user name: от 5 до 8 символа, които включват цифри от 6 до 9, малки, латински букви, като задължително първият символ в низа да не е цифра. Изискванията за валидиране към password: допускат се точно 4 символа, като посленият от тях да е задължително символа „?“.