SPRAWOZDANIE	
Przedmiot	**Temat**
Niezawodność systemów informatycznych	Stos i kolejka – C# - NSI1
Imię i nazwisko	**Data wykonania zadania**
Konrad Kulka	19.10.2022 r.

Zadanie:

Zadanie

Zaimplementuj klasy: stos, kolejka na bazie następującego interfejsu polimorficznego (na ocenę dst):

```
0 references
abstract class Container
{
    protected int pointer = -1;
    protected int[] buffer = new int[10];

    0 references
    public abstract int pop();

    0 references
    public abstract void push(int value);
}
```

Do powyższej implementacji dołóż możliwości sprawdzania rozmiaru kolekcji, czyszczenia jej itp (metody Clear(), GetCount(), IsEmpty(), IsFull()). Staraj się to wykonać w optymalny sposób, korzystając z dziedziczenia i polimorfizmu. (na ocenę db).

Wykonaj implementacje, która w momencie tworzenia obiektu pozwala zdecydować o ilości miejsca w buforze oraz typie elementów (typ generyczny), na którym operuje implementowany kontener. (na ocenę bdb).

Rozwiązanie:

Na początku zdefiniowany został podstawowy interfejs polimorficzny, wykorzystany zostanie do rozwiązania zadania. Znajduje się w nim: definicja wskaźnika, dynamiczny rozmiar, wypisanie i przeliczenie zawartości, wypychanie, usuwanie i czyszczenie containera, sprawdzenie czy jest pusty i pełny:

```
abstract class Container<S>
{
    protected int pointer = -1;
    protected S[] buffer = new S[10]; // tu na bdb definicja jakiego rodzaju zmiennych użyć w container, do zapamiętania S to zmienna typu
    protected int rozmiar = 10; // rozmiar container - domyślnie 10
    Odwołania: 8
    public abstract void write(); // wypisywanko
    Odwołania: 5
    public abstract S pop();
    Odwołania: 14
    public abstract void push(S value);
    Odwołania: 0
    public Container() {} // na bdb definiowanie rozmiaru kontenera
    Odwołania: 2
    public Container(int roozmiar)
    {
        if (rozmiar > 0)
        {
            buffer = new S[roozmiar];
            rozmiar = roozmiar; // kapitalne nazwanie rozmiaru kontenera, świetny pomysł
        }
    }
    //niestety trzeba tutaj zdefiniować clear, getCount, isEmpty i isFull bo klasy się buntują jak tego tutaj nie zrobię
    Odwołania: 0
    public void clear() //czyszczenie buforów, tworzenie nowego containera
    {
        buffer = new S[10];
        pointer = -1;
    }

    Odwołania: 2
    public int getCount() //wypisanie ilości zapełnionych buforów w containerze
    {
        return pointer + 1;
    }
```

```
    public Boolean isEmpty() //wypisanie, czy container jest pusty, bool tu nie zadziała
    {
        if (pointer == -1)
        {
            return true;
        } else
        {
            return false;
        }

    }
    Odwołania: 2
    public Boolean isFull() { //wypisanie, czy container jest pełny
        if (pointer == rozmiar - 1)
        {
            return true;
        }
        else {
            return false;
        }
    }

}
```

Aby zoptymalizować działanie programu, funkcje write(), clear(), getCount, isEmpty(), isFull() zostały zdefiniowane w głównej klasie Container, przez co nie zachodzi potrzeba definiowania ich przy definiowaniu klas dziedziczących.

Następnie zdefiniowano klasę Stos, dziedzicząca z klasy Container właściwości. Dzięki temu będzie można nadpisywać przykładowo zmienną przeliczenia ilości zapełnionych buforów w stosie:

```
Odwołania: 5
class Stos<S> : Container<S> //<S> definiuje rozmiar stosu
{
    Odwołania: 2
    public Stos(int rozmiar) : base(rozmiar) { } //inicjalizacja nowego stosu
    Odwołania: 3
    public override S pop()
    {
        if (pointer < 0)
        {
            pointer = 0;
        }
        S popper = buffer[pointer];
        pointer--;
        return popper;
    }
    Odwołania: 12
    public override void push(S value)
    {
        pointer++;
        if (pointer < rozmiar)
        {
            buffer[pointer] = value;
        }
        else
        {
            pointer = rozmiar - 1;
        }
    }

    Odwołania: 5
    public override void write()
    {
        //Console.WriteLine("Stos: ");
        for (int i = 0; i <= pointer; i++)
        {
            Console.Write(buffer[i] + " ");
        }
        Console.WriteLine(" ");
    }

}
```

Następnie zdefiniowano klasę Kolejka, która tak jak klasa Stos również dziedziczy z klasy Container:

```
Odwołania: 3
class Kolejka<S> : Container<S> //<S> definiuje rozmiar kolejki
{
    1 odwołanie
    public Kolejka(int rozmiar) : base(rozmiar) { } //inicjalizacja nowej kolejki
    Odwołania: 2
    public override S pop()
    {
        S popper2 = buffer[0];
        for (int i = 1; i <= pointer; i++)
        {
            buffer[i - 1] = buffer[i];
        }
        pointer--;
        if (pointer < -1)
        {
            pointer = -1;

        }
        return popper2;
    }
    Odwołania: 2
    public override void push(S value)
    {
        pointer++;
        if (pointer < rozmiar)
        {
            buffer[pointer] = value;
        }
        else
        {
            pointer = rozmiar - 1;
        }
    }
    Odwołania: 3
    public override void write()
    {
      // Console.WriteLine("Kolejka: ");
        for (int i = 0; i <= pointer; i++)
        {
            Console.Write(buffer[i] + " ");
        }
        Console.WriteLine(" ");
```

Po zdefiniowaniu klas Stos i Kolejka, można już zdefiniować klasę Program, w którym odbędą się operacje na klasach. Działa to w następującej kolejności: zdefiniowanie nowego stosu o rozmiarze 10, zapełnienie go liczbami od 0 do 9 - push, wypisywanie w trakcie zapełniania liczby zapełnionych buforów - getCount, wypisanie gotowego stosu - write, sprawdzenie, czy stos jest pusty i pełny - i isFull, usunięcie ze stosu - pop, ponowne wypisanie gotowego stosu w celu sprawdzenia, czy usunięcie przebiegło pomyślnie:

```
Console.WriteLine("Niezawodność systemów informatycznych - Stos i kolejka - zadanie z 20.10.2022 r. - Konrad Kulka - P1");
Console.WriteLine(" ");
/*Console.WriteLine("Jakiego typu zmiennych ma być zapełniona kolejka?");
string typzmiennych;
int typzmiennychnaint;
typzmiennych = console.readline();
typzmiennychnaint = convert.toint32(typzmiennych);*/
Stos<int> s = new Stos<int>(10); //zdefiniowanie nowego stosu o rozmiarze 10 i typie int
Console.WriteLine("Zapełnianie stosu...");
for (int i = 0; i < 10; i++) //zamiast wypisywania przez użytkownika do stosu dodaje się liczby od 1 do 10
{
    s.push(i);
    Console.WriteLine("Obecna liczba zapełnionych buforów: " + s.getCount());
}
Console.WriteLine(" ");
Console.WriteLine("Stos wygląda następująco: ");
s.write();
Console.WriteLine(" ");
bool ifempty = s.isEmpty();
bool iffull = s.isFull();
if (ifempty == true) //na db sprawdzenie czy stos jest pusty
{
    Console.WriteLine("Stos jest pusty");
} else
{
    Console.WriteLine("Stos nie jest pusty");
}
if (iffull == true) //na db sprawdzenie czy stos jest pełny
{
    Console.WriteLine("Stos jest pełny");
}
else
{
    Console.WriteLine("Stos nie jest pełny");
}
s.pop(); //usunięcie ze stosu wartości wyżej
Console.WriteLine(" ");
Console.WriteLine("Po usunięciu stos wygląda następująco: ");
s.write();
Console.WriteLine(" ");
```

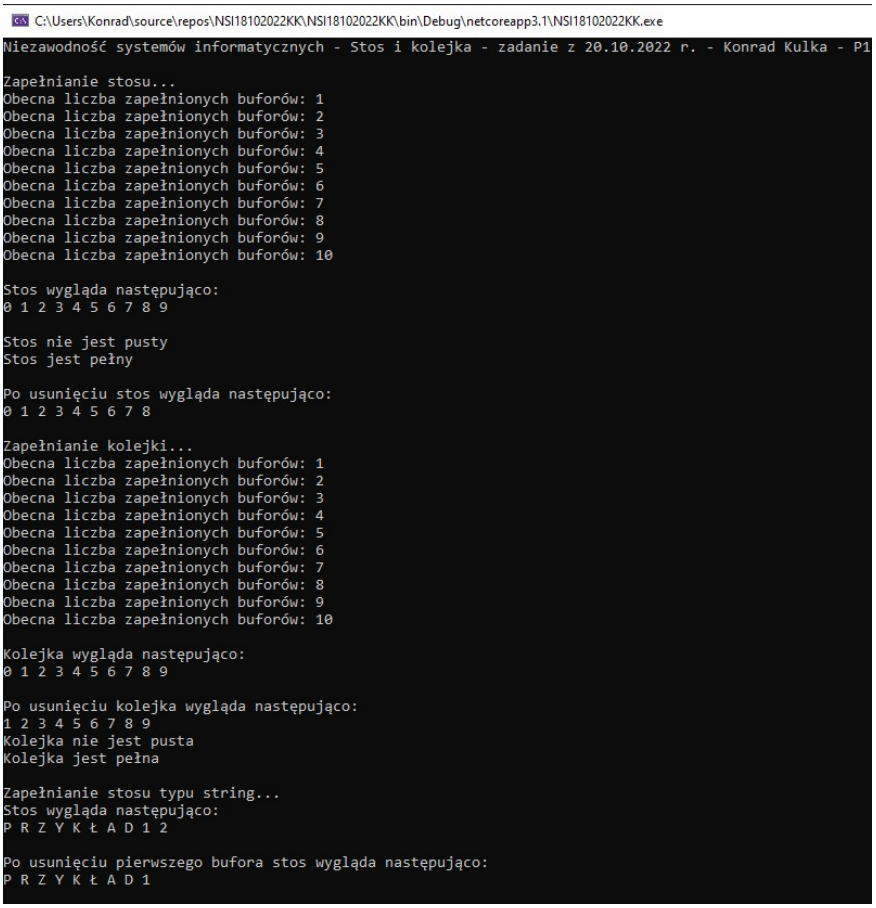
Podobnie jak w klasie Stos, następne działania zostaną podjęte w następującej kolejności: zdefiniowanie nowej kolejki o rozmiarze 10, zapełnienie jej liczbami od 0 do 9 - push, wypisywanie w trakcie zapełniania liczby zapełnionych buforów - getCount, wypisanie gotowej kolejki - write, sprawdzenie, czy kolejka jest pusta i pełna – isFull, isEmpty, usunięcie z kolejki - pop, ponowne wypisanie gotowego stosu w celu sprawdzenia, czy usunięcie przebiegło pomyślnie:

```
Kolejka<int> k = new Kolejka<int>(10); //zdefiniowanie nowej kolejki o rozmiarze 10  i typie int
Console.WriteLine("Zapełnianie kolejki...");
for (int i = 0; i < 10; i++) //zamiast wypisywania przez użytkownika do stosu dodaje się liczby od 1 do 10
{
    k.push(i);
    Console.WriteLine("Obecna liczba zapełnionych buforów: " + k.getCount());
}
Console.WriteLine(" ");
Console.WriteLine("Kolejka wygląda następująco: ");
k.write();
bool ifemptyk = k.isEmpty();
bool iffullk = k.isFull();
k.pop(); //usunięcie z kolejki
Console.WriteLine(" ");
Console.WriteLine("Po usunięciu kolejka wygląda następująco: ");
k.write();
if (ifemptyk == true) //na db sprawdzenie czy stos jest pusty
{
    Console.WriteLine("Kolejka jest pusta");
}
else
{
    Console.WriteLine("Kolejka nie jest pusta");
}
if (iffullk == true) //na db sprawdzenie czy stos jest pełny
{
    Console.WriteLine("Kolejka jest pełna");
}
else
{
    Console.WriteLine("Kolejka nie jest pełna");
}
Console.WriteLine(" ");
```

W celu sprawdzenia możliwości definicji integeru danych w stosie, zdefiniowano nowy stos wykorzystujący string i dokonano modyfikacji. Zamiast generowania liczb wypychane będą litery „na sztywno” - push,wypisanie – write, usunięcie ze stosu – pop, ponowne wypisanie:

```
        Stos<string> s2 = new Stos<string>(10); //zdefiniowanie stosu o rozmiarze 10 i typie string, pushowane będą litery
        Console.WriteLine("Zapełnianie stosu typu string...");
        s2.push("P");
        s2.push("R");
        s2.push("Z");
        s2.push("Y");
        s2.push("K");
        s2.push("Ł");
        s2.push("A");
        s2.push("D");
        s2.push("1");
        s2.push("2");
        Console.WriteLine("Stos wygląda następująco: ");
        s2.write();
        Console.WriteLine(" ");
        s2.pop();
        Console.WriteLine("Po usunięciu pierwszego bufora stos wygląda następująco: ");
        s2.write();
        Console.ReadLine();
    }
}
}
```

Wynik:



Wnioski:

Wykonanie zadania, pomimo paru problemów natury technicznej (konfiguracja środowiska Visual Studio – na początku nie był w stanie skompilować programu, konsola samoistnie się wyłączała – rozwiązano przy pomocy Console.ReadLine()) udało się pomyślnie rozwiązać zadanie. W celu ułatwienia rozwiązania i powtórki wiedzy z zakresu języka C#, a dokładniej z tematu polimorfizmu i interfejsów, skorzystano z dokumentacji zawartej na stronie w3schools.com. Zdobyta wiedza będzie przydatna w rozwiązywaniu kolejnych zadań z przedmiotu Niezawodność systemów informatycznych.