using NUnit.Framework;

using ExtendedDatabase;

using System;

namespace Tests

{

public class ExtendedDatabaseTests

{

private const long InitialId = 100;

private const long DidiId = 156;

private const long TonyId = 298;

private const int NewID = 987;

private const long NegativeId = -9;

private const int InvalidID = 999;

private const string InitialUserName = "Gogo";

private const string DidiUserName = "Didi";

private const string TonyUserName = "Tony";

private const string NewUsername = "Fifi";

private const string InvalidUserName = "Pipi";

private const int InitialCount = 3;

private const int ExpectedCount = 4;

private const int ExpectedRemoveCount = 2;

private const int InvalidArrayLength = 17;

private const int ValidArrayLength = 13;

private Person personGogo;

private Person personDidi;

private Person personTony;

private ExtendedDatabase.ExtendedDatabase database;

[SetUp]

public void Setup()

{

this.personGogo = new Person(InitialId, InitialUserName);

this.personDidi = new Person(DidiId, DidiUserName);

this.personTony = new Person(TonyId, TonyUserName);

this.database = new ExtendedDatabase.

ExtendedDatabase(personGogo, personDidi, personTony);

}

[Test]

public void CtorOfTheClassPersonShoudInitializeCorrectly()

{

long expectedId = InitialId;

string expectedUsername = InitialUserName;

long actualId = this.personGogo.Id;

string actualUserName = this.personGogo.UserName;

Assert.IsTrue(expectedId == actualId);

Assert.True(expectedUsername == actualUserName);

}

[Test]

public void CtorExtendedDatabaseShouldInitializePersonArrayCorrectly()

{

int expectedCount = InitialCount;

int actluatCount = this.database.Count;

Assert.IsTrue(expectedCount == actluatCount);

}

[Test]

public void CtorExtendedDatabaseShouldThrowArgumentExceptionInvalidCapacity()

{

Person[] data = new Person[InvalidArrayLength];

for (int i = 0; i < data.Length; i++)

{

Person person = new Person(i, InitialUserName + i);

data[0] = person;

}

Assert.That(() => new ExtendedDatabase.ExtendedDatabase(data),

Throws

.ArgumentException

.With

.Message

.EqualTo("Provided data length should be in range [0..16]!"));

}

[Test]

public void AddElementWhenTheCollectionIsFullAndThrowInvalidOperationException()

{

Person[] data = new Person[ValidArrayLength];

for (int i = 0; i < data.Length; i++)

{

Person person = new Person(i, InitialUserName + i);

this.database.Add(person);

}

Person newPerson = new Person(NewID, NewUsername);

Assert.That(() => this.database.Add(newPerson),

Throws

.InvalidOperationException

.With

.Message

.EqualTo("Array's capacity must be exactly 16 integers!"));

}

[Test]

public void AddElementWithExistingUsernameShouldThrowException()

{

Person person = new Person(NewID, InitialUserName);

Assert

.Throws<InvalidOperationException>(

() => this.database.Add(person));

}

[Test]

public void AddElementWithExistingIdShouldThrowException()

{

Person person = new Person(InitialId, NewUsername);

Assert

.Throws<InvalidOperationException>(

() => this.database.Add(person));

Assert.That(() => this.database.Add(person),

Throws

.InvalidOperationException

.With

.Message

.EqualTo("There is already user with this Id!"));

}

[Test]

public void AddNewElementShouldWorkCorrect()

{

Person person = new Person(NewID, NewUsername);

this.database.Add(person);

int expectedCount = ExpectedCount;

int actualCount = this.database.Count;

Assert.AreEqual(expectedCount, actualCount);

}

[Test]

public void TestRemoveElementWhenTheCollectionIsEmptyShouldThrowException()

{

this.database = new ExtendedDatabase.ExtendedDatabase();

Assert

.Throws<InvalidOperationException>(

() => this.database.Remove());

}

[Test]

public void TestRemoveelementShouldWorkCorrect()

{

this.database.Remove();

int expectedCount = ExpectedRemoveCount;

int actualCount = this.database.Count;

Assert.That(expectedCount == actualCount);

}

[Test]

public void TestFindByUsernameWhenItIsNullOrEmptyShoulThrowException()

{

Assert

.Throws<ArgumentNullException>(

() => this.database.FindByUsername(null));

}

[Test]

public void TestFindByUsernameWhenItNotExistShouldThrowException()

{

Assert.That(() => this.database.FindByUsername(InvalidUserName),

Throws

.InvalidOperationException

.With

.Message

.EqualTo("No user is present by this username!"));

}

[Test]

public void TestFindByUsernameShouldWorkCorrect()

{

Person person = this.database

.FindByUsername(InitialUserName);

string expectedUsername = InitialUserName;

string actualUsername = person.UserName;

Assert.AreEqual(expectedUsername, actualUsername);

}

[Test]

public void TestFindByIdWhenItIsNegativeShouldThrowException()

{

Assert

.Throws<ArgumentOutOfRangeException>

(() => this.database.FindById(NegativeId));

}

[Test]

public void TestFindByIdWhenItIsNotExistShouldThrowException()

{

Assert.That(() => this.database.FindById(InvalidID),

Throws

.InvalidOperationException

.With

.Message

.EqualTo("No user is present by this ID!"));

}

[Test]

public void TestFindByIdWhenItIsExistShouldWorkCorrect()

{

Person person = this.database

.FindById(InitialId);

long expectedId = InitialId;

long actualId = person.Id;

Assert.AreEqual(expectedId, actualId);

}

}

}

RAW Paste Data