

Platformy Programistyczne .NET i Java

Laboratorium 1

Projekt aplikacji .NET na przykładzie prostego problemu optymalizacyjnego

prowadzący: Dr inż. Radosław Idzikowski, mgr inż. Michał Jaroszczuk

1 Cel laboratorium

Celem laboratorium jest zapoznanie się z podstawami projektowania aplikacji .NET oraz pisanie testów jednostkowych na przykładzie problemu plecakowego. W ramach zajęć należy stworzyć program w języku C#, który będzie rozbudowywany o kolejne funkcjonalności. Jako IDE zostanie użyty Microsoft Visual Studio 2022 pod systemem operacyjnym Microsoft Windows. Praca będzie oceniana na bieżąco podczas zajęć. **Ukończenie każdego etapu powinno być zgłoszone prowadzącemu w celu akceptacji i odnotowania postępów.** Sam program należy umieścić na repozytorium github i wysłać zaproszenie do prowadzącego. Czas na wykonanie zadania to dwa zajęcia laboratoryjne. Podczas pierwszego spotkania trzeba wykonać co najmniej jedno zadanie zdefiniowane w Rozdziale 2.

2 Zadania

W ramach zajęć należy w zespołach wykonać następujące zadania:

1. Pierwsza aplikacja konsolowa .NET – optymalizacja problemu plecakowego.
2. Testy jednostkowe dla aplikacji konsolowej.
3. Graficzny interfejs użytkownika.

Za wykonanie zadania nr 1 wystawiana jest ocena dostateczna (3.0), za każde kolejne zadanie jest +1 do oceny. Na ocenę bardzo dobrą (5.0) należy wykonać wszystkie trzy zadania. W zadaniu oceniane będą także czystość kodu (trzymanie się konwencji nazewniczych, uporządkowanie kodu, itp.) oraz przestrzeganie paradygmatów programowania obiektowego. Link do repozytorium należy przesłać dopiero po oddaniu i ocenieniu pracy na laboratorium.

3 Opis zadań

W tej sekcji zostaną kolejno omówione wszystkie zadania do wykonania podczas laboratorium. Komputery w laboratorium zostały przygotowane do przeprowadzenia zajęć.

3.1 Zadanie 1

W ramach tego zadania będzie do wykonana nasza pierwsza aplikacja w technologii .NET. W tym celu należy uruchomić Microsoft Visual Studio 2022 oraz utworzyć nowy projekt Console App (Aplikacja konsolowa) zgodna z Framework .NET 8.0 (LTS). **UWAGA!** Domyślnie od .NET

6.0 aplikacja zostanie utworzona z użyciem `top-level-statements` (instrukcje najwyższego poziomu), tzn. nie będzie **jawnie** zdefiniowana klasa `Program` oraz metoda `Main`. Po zaznaczeniu opcji, aby ich **nie** używać (Do not use top-level statements) i utworzeniu projektu, otrzymamy poniższy kod:

```
1 namespace ConsoleApp1
2 {
3     internal class Program
4     {
5         static void Main(string[] args)
6         {
7             Console.WriteLine("Hello, World!");
8         }
9     }
10 }
```

Problem plecakowy

W pierwszej kolejności należy przygotować generator instancji problemu plecakowego zgodnie z jego opisem. W problemie plecakowym mamy zbiór przedmiotów $\mathcal{I} = \{1, 2, \dots, n\}$, gdzie n to liczba przedmiotów. Każdy przedmiot jest opisany dwoma parametrami:

- v_i – wartość,
- w_i – waga.

Dla każdego przedmiotu należy określić wartość binarną $x_i \in \{0, 1\}$, gdzie „0” oznacza niewłożenie przedmiotu do plecaka, a „1” umieszczenie przedmiotu w plecaku. Pojemność plecaka jest ograniczona wartością C , gdzie suma wag przedmiotów w plecaku nie może przekraczać jego maksymalnej pojemności:

$$\sum_{i=1}^n x_i w_i \leq C. \quad (1)$$

Zadaniem jest znalezienie zbioru przedmiotów o jak największej wartości:

$$\text{maximize } \sum_{i=1}^n x_i v_i. \quad (2)$$

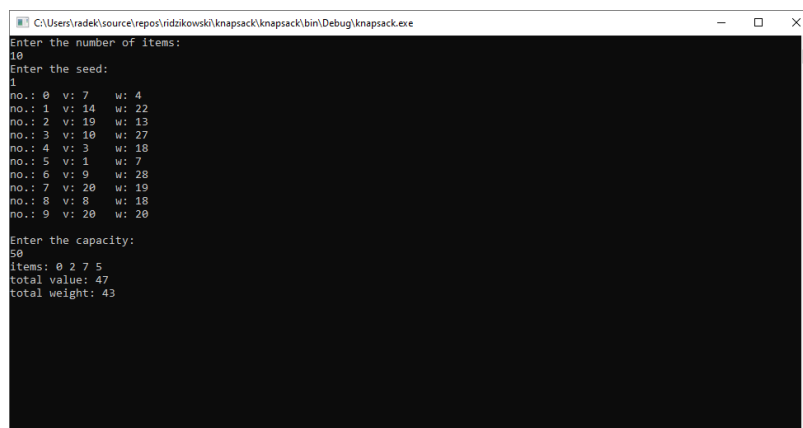
pamiętając o ograniczeniu (1). W tym celu należy utworzyć w projekcie nową klasę `Problem` dla opisanego problemu plecakowego. We wspomnianej klasie musimy mieć co najmniej dwa pola odpowiedzialne za: (1) liczbę przedmiotów oraz (2) listę przedmiotów (listę obiektów klasy `Przedmiot`). Listę przedmiotów można zastąpić dwoma listami lub tablicami, które będą przechowywać informacje o przedmiotach, czyli ich wagę i wartość. Następnie należy utworzyć konstruktor z dwoma parametrami: (1) `n` – liczba przedmiotów oraz (2) `seed` – źródło/nasionko losowania, który wygeneruje nam instancję problemu. Należy skorzystać z domyślnego generatora liczb pseudolosowych, tzn. w pierwszej kolejności tworzymy obiekt klasy `Random`, a później przy użyciu metody `Next()` można wylosować liczbę całkowitą. Jeśli nie podamy argumentów, to wylosowana liczba będzie z przedziału od 0 do `Int32.MaxValue`, w przypadku jednego parametru od 0 do tej wartości bez niej lub dla dwóch parametrów od pierwszej wartości, włącznie z nią. Na potrzeby niniejszego projektu należy przyjąć wartość i wagę pojedynczego przedmiotu z zakresu $v_i, w_i \in < 1, 10 >$

```
1 Console.WriteLine("Enter the seed:");
2 int seed = int.Parse(Console.ReadLine());
3 Random random = new Random(seed);
4 Console.WriteLine(random.Next(10));
```

Na koniec należy jeszcze przeciążyć metodę `ToString()`, żeby możliwe było zwracanie instancji naszego problemu jako `string`. Będzie to przydatne w późniejszych etapach implementacji aplikacji - znacznie uprości wykonanie testów i przejście na graficzny UI.

Metoda rozwiązania

Do rozwiązania problemu należy zaimplementować algorytm zachłanny. W tym celu trzeba utworzyć metodę `Solve(int capacity)` wewnątrz naszej klasy problemu, która jako argument powinna przyjmować pojemność plecaka oraz zwracać rozwiązanie jako obiekt klasy `Result`. Klasa `Result` powinna zawierać listę numerów przedmiotów w plecaku, ich sumaryczną wartość, sumaryczną wagę oraz przeciążoną metodę `ToString()`. Wewnątrz metody rozwiązującej problem, w pierwszej kolejności należy posortować przedmioty po stosunku ich wartości do wagi. Warto skorzystać z gotowej metody w ramach kolekcji. Przedmioty najbardziej opłacalne mają być na początku. Następnie należy kolejno próbować umieszczać przedmioty w plecaku dopóki się w nim mieszczą. Algorytm zakończy się w momencie sprawdzenia wszystkich przedmiotów lub jeśli plecak zostanie wypełniony całkowicie. Na Rysunku 1 przedstawiono przykładowe działanie programu. Rysunek ma służyć wyłącznie za inspirację.



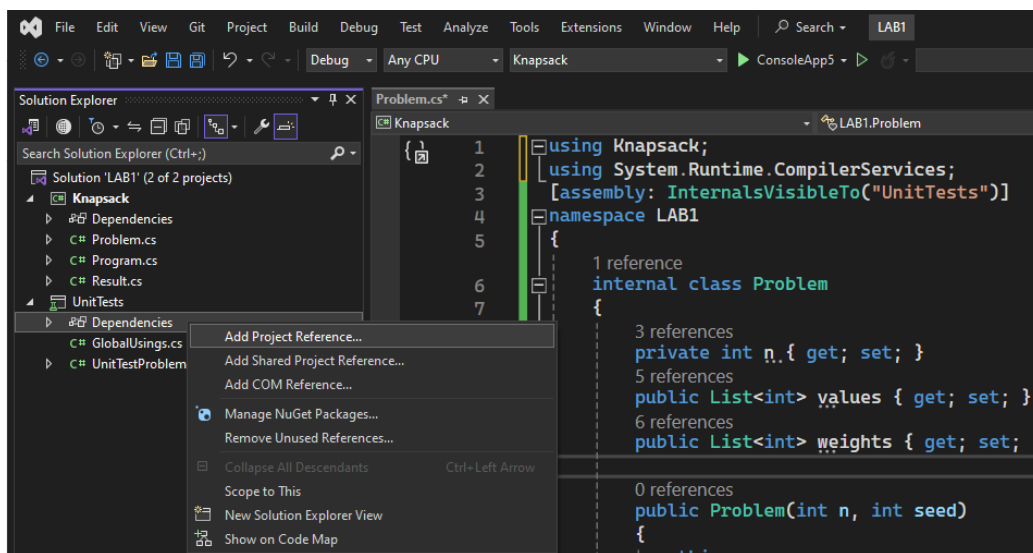
```
C:\Users\vadek\source\repos\ridzikowski\knapsack\knapsack\bin\Debug\knapsack.exe
Enter the number of items:
10
Enter the seed:
1
no.: 0 v: 7 w: 4
no.: 1 v: 14 w: 22
no.: 2 v: 19 w: 13
no.: 3 v: 10 w: 27
no.: 4 v: 3 w: 18
no.: 5 v: 1 w: 7
no.: 6 v: 9 w: 28
no.: 7 v: 20 w: 19
no.: 8 v: 8 w: 18
no.: 9 v: 20 w: 20
Enter the capacity:
50
Items: 0 2 7 5
total value: 47
total weight: 43
```

Rysunek 1: Aplikacja konsolowa – problem plecakowy

3.2 Zadanie 2

W tym zadaniu należy przećwiczyć tworzenie i przeprowadzanie testów jednostkowych zaimplementowanych funkcjonalności. W tym celu w ramach naszego `Solution (Rozwiązania)` trzeba dodać nowy projekt `MSTest Project`. Można również skorzystać z `JUnit Test Project`, jednak należy mieć na uwadze, że niektóre metody mogą się różnić w porównaniu z `MSTest`. Bardzo ważne, aby nowo dodany projekt był zgodny z tym samym `Framework`, w naszym wypadku `.NET 8.0 (LTS)`. Zazwyczaj do każdej klasy powinna być utworzona osobna klasa testów jednostkowym, ale w omawianym przypadku wystarczy jedna klasa. W pierwszej kolejności w nowo utworzonym projekcie testów, należy dodać `Dependencies (odwołanie)` do projektu z głównym programem, tak jak pokazano na Rysunku 2.

Ponadto musimy ustawić widoczność testowanej klasy (**bez zmieniania** jej typu na publiczny) w bazowym projekcie w ramach całego zestawu (`assembly`). W tym celu trzeba dodać dyrektywę `[assembly: InternalsVisibleTo("TestProject")]`, gdzie `TestProject` to nazwa naszego projektu z testami, co jest dobrze widocznie również na Rysunku 2. Dyrektywę umieszczamy w pliku z testowaną klasą bezpośrednio nad nazwą naszego `namespace`. Teraz będzie można zaimportować `namespace` wraz z zdefiniowaną klasą w projekcie z testami poprzez użycie polecenia `using`. Testy jednostkowe mogą mieć postawiony warunek na różne sposoby, ale zawsze muszą być tak napisane, aby **prawidłowo działający kod przeszedł wszystkie testy**. Poniżej zamieszczono przykład testu jednostkowego dla klasy generującej `n` losowych liczb. Test sprawdza czy w rzeczywistości wewnątrz obiektu dodało się do listy dokładnie `n` elementów.



Rysunek 2: Dodanie odwołania do projektu

```

1 using ConsoleApp;
2 namespace TestProject
3 {
4     [TestClass]
5     public class UnitTest
6     {
7         [TestMethod]
8         public void TestMethodCountElements()
9         {
10             List<int> sizes = new List<int>() { 10, 20, 30, 40, 50 };
11             foreach (int n in sizes)
12             {
13                 Problem problem = new Problem(n);
14                 Assert.AreEqual(n, problem.values.Count);
15             }
16         }
17     }
18 }

```

Zasadniczym celem zadania jest napisanie kilku testów jednostkowych, w celu sprawdzenia poprawności działania zaimplementowanych funkcji. Przykładowe testy jednostkowe, zostały zaproponowane poniżej:

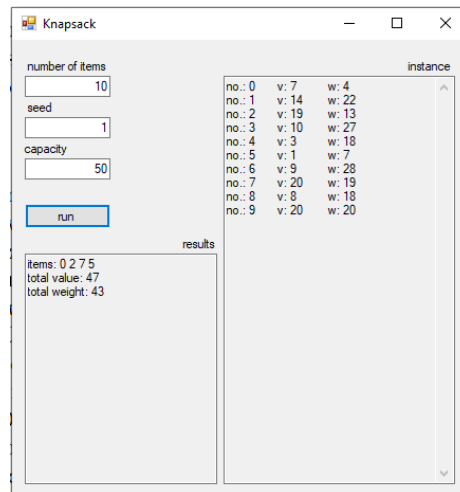
- Sprawdzenie, czy jeśli co najmniej jeden przedmiot spełnia ograniczenia, to zwrócono co najmniej jeden element.
- Sprawdzenie, czy jeśli żaden przedmiot nie spełnia ograniczeń, to zwrócono puste rozwiązanie.
- Sprawdzenie poprawności wyniku dla konkretnej instancji.

Istotne jest, aby testy były zróżnicowane i miały różnie postawiony warunek. Obowiązkowo trzeba napisać wszystkie zaproponowane testy plus 2 wymyślone samodzielnie w celu uzyskania maksymalnej oceny za tą część zadania. Przy ocenianiu będzie brane pod uwagę zróżnicowanie testów i typów asercji oraz to czy zaproponowane testy mają sens.

3.3 Zadanie 3

Celem zadania jest utworzenie Graficznego Interfejsu Użytkownika (GUI) dla aplikacji z zadania pierwszego. Jeśli zadanie 1 zostało wykonane zgodnie z wytycznymi to nie będzie potrzeby wprowadzania w nim zmian i będzie można wykorzystać stworzone wcześniej klasy. W pierwszej kolejności postępujemy analogicznie jak w przypadku projektu z testami jednostkowymi. Należy dodać projekt **Windows Forms App** do naszego **Solution (Rozwiązania)**, pamiętając o **.NET 8.0 (LTS)**. W tym wypadku również dodajemy w projekcie z **GUI Dependencies (odwołanie)** do projektu z aplikacją konsolową, potem również rozszerzamy widoczność klasy z problemem na ten projekt, `[assembly: InternalsVisibleTo("TestProject"), InternalsVisibleTo("GUI")]` (proszę zwrócić uwagę, że jest w formie listy).

Do wykonania tego zadania będzie potrzebne co najmniej kilka kontrolki `textBox` oraz minimum jeden `button`. Obie kontrolki można dodać z poziomu **Design** z **Toolbox** i przesunąć na pole naszego **Form** z widokiem aplikacji. Należy zadbać o odpowiednie (spójne i jednoznaczne) nazewnictwo dodanych kontroltek, gdyż ułatwia to ich użycie i wprowadza czytelność w projekcie. Do tekstu w ramach `textBox` odwołujemy się przez pole `Text`, które jest typu `string`, co niesie za sobą konieczność parsowania wprowadzonej wartości. Dla poprawy czytelności warto używać kontrolki `label`, aby dodać odpowiednie podpisy. `Textbox` (warto włączyć opcję `multiline`) z wynikiem powinien być tylko do odczytu (alternatywnie warto użyć `listBox`). W celu uzyskania maksymalnej oceny trzeba zadbać o minimalną kontrolę błędów. Może być ona zrealizowana na przykład poprzez podświetlanie pola tekstowego na odpowiedni kolor po wpisaniu liczby spoza zakresu. **UWAGA!** w trybie projektowania dwuklik myszką na dowolną dodaną kontrolkę powoduje wygenerowanie funkcji, którą podcina pod `Event` po jej naciśnięciu (przydatne w przypadku przycisków). Kształt czy zachowanie wstawionych kontroltek, można zmieniać na dwa sposoby: (1) z poziomu projektowania – zakładka **Properties (Właściwości)** lub (2) bezpośrednio z poziomu kodu. Przykładową aplikację pokazano na Rysunku 3.



Rysunek 3: Aplikacja okienkowa – problem plecakowy