Politechnika Wrocławska

Dokumentacja
Platformy programistyczne .NET i Java

15 marca 2024

Filip Mirski nr indeksu 264267

1 Cel zadania

Celem laboratorium jest zapoznanie się z podstawami projektowania aplikacji .NET oraz pisaniem testów jednostkowych na przykładzie problemu plecakowego.

2 Zadanie

2.1 Pierwsza aplikacja konsolowa .NET – optymalizacja problemu plecakowego.

W tej części zadania wykonano aplikację konsolową dla problemu plecakowego. Na początku została stworzona klasa Backpack, gdzie została stworzona metoda Solve, która wykonywała ogólną logikę problemu.

```
20 references | 6/6 passing
public Result Solve(int capacity)
{
    items = items.OrderByDescending(item => (double)item.value / item.weight).ToList();

    List<int> selectedItems = new List<int>();
    int totalValue = 0;
    int totalWeight = 0;

    foreach (var item in items)
    {
        if (totalWeight + item.weight <= capacity)
        {
            selectedItems.Add(item.number);
            totalValue += item.value;
            totalWeight += item.weight;
            // item.fit = true; //DOPYTAĆ JAK TO OBEJŚĆ
        }
        else
        {
            break;
        }
    }

    Result result = new Result(selectedItems, totalValue, totalWeight);
    return result;
}
```

Rysunek 1: Metoda Solve dla problemu plecakowego

W tej metodzie najpierw posortowano listę z elementami "item" po stosunku wartości do wagi. Następnie na nową listę dodano elementy, które są najbardziej wartościowe i mieszczą się w plecaku. Zwróconą wartością był obiekt klasy Result, która ma przeciążoną funkcję ToString i zwraca odpowiedni format stringa z pobranymi elementami z posortowanej listy.

```
internal class Result
{
    public List<int> items;
    public int value;
    public int weight;

    2 references | 1/1 passing
    public Result(List<int> items, int value, int weight)
    {
        this.items = items;
        this.weight = weight;
        this.value = value;
    }

    2 references | 1/1 passing
    public override string ToString()
    {
        StringBuilder sb = new StringBuilder();
        sb.AppendLine("Items: ");
        for (int i = 0; i < items.Count(); i++)
        {
            sb.AppendLine("Item " + items[i]);
        }
        sb.AppendLine("Total value: " + value);
        sb.AppendLine("Total weight: " + weight);
        return sb.ToString();
    }
}
```

Rysunek 2: Zrzut ekranu z klasy Result

2.2 Testy jednostkowe dla aplikacji konsolowej.

W celu napisania testów jednostkowych dla aplikacji konsolowej należało dodać do projektu nowy projekt oparty na szablonie MSTest i zgodnie z syntaxem napisano 4 metody zaproponowane przez prowadzącego i 4 własne metody. Były to metody sprawdzające przeciążenie metody ToString, dwa razy, sprawdzenie konstruktora klasy Result oraz metody AddItem w klasie Backpack.

```
[TestMethod]
0 references
public void AddItemWorkProperly()
{
    Backpack specifiedItem = new Backpack(1, 12343, true);
    specifiedItem.AddItem(5, 10);

    CollectionAssert.AreEqual(new List<int> { 1 }, specifiedItem.Solve(10).items);
    Assert.AreEqual(5, specifiedItem.Solve(10).weight);
    Assert.AreEqual(10, specifiedItem.Solve(10).value);
}

[TestMethod]
0 references
public void CheckToStringMethod()
{
    Backpack specifiedItem = new Backpack(1, 12343, true);
    specifiedItem.AddItem(5, 10);

    string result = specifiedItem.ToString();
    byte[] byteArray = Encoding.UTF8.GetBytes(result);

    StringBuilder sb = new StringBuilder();
    sb.AppendLine("Number of items: " + 1);
    sb.AppendLine("Items: ");
    sb.AppendLine("Item " + 1 + ": weight = " + 5 + ", value = " + 10);
    string expected = sb.ToString();

    byte[] byteArrayExpected = Encoding.UTF8.GetBytes(expected);

    Assert.IsTrue(byteArrayExpected.SequenceEqual(byteArray));
}
```

Rysunek 3: Metody testowe sprawdzające metodę AddItem oraz metodę ToString z klasy Backpack

```
0 references
public void CheckResultToStringMethod()
{
    Backpack specifiedItem = new Backpack(1, 12343, true);
    specifiedItem.AddItem(5, 10);

    Result result = specifiedItem.Solve(10);

    string resultString = result.ToString();
    byte[] byteArray = Encoding.UTF8.GetBytes(resultString);

    StringBuilder sb = new StringBuilder();
    sb.AppendLine("Items: ");
    sb.AppendLine("Item " + 1);
    sb.AppendLine("Total value: " + 10);
    sb.AppendLine("Total weight: " + 5);
    string expected = sb.ToString();

    byte[] byteArrayExpected = Encoding.UTF8.GetBytes(expected);

    Assert.IsTrue(byteArrayExpected.SequenceEqual(byteArray));
}

[TestMethod]
0 references
public void TestResultConstructor()
{
    Result result = new Result(new List<int> { 1, 2, 3 }, 10, 5);
    Assert.AreEqual(10, result.value);
    Assert.AreEqual(5, result.weight);
    CollectionAssert.AreEqual(new List<int> { 1, 2, 3 }, result.items);
}
```

Rysunek 4: Metody testowe sprawdzające metodę ToString z klasy Result i kontruktor klasy Result

2.3 Graficzny interfejs użytkownika.

Na koniec całego zadania za pomocą szblonu WinForms stworzono aplikację okienkową w oparciu logikę konsolową aplikacji. W tym celu po stworzeniu interfejsu użytkownika dodano logikę do obiektu przycisku, która wyświetlała wyniki funkcji ToString w polach tekstowych.

```
1 reference
private void button1_Click(object sender, EventArgs e)
{
    int number = int.Parse(NumberOfItems.Text);
    int random_gen = int.Parse(random.Text);
    int capacity = int.Parse(volume.Text);

    Backpack pack = new Backpack(number, random_gen);
    InstanceTextBox.Text = pack.ToString();
    ResultsTextBox.Text = pack.Solve(capacity).ToString();
}
```

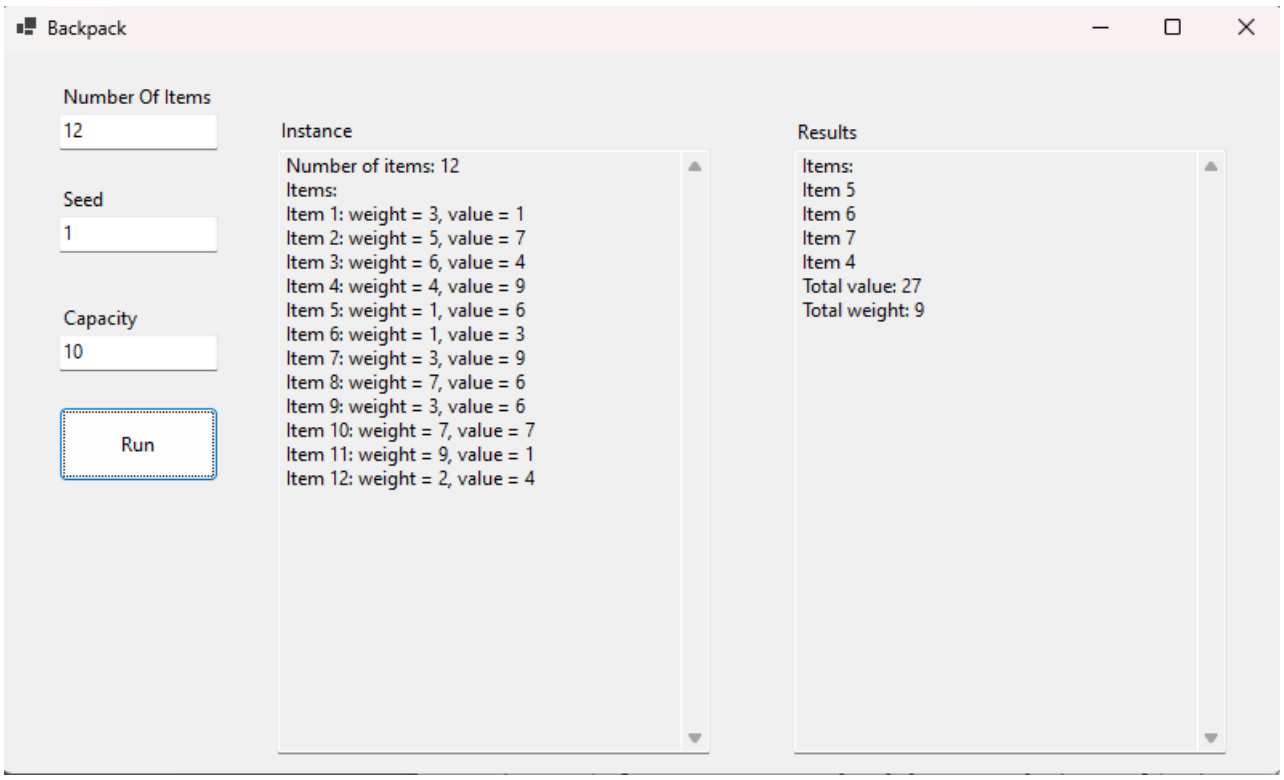
Rysunek 5: Logika aplikacji okienkowej

Aby zapobiec wszelkim problemom w atrybutach okienek tekstowych, które miały wyświetlać zawartość dodano flagę ReadOnly, aby wartości nie mogły zostać zmienione przez użytkownika oraz w miejscach gdzie należy wstawić wartości umożliwiono jedynie wprowadzanie wartości liczbowych.

```
1 reference
private void Form1_Load(object sender, EventArgs e)
{
    InstanceTextBox.ReadOnly = true;
    ResultsTextBox.ReadOnly = true;
}

1 reference
private void NumberOfItems_KeyPress(object sender, KeyPressEventArgs e)
{
    if (!char.IsControl(e.KeyChar) && !char.IsDigit(e.KeyChar)) { e.Handled = true; }
}
```

Rysunek 6: Metody z wprowadzonymi ograniczeniami



Rysunek 7: Graficzny interfejs użytkownika