

ΕΡΓΑΣΙΑ ΣΤΗ ΠΟΙΟΤΗΤΑ ΚΑΙ ΑΞΙΟΠΙΣΤΙΑ ΛΟΓΙΣΜΙΚΟΥ

Αναστασία Κατσίπη - ice20390099

Γεώργιος Δαναλάτος - ice20390045

Κωνσταντίνος Καρτσαγκούλης Δασκαλάκης- ice20390095

**1η Συνάρτηση Υλοποίησης**

**Α. DLL**

public bool ValidName(string Name)

{

// Check if the name is not null or empty

if (string.IsNullOrEmpty(Name))

{

return false;

}

// Check if the name contains only letters and spaces

if (!Regex.IsMatch(Name, @"^[A-Za-z\s]+$"))

{

return false;

}

return true;

}

**B. Δημιουργία Περιπτώσεων Ελέγχου (TestCases)**

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| **ID** | **ΣΥΝΘΗΚΗ** | **ΕΓΚΥΡΗ** | **ΑΚΥΡΗ** |
| **1** | **Το string να περιέχει μόνο γράμματα.** | **Regex.IsMatch(Name, @"^[A-Za-z\s]+$")** | **!Regex.IsMatch(Name, @"^[A-Za-z\s]+$")** |
| **2** | **Το string να περιέχει πάνω από μία λέξη.** | **wordCount >= 2** | **wordCount < 2** |
| **3** | **Να μην είναι κενό.** | **!string.IsNullOrEmpty(Name)** | **string.IsNullOrEmpty(Name)** |

|  |  |  |
| --- | --- | --- |
|  |  |  |
| **ID** | **Δοκιμαστικά Δεδομένα Εισόδου** | **Αναμενόμενα Αποτελέσματα** |
| **1** | **John Doe** | **true** |
| **2** | **null** | **false** |
| **3** | **John123** | **false** |
| **4** | **John** | **false** |
| **5** | **Maria Dollores Ermoza** | **true** |

**Γ. Ανάπτυξη Ελέγχων Μονάδων (UnitTests)**

HRLibSDLL.HRLibSDLL hl = new HRLibSDLL.HRLibSDLL();

object[,] testcases =

{

{ 1, "John Doe", true, "Valid name" },

{ 2, null, false, "Null input" },

{ 3, "John123", false, "Invalid characters" },

{ 4, "John", false, "One word" },

{ 5, "Maria Dollores Ermoza", true, "Valid Name" },};

// Act & Assert

for (int i = 0; i < testcases.GetLength(0); i++)

{

try

{

// Arrange

string input = (string)testcases[i, 1];

bool expectedResult = (bool)testcases[i, 2];

// Act

bool result = hl.ValidName(input);

// Assert

Assert.AreEqual(expectedResult, result, $"Test Case {testcases[i, 0]} failed. Reason: {testcases[i, 3]}");

}

catch (AssertFailedException e)

{

// Log failure

Console.WriteLine($"Failed Test Case: {testcases[i, 0]} - {e.Message}");

throw; // Re-throw exception to mark the test as failed

}

}

**Δ. Αναφορές ελέγχου**

1. Αναγνωριστικό Σφάλματος (αποτελούμενο από ένα λεκτικό αναγνωριστικό του σφάλματος και έναν αύξοντα αριθμό)

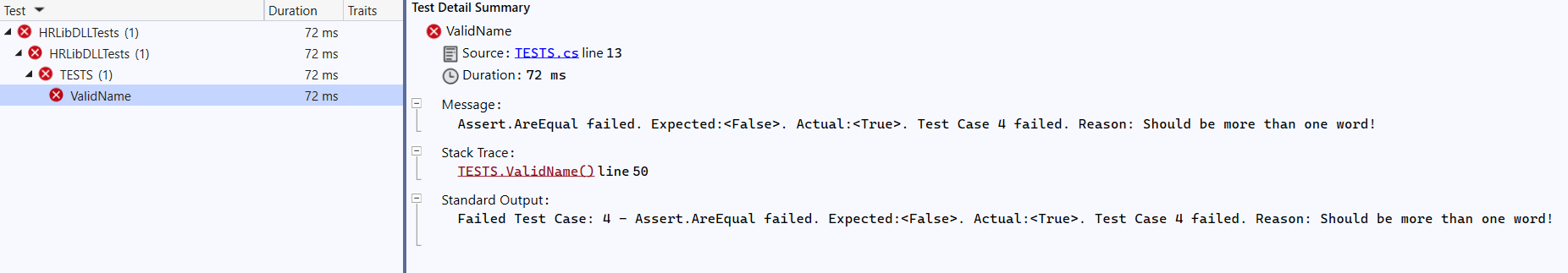
**Assert.AreEqual failed. Expected:<False>. Actual:<True>. Test Case 4 failed. Reason: Should be more than one word!**

2. Όνομα μεθόδους ελέγχου (test method) που απέτυχε: ValidName()

3. Περιγραφή ελέγχου που πραγματοποιήθηκε (δηλαδή της test method): Έγινε έλεγχος για το εάν το string είναι ένα έγκυρο όνομα υπαλλήλου.

4. Αναφορά σφάλματος που παρουσιάστηκε: Εφόσον τα ονόματα είναι 2 ή παραπάνω λέξεις δεν μπορεί να είναι λιγότερο από δύο. Άρα το “Jhon” πρέπει να έχει ως αποτέλεσμα false ενώ εγώ έχω πάρει true.

5. Στιγμιότυπα οθόνης (screenshots) του Test Explorer



**2η Συνάρτηση Υλοποίησης**

**Α. DLL**

//Β.2.β

char[] symbols = { '@', '#', '$', '%', '^', '&', '\*', '(', ')', '\_', '+', '-', '=', '{', '}', '[', ']', ';', ':', '<', '>', ',', '.' };

bool hasSymbol = symbols.Any(Password.Contains);

bool hasUppercase = Password.Any(char.IsUpper);

bool hasLowercase = Password.Any(char.IsLower);

bool hasDigit = Password.Any(char.IsDigit);

//Β.2.γ

foreach (char c in Password)

{

if (char.IsLetter(c) && !((c >= 'A' && c <= 'Z') || (c >= 'a' && c <= 'z')))

{

return false;

}

}

if (!(char.IsUpper(Password[0]) && (char.IsLetter(Password[0]))))

{

return false;

}

if (!char.IsDigit(Password[Password.Length - 1]))

{

return false;

}

return hasSymbol && hasUppercase && hasLowercase&& hasDigit;

}

**B. Δημιουργία Περιπτώσεων Ελέγχου (TestCases)**

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| **ID** | **ΣΥΝΘΗΚΗ** | **ΕΓΚΥΡΗ** | **ΑΚΥΡΗ** |
| **1** | Τουλάχιστον 12 χαρακτήρες**.** | **Password.Length >= 12** | **Password.Length < 12** |
| **2** | Συνδυασμός κεφαλαίων γραμμάτων, πεζών γραμμάτων, αριθμών και συμβόλων | **If password contain at least one special character.** | **If password doesnt contain not even one special characters.** |
| **3** | Τα γράμματα να είναι λατινικοί χαρακτήρες | **All characters shoulds be latin.** | **If password contains at least one character that is not latin** |
| **4** | Να ξεκινάει από κεφαλαίο γράμμα και να τελειώνει με αριθμό | **Should start with an uppercase letter and end wih a number** | **If it doesnt Should start with an uppercase letter and end wih a number** |

|  |  |  |
| --- | --- | --- |
|  |  |  |
| **ID** | **Δοκιμαστικά Δεδομένα Εισόδου** | **Αναμενόμενα Αποτελέσματα** |
| **1** | **SDFGHgfyrsFF\_43566789gfdstjf3** | **true** |
| **2** | **sDFGyrsFF\_3566789gfdstjf3** | **false** |
| **3** | **SDFGHgμsFF\_43566789gfdstjf3** | **false** |
| **4** | **SDFGHgfyrsFF\_43566789gfdstjf** | **false** |
| **5** | **SDFGHgfyrsFF43566789gfdstjf3** | **false** |

**Γ. Ανάπτυξη Ελέγχων Μονάδων (UnitTests)**

[TestMethod]

public void ValidPassword()

{

//Έλεγχος ονομάτων

//Δημιουργία ενός αντικειμένου της κλάσης του dll που θέλουμε να τεστάρουμε

HRLibSDLL.HRLibSDLL hl = new HRLibSDLL.HRLibSDLL();

//Δημιουργία Περιπτώσεων Ελέγχου

object[,] testcases =

{

{ 1, "SDFGHgfyrsFF\_43566789gfdstjf3", true, "Valid Password" },

{ 2, "sDFGyrsFF\_3566789gfdstjf3", false, "Starts with small character" },

{ 3, "SDFGHgμsFF\_43566789gfdstjf3", false, "Only latin characters accepted" },

{ 4, "SDFGHgμsFF\_43566789gfdstjf", false, "Doesnt end with number" },

{ 5, "SDFGHgμsFF43566789gfdstjf", false, "Doesnt have special character" },

};

// Act & Assert

for (int i = 0; i < testcases.GetLength(0); i++)

{

try

{

// Arrange

string input = (string)testcases[i, 1];

bool expectedResult = (bool)testcases[i, 2];

// Act

bool result = hl.ValidPassword(input);

// Assert

Assert.AreEqual(expectedResult, result, $"Test Case {testcases[i, 0]} failed. Reason: {testcases[i, 3]}");

}

catch (AssertFailedException e)

{

Console.WriteLine($"Failed Test Case: {testcases[i, 0]} - {e.Message}");

throw; // Re-throw exception to mark the test as failed

}

}

}

**Δ. Αναφορές ελέγχου**

1. Αναγνωριστικό Σφάλματος (αποτελούμενο από ένα λεκτικό αναγνωριστικό του σφάλματος και έναν αύξοντα αριθμό):

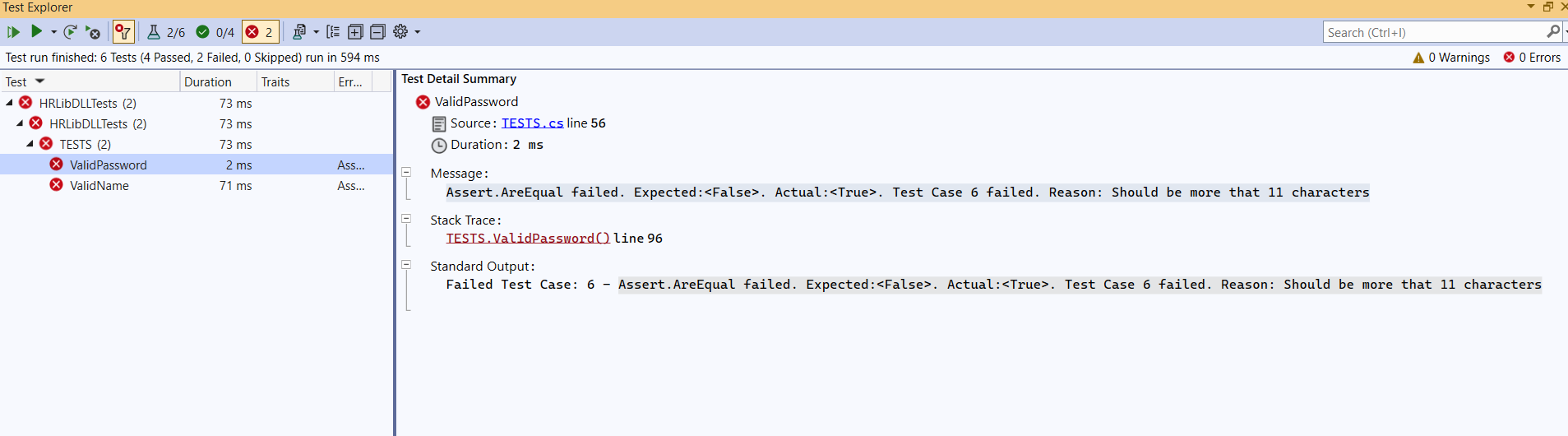
**Assert.AreEqual failed. Expected:<False>. Actual:<True>. Test Case 6 failed. Reason: Should be more that 11 characters**

2. Όνομα μεθόδους ελέγχου (test method) που απέτυχε: ValidPassword()

3. Περιγραφή ελέγχου που πραγματοποιήθηκε (δηλαδή της test method): Έγινε έλεγχος για το εάν o κωδικός περιέχει 12 ή παραπάνω χαρακτήρες.

4. Αναφορά σφάλματος που παρουσιάστηκε: Ο κωδικός ενώ δεν περιέχει 12 ή παραπάνω χαρακτήρες και περιμένουμε να είναι false παίρνουμε true.

5. Στιγμιότυπα οθόνης (screenshots) του Test Explorer:



**3η Συνάρτηση Υλοποίησης**

**Α. DLL**

public void EncryptPassword(string password, ref string encryptedPW)

{

int shift = 6;

StringBuilder EncryptedPassword = new StringBuilder();

foreach (char c in password)

{

char encryptedChar = c;

// Check if the character is an ASCII printable character (32 to 126)

if (c >= 32 && c <= 126)

{

// Apply the Caesar Cipher shift

encryptedChar = (char)(c + shift);

// Wrap around if the shift goes beyond printable ASCII characters

if (encryptedChar > 126)

{

encryptedChar = (char)(encryptedChar - 95);

}

}

EncryptedPassword.Append(encryptedChar);

}

encryptedPW = EncryptedPassword.ToString();

}

**B. Δημιουργία Περιπτώσεων Ελέγχου (TestCases)**

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| **ID** | **ΣΥΝΘΗΚΗ** | **ΕΓΚΥΡΗ** | **ΑΚΥΡΗ** |
| **1** | To password κρυπτογραφημέ**νο** με βάση τον Κώδικα του Καίσαρα (Caesar’s Cipher), με αλφάβητο το ASCII και ολίσθηση κατά 5 θέσεις | **Shift = 5** | **Shift διάφορο του 5** |

|  |  |  |
| --- | --- | --- |
|  |  |  |
| **ID** | **Δοκιμαστικά Δεδομένα Εισόδου** | **Αναμενόμενα Αποτελέσματα** |
| **1** | **ABCD10** | **FGHI65** |
| **2** | **abcd10** | **fghi65** |
| **3** | **abcd10** | **abcd10** |

Γ. Ανάπτυξη Ελέγχων Μονάδων (UnitTests)

[TestMethod]

public void EncryptPassword\_ShouldEncryptPasswords()

{

// Arrange

HRLibSDLL.HRLibSDLL hl = new HRLibSDLL.HRLibSDLL();

var testCases = new (int, string, string, string)[]

{

(1, "ABCD10", "FGHI65", "Correct encryption"),

(2, "abcd10", "fghi65", "Correct encryption"),

(3, "abcd10", "abcd10", "The password stays the same!"),

};

// Act & Assert

for (int i = 0; i < testCases.GetLength(0); i++)

{

try

{

// Arrange

string input = (string)testCases[i].Item2;

string expectedResult = (string)testCases[i].Item3;

// Act

string encryptedPassword = string.Empty;

hl.EncryptPassword(input, ref encryptedPassword);

// Assert

Assert.AreEqual(expectedResult, encryptedPassword);

}

catch (AssertFailedException e)

{

// Log failure

Console.WriteLine($"Failed Test Case: {testCases[i].Item1} - {e.Message}");

throw; // Re-throw exception to mark the test as failed

}

}

}

**Δ. Αναφορές ελέγχου**

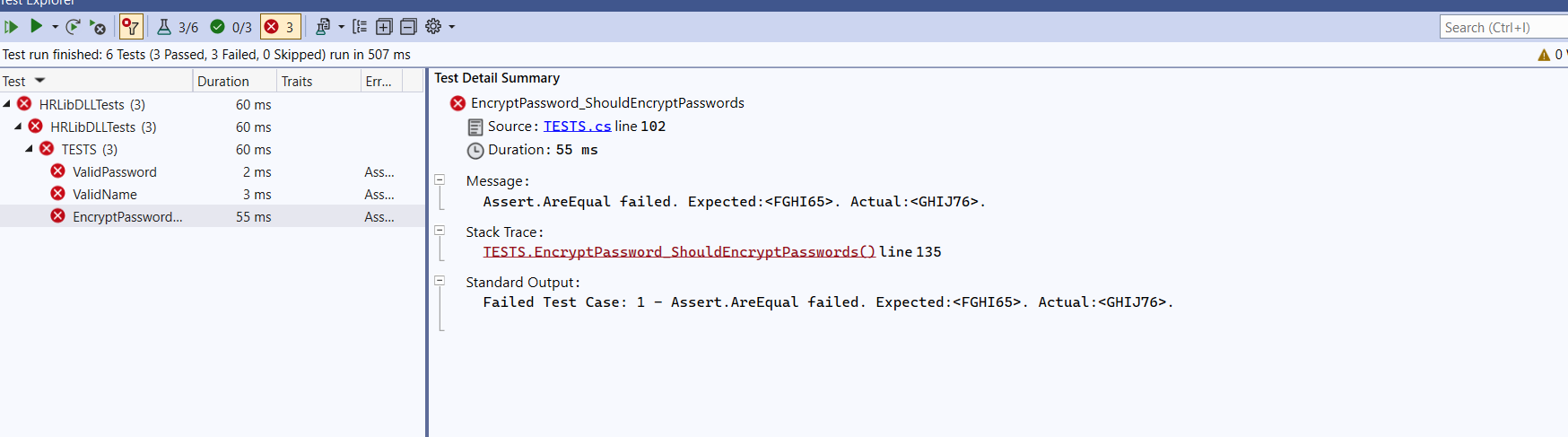
1. Αναγνωριστικό Σφάλματος (αποτελούμενο από ένα λεκτικό αναγνωριστικό του σφάλματος και έναν αύξοντα αριθμό):

**Failed Test Case: 1 - Assert.AreEqual failed. Expected:<FGHI65>. Actual:<GHIJ76>.**

2. Όνομα μεθόδους ελέγχου (test method) που απέτυχε: EncryptPassword\_ShouldEncryptPasswords()

3. Περιγραφή ελέγχου που πραγματοποιήθηκε (δηλαδή της test method): Έγινε έλεγχος για το η αποκρυπτογράφηση του κωδικού γίνεται σωστά.

4. Αναφορά σφάλματος που παρουσιάστηκε: Ο κωδικός δεν αποκρυπτογραφείται σωστά διότι δεν μας δίνεται το αναμενόμενο αποτέλεσμα.

5. Στιγμιότυπα οθόνης (screenshots) του Test Explorer:

**4η Συνάρτηση Υλοποίησης**

**Α. DLL**

**public void CheckPhone(string Phone, ref int TypePhone, ref string InfoPhone)**

**{**

**if (Phone.Length == 10)**

**{**

**string numberString = Phone.ToString();**

**string pattern1 = @"^2[0-9][0-9]{8}$";**

**Boolean HomeNumber = Regex.IsMatch(Phone, pattern1);**

**string peraeus = @"^21[0-9]{8}$";**

**string attica = @"^22[0-9]{8}$";**

**string centralMacedonia = @"^23[0-9]{8}$";**

**string thessalia\_WestMacedonia = @"^24[0-9]{8}$";**

**string thrakh\_EastMacedonia = @"^25[0-9]{8}$";**

**string Epirus\_WesternCentralGreece\_WesternPeloponnese\_IonianIslands = @"^26[0-9]{8}$";**

**string easternPeloponnese\_Kythira = @"^27[0-9]{8}$";**

**string crete = @"^28[0-9]{8}$";**

**bool isPeraeus = Regex.IsMatch(Phone, peraeus);**

**bool isAttica = Regex.IsMatch(Phone, attica);**

**bool isCentralMacedonia = Regex.IsMatch(Phone, centralMacedonia);**

**bool isThessalia\_WestMacedonia = Regex.IsMatch(Phone, thessalia\_WestMacedonia);**

**bool isThrakh\_EastMacedonia = Regex.IsMatch(Phone, thrakh\_EastMacedonia);**

**bool isEpirus\_WesternCentralGreece\_WesternPeloponnese\_IonianIslands = Regex.IsMatch(Phone, Epirus\_WesternCentralGreece\_WesternPeloponnese\_IonianIslands);**

**bool isEasternPeloponnese\_Kythira = Regex.IsMatch(Phone, easternPeloponnese\_Kythira);**

**bool isCrete = Regex.IsMatch(Phone, crete);**

**string novaPattern = @"^69[069][0-9]{7}$";**

**string vodafonePattern = @"^69[45][0-9]{7}$";**

**string cosmotePattern = @"^69[78][0-9]{7}$";**

**bool isNova = Regex.IsMatch(Phone, novaPattern);**

**bool isVodafone = Regex.IsMatch(Phone, vodafonePattern);**

**bool isCosmote = Regex.IsMatch(Phone, cosmotePattern);**

**if (HomeNumber)**

**{**

**TypePhone = 0;**

**Console.WriteLine("This is a home number.");**

**if (isPeraeus)**

**{**

**InfoPhone = "Ζώνη21.ΜητροπολιτικήΠεριοχήΑθήνας–Πειραιά";**

**}**

**else if (isAttica)**

**{**

**InfoPhone = "Ζώνη22.ΑνατολικήΣτερεάΕλλάδα,Αττική,ΝησιάΑιγαίου";**

**}**

**else if (isCentralMacedonia)**

**{**

**InfoPhone = "Ζώνη23.ΚεντρικήΜακεδονία";**

**}**

**else if (isThessalia\_WestMacedonia)**

**{**

**InfoPhone = "Ζώνη24.Θεσσαλία,ΔυτικήΜακεδονία";**

**}**

**else if (isThrakh\_EastMacedonia)**

**{**

**InfoPhone = "Ζώνη25.Θράκη,ΑνατολικήΜακεδονία";**

**}**

**else if (isEpirus\_WesternCentralGreece\_WesternPeloponnese\_IonianIslands)**

**{**

**InfoPhone = "Ζώνη26.Ήπειρος,ΔυτικήΣτερεάΕλλάδα,ΔυτικήΠελοπόννησος,Ιόνια Νησιά";**

**}**

**else if (isEasternPeloponnese\_Kythira)**

**{**

**Console.WriteLine("Ζώνη 27. Ανατολική Πελοπόννησος, Κύθηρα");**

**InfoPhone = "Ζώνη27.ΑνατολικήΠελοπόννησος,Κύθηρα";**

**}**

**else if (isCrete)**

**{**

**Console.WriteLine("Ζώνη28. Κρήτη");**

**InfoPhone = "Ζώνη 28.Κρήτη";**

**}**

**}**

**else if (isNova || isVodafone || isCosmote)**

**{**

**TypePhone = 1;**

**Console.WriteLine("This is a mobile number.");**

**if (isNova)**

**{**

**Console.WriteLine("NOVA");**

**InfoPhone = "NOVA";**

**}**

**else if (isVodafone)**

**{**

**Console.WriteLine("VODAFONE");**

**InfoPhone = "VODAFONE";**

**}**

**else if (isCosmote)**

**{**

**Console.WriteLine("COSMOTE");**

**InfoPhone = "COSMOTE";**

**}**

**}**

**else**

**{**

**TypePhone = -1;**

**InfoPhone = null;**

**Console.WriteLine("This is not a valid number.");**

**}**

**}**

**else**

**{**

**TypePhone = -1;**

**InfoPhone = null;**

**Console.WriteLine("Invalid phone number length.");**

**}**

**}**

**B. Δημιουργία Περιπτώσεων Ελέγχου (TestCases)**

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| **ID** | **ΣΥΝΘΗΚΗ** | **ΕΓΚΥΡΗ** | **ΑΚΥΡΗ** |
| **1** | **Εάν είναι ίσο με 10 αριθμους.** | **Είναι ίσο με 10.** | **Είναι διάφορο του 10 για παράδειγμα >10 ή 10<** |
| **2** | Εάν τα 2 πρώτα ψηφία είναι ίσα με 69 ή μεταξύ των αριθμών 20-29. | **Να ξεκινάει με 69,21,22,23,244,25,26,27,28** | **Να ξεκινάει με οποιοδήποτε άλλο αριθμό απο τα 69,21,22,23,244,25,26,27,28** |
| **3** | Εάν όλα τα ψηφία είναι αριθμοί | **Όλα τα ψηφία να είναι αριθμοί.** | **Κάποιο από τα ψηφία να μην είναι αριθμός.** |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| **ID** | **Δοκιμαστικά Δεδομένα Εισόδου** | **Αναμενόμενα Αποτελέσματα 1** | **Αναμενόμενα Αποτελέσματα 2** |
| **1** | **2104009764** | **0** | **Ζώνη21.ΜητροπολιτικήΠεριοχήΑθήνας–Πειραιά** |
| **2** | **2104764** | **-1** | **null** |
| **3** | **6947446196** | **1** | **VODAFONE** |
| **4** | **6977446196** | **1** | **COSMOTE** |
| **5** | **2304009764** | **O** | **Ζώνη23.ΚεντρικήΜακεδονία** |

Γ. Ανάπτυξη Ελέγχων Μονάδων (UnitTests)

public void CheckPhone\_should()

{

// Arrange

HRLibSDLL.HRLibSDLL hl = new HRLibSDLL.HRLibSDLL();

var testCases = new (int, string, int, string, string)[]

{

(1, "2104009764", 0, "Ζώνη21.ΜητροπολιτικήΠεριοχήΑθήνας–Πειραιά", "Correct number"),

(2, "2104764", -1, null, "Correct number"),

(3, "6947446196", 1, "VODAFONE", "Correct number"),

(4, "6977446196", 1, "COSMOTE", "Correct number"),

(5, "2304009764", 0, "Ζώνη23.ΚεντρικήΜακεδονία", "Correct number"),

};

// Act & Assert

for (int i = 0; i < testCases.GetLength(0); i++)

{

try

{

// Arrange

string input = (string)testCases[i].Item2;

int expectedTypePhone = (int)testCases[i].Item3;

string expectedInfoPhone = (string)testCases[i].Item4;

// Act

string infoPhone = string.Empty;

int typePhone = int.MinValue;

hl.CheckPhone(input, ref typePhone, ref infoPhone);

// Assert

Assert.AreEqual(expectedTypePhone, typePhone, $"Test Case {testCases[i].Item1} - Type of phone is not as expected");

Assert.AreEqual(expectedInfoPhone, infoPhone, $"Test Case {testCases[i].Item1} - Info about the phone is not as expected");

}

catch (AssertFailedException e)

{

// Log failure

Console.WriteLine($"Failed Test Case: {testCases[i].Item1} - {e.Message}");

throw; // Re-throw exception to mark the test as failed

}

}

}

**Δ. Αναφορές ελέγχου**

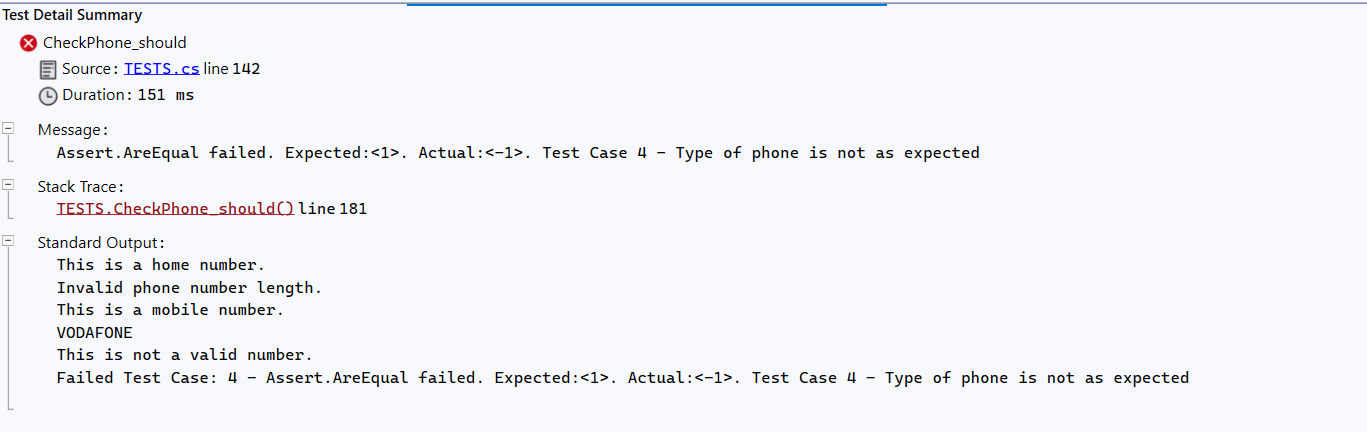
1. Αναγνωριστικό Σφάλματος (αποτελούμενο από ένα λεκτικό αναγνωριστικό του σφάλματος και έναν αύξοντα αριθμό):

**Failed Test Case: 4 - Assert.AreEqual failed. Expected:<1>. Actual:<-1>. Test Case 4 - Type of phone is not as expected**

2. Όνομα μεθόδους ελέγχου (test method) που απέτυχε: CheckPhone\_should

3. Περιγραφή ελέγχου που πραγματοποιήθηκε (δηλαδή της test method): Ελέγχω εάν ο αριθμός που ξεκινάει από 697 αντιστοιχίζεται σωστά στην τηλεφωνική εταιρεία COSMOTE!

4. Αναφορά σφάλματος που παρουσιάστηκε: Ενώ το νούμερο που ελέγχουμε ανήκει στην τηλεφωνική εταιρεία COSMOTE μας το παίρνει ως λάθος.

5. Στιγμιότυπα οθόνης (screenshots) του Test Explorer:

**5η Συνάρτηση Υλοποίησης**

**Α. DLL**

public void InfoEmployee(Employee EmplX, ref int Age, ref int YearsOfExperience)

{

DateTime currentDate = DateTime.Now;

Age = currentDate.Year - EmplX.Birthday.Year;

if (EmplX.Birthday.Date > currentDate.AddYears(-Age))

{

Age--;

}

YearsOfExperience = currentDate.Year - EmplX.HiringDate.Year;

if (EmplX.HiringDate.Date > currentDate.AddYears(-YearsOfExperience))

{

YearsOfExperience--;

}

}

**B. Δημιουργία Περιπτώσεων Ελέγχου (TestCases)**

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| **ID** | **ΣΥΝΘΗΚΗ** | **ΕΓΚΥΡΗ** | **ΑΚΥΡΗ** |
| **1** | **Η προϋπηρεσία μεγαλύτερη του 0** | **Προϋπηρεσία > 0** | **Προυπηρεσία<0** |
| **2** | Η ηλικία να είναι μεγαλύτερη ή ίση των 18. | **Ηλικία >=18** | **Ηλικία<18** |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Δοκιμαστικά Δεδομένα Εισόδου 1** | **Δοκιμαστικά Δεδομένα Εισόδου 2** | **Αναμενόμενα Αποτελέσματα 1** | **Αναμενόμενα Αποτελέσματα 2** |
| **ID** | **Ημερομηνία Γέννησης** | **Ημερομηνία Πρόσληψης** | **Ηλικία** | **Χρόνια προυπηρεσίας** |
| **1** | **new DateTime(1990, 1, 15)** | **new DateTime(2010, 3, 1)** | **34** | **14** |
| **2** | **new DateTime(1985, 5, 20)** | **new DateTime(2012, 8, 10)** | **38** | **11** |
| **3** | **new DateTime(1983, 2, 20)** | **new DateTime(2015, 8, 10)** | **41** | **8** |

[TestMethod]

public void InfoEmployee\_CalculateAgeAndExperience\_Success()

{

// Arrange

HRLibSDLL.HRLibSDLL library = new HRLibSDLL.HRLibSDLL();

var testCases = new (int, string, string, string, DateTime, DateTime, int, int)[]

{

(1, "John Doe", "1234567890", "9876543210", new DateTime(1990, 1, 15), new DateTime(2010, 3, 1), 34, 14),

(2, "Jane Smith", "5551234567", "9998887777", new DateTime(1985, 5, 20), new DateTime(2012, 8, 10), 38, 11),

(3, "Lana Del Rey", "5551262567", "9998882377", new DateTime(1983, 2, 20), new DateTime(2015, 8, 10), 41, 8),

};

// Act & Assert

for (int i = 0; i < testCases.GetLength(0); i++)

{

try

{

// Arrange

Employee employee = new Employee(testCases[i].Item2, testCases[i].Item3, testCases[i].Item4, testCases[i].Item5, testCases[i].Item6);

// Act

int age = 0;

int yearsOfExperience = 0;

library.InfoEmployee(employee, ref age, ref yearsOfExperience);

// Assert

Assert.AreEqual(testCases[i].Item7, age, $"Age is not as expected for employee {testCases[i].Item2}");

Assert.AreEqual(testCases[i].Item8, yearsOfExperience, $"Years of experience is not as expected for employee {testCases[i].Item2}");

}

catch (AssertFailedException e)

{

// Log failure

Console.WriteLine($"Failed Test Case: {testCases[i].Item1} - {e.Message}");

throw;

}

}

}

**6η Συνάρτηση Υλοποίησης**

**Α. DLL**

public int LiveInAthens(Employee[] Empls)

{

int numberOfEmployeesInAthens = 0;

foreach (Employee employee in Empls)

{

string phone = employee.HomePhone;

int phoneType = -1; // Assuming -1 means an invalid phone number

string info = null;

CheckPhone(phone, ref phoneType, ref info);

if (phoneType == 0 && info == "Ζώνη22.ΑνατολικήΣτερεάΕλλάδα,Αττική,ΝησιάΑιγαίου")

{

numberOfEmployeesInAthens++;

}

}

return numberOfEmployeesInAthens;

}

**B. Δημιουργία Περιπτώσεων Ελέγχου (TestCases)**

|  |  |  |
| --- | --- | --- |
|  | **Δοκιμαστικά Δεδομένα Εισόδου** | **Αναμενόμενα Αποτελέσματα** |
| **ID** | **Σύνολο των εργαζομένων** | **Πόσα άτομα** κατοικούν στην Αθήνα. |
| **1** | **Employees0\*\*\*** | **1** |
| **2** | **Employees1\*\*\*** | **2** |
| **3** | **Employees2\*\*\*** | **0** |

\*\*\*\*\*\*

Employee[] employees0 = new Employee[]

{

new Employee("John Doe", "2452345678", "69453458901", new DateTime(1985, 1, 15), new DateTime(2010, 3, 1)),

new Employee("Sam Smith", "2104009764", "6954277098", new DateTime(1990, 5, 10), new DateTime(2015, 7, 20)),

};

Employee[] employees1 = new Employee[]

{

new Employee("Margaret Luck", "2104009768", "6945629321", new DateTime(1960, 9, 18), new DateTime(2015, 6, 17)),

new Employee("Jane Sparrow", "2109876543", "6951266098", new DateTime(1999, 8, 10), new DateTime(2012, 2, 25)),

};

Employee[] employees2 = new Employee[]

{

new Employee("Maria Dolorez", "2348264738", "6945678901", new DateTime(1975, 1, 28), new DateTime(2020, 3, 11)),

new Employee("Nick Kayanas", "2228473848", "6952384928", new DateTime(1976, 5, 2), new DateTime(2019, 7, 23)),

};

[TestMethod]

public void LiveInAthens\_ReturnsCorrectNumberOfEmployees()

{

// Arrange

HRLibSDLL.HRLibSDLL library = new HRLibSDLL.HRLibSDLL();

Employee[] employees0 = new Employee[]

{

new Employee("John Doe", "2452345678", "69453458901", new DateTime(1985, 1, 15), new DateTime(2010, 3, 1)),

new Employee("Sam Smith", "2104009764", "6954277098", new DateTime(1990, 5, 10), new DateTime(2015, 7, 20)),

};

Employee[] employees1 = new Employee[]

{

new Employee("Margaret Luck", "2104009768", "6945629321", new DateTime(1960, 9, 18), new DateTime(2015, 6, 17)),

new Employee("Jane Sparrow", "2109876543", "6951266098", new DateTime(1999, 8, 10), new DateTime(2012, 2, 25)),

};

Employee[] employees2 = new Employee[]

{

new Employee("Maria Dolorez", "2348264738", "6945678901", new DateTime(1975, 1, 28), new DateTime(2020, 3, 11)),

new Employee("Nick Kayanas", "2228473848", "6952384928", new DateTime(1976, 5, 2), new DateTime(2019, 7, 23)),

};

var testCases = new (int, Employee[], int)[]

{

(1, employees0, 1),

(2, employees1, 2),

(3, employees2, 0),

};

// Act & Assert

for (int i = 0; i < testCases.GetLength(0); i++)

{

try

{

// Act

int numberOfEmployeesInAthens = library.LiveInAthens(testCases[i].Item2);

// Assert

Assert.AreEqual(testCases[i].Item3, numberOfEmployeesInAthens);

}

catch (AssertFailedException e)

{

// Log failure

Console.WriteLine($"Failed Test Case: {testCases[i].Item1} - {e.Message}");

throw;

}

}

}

// Act & Assert

for (int i = 0; i < testCases.GetLength(0); i++)

{

try

{

// Act

int numberOfEmployeesInAthens = library.LiveInAthens(testCases[i].Item2);

// Assert

Assert.AreEqual(testCases[i].Item3, numberOfEmployeesInAthens);

}

catch (AssertFailedException e)

{

// Log failure

Console.WriteLine($"Failed Test Case: {testCases[i].Item1} - {e.Message}");

throw;

}

}

}