SVEUČILIŠTE U SPLITU

FAKULTET ELEKTROTEHNIKE, STROJARSTVA I BRODOGRADNJE

Algoritmi i strukture podataka

Vježba 5.

Stacks and queues

Ivan Banovac, 220

Datum izvođenja vježbe:
22.12.2025

Zadatak 1. – Stacks

Stack.cs:

```
using Z1;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace Stack
{
    class Stack
    {
        private List list;
        public bool IsEmpty()
        {
            return list.IsEmpty();
        }
        public void Display()
        {
            list.Display();
        }
        public Stack()
        {
            list = new List();
        }
        public void Push(object element)
        {
            list.InsertFront(element);
        }
        public object Pop()
        {
            if (IsEmpty())
                throw new Exception("Stack is empty!");
            return list.Remove Front();
        }
    }
}
```

Program.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace Stack
{
    class Program
    {
{
    static void Main(string[] args)
    Stack stack = new Stack();
        stack.Push("Ivo");
        stack.Push("Ana");
        stack.Push("Bob");
        stack.Push("Iva");
        stack.Push("Kim");
        stack.Display();
try try
{
for (int i = 0; i< 6 i++)
{
stack.Pop();
stack.Display();
}
}
catch (Exception x)a
{
}
Console.WriteLine(x.Message);
stack.Display();
}
}
}
```

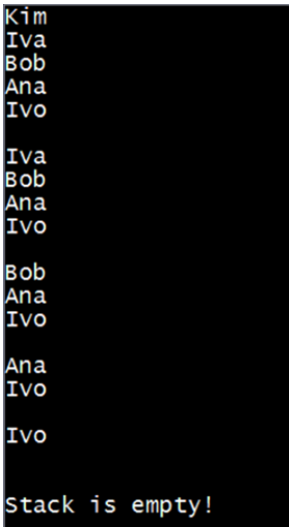
Cilj ove vježbe je implementacija stoga i reda čekanja korištenjem jednostruko povezane liste(povezano sa V4).

Umjesto klasičnog nasljeđivanja, koristi se delegiranje poziva metodama klase *List* kako bi se izbjeglo preuzimanje svih njezinih metoda i zadržale samo potrebne operacije.
Stog radi po principu *FILO (First-In-Last-Out)*, što znači da se elementi dodaju i uklanjaju isključivo s vrha povezane liste.

Metoda *Push* koristi *InsertFront* za dodavanje elemenata na vrh, dok metoda *Pop* koristi *RemoveFront* za njihovo uklanjanje. Prije svakog uklanjanja, sustav pomoću metode *IsEmpty* provjerava je li stog prazan kako bi se spriječile pogreške. U glavnom programu, nakon dodavanja nekoliko stringova na stog, elementi se uklanjaju jedan po jedan unutar catch bloka sve dok se ne dogodi iznimka zbog praznog stoga.

Unutar petlji u *main-a* pokušavamo ukloniti šest elemenata, a na stogu ih je pohranjeno pet. Prilikom zadnje iteracije dolazi do pokušaja operacije nad praznom strukturom, što rezultira greškom tj iznimkom. Greška se spriječava tako da presretnemo grešku u *catch* bloku gdje ispisujemo poruku.

Rezultat:



Zadatak 2. – Queues

Queue.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Single_List;
namespace Queue
{
    class Queue
    {
        private List list;
        public Queue()
        {
            list = new List();
        }
        public void Enqueue(object element)
        {
            list.InsertEnd(element);
        }
        public void Dequeue()
        {
            if (IsEmpty())
                throw new Exception("Queue is empty!");
            else
                list.Remove Front();
        }
        public bool IsEmpty() { return list.IsEmpty(); }
}
public void Display() { list.Display(); }
    }
```

Program.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace Queue
{
    class Program
    {
        static void Main(string[] args)
       {
       Queue queue = new Queue();
       queue.Enqueue("Ivo");
       queue.Enqueue("Ana");
       queue.Enqueue("Bob");
       queue.Enqueue("Iva");
       queue.Enqueue("Leo");
       queue.Display();
try
{
       for (int i = 0; i< 6 ; i++)
       {
       queue.Dequeue();
       queue.Display();
       }
}
catch (Exception x)
{
       {
              Console.WriteLine(x.Message);
       queue.Display();
       }
}
```

U klasi *Queue* dodaje se referenca na jednostruko vezanu listu kako bi se mogle koristiti njezine metode. Time se izbjegava nasljeđivanje klase *List*, budući da su za implementaciju reda potrebne samo osnovne operacije dodavanja i uklanjanja elemenata. U klasi se definiraju dvije nove metode: *Enqueue(object element)* i *Dequeue()*. Metoda *Enqueue* koristi metodu *InsertEnd* vezane liste kako bi se novi elementi dodavali na kraj reda, čime se osigurava *FIFO (First-InFirst-Out)* princip rada. Kod metode *Dequeue* najprije se provjerava je li red prazan pomoću metode *IsEmpty*. Ako red nije prazan, koristi se metoda *RemoveFront* kojom se uklanja element s početka reda.

Unutar glavnog programa započinje se kreiranje instance objekta *Queue*. Nakon inicijalizacije, u red se pomoću metode *Enqueue* dodaje ukupno pet elemenata tipa string: "Ivo", "Ana", "Bob", "Iva" i "Leo". Kako bi se dobio uvid u početno stanje strukture prije bilo kakvih modifikacija, poziva se metoda *Display()* koja ispisuje trenutni sadržaj reda na konzolu