Министерство образования Республики Беларусь

Учреждение образования

«Белорусский государственный университет

информатики и радиоэлектроники»

Кафедра инженерной психологии и эргономики

­­­­­

Конструирование программного обеспечения

Отчет по лабораторной работе №6

«Делегаты, события и лямбды»

Выполнил: Усов А.М.

Студент группы 310901

Преподаватель: Давыдович К. И.

Минск 2023

Цель: познакомиться с использованием делегатов в приложениях. Научится описывать собственные события. Познакомится с механизмом обработки событий.

Задание 1. Для созданного в предыдущей лабораторной работе контейнерного класса реализовать методы, которые в качестве аргумента принимают делегат:

* метод сортировки;
* метод поиска;
* метод фильтрации.

Для каждого делегата определить один метод и одно лямбда-выражение.

Коллекция изменяется при удалении/добавлении элементов или при изменении одной из входящих в коллекцию ссылок, например, когда одной из ссылок присваивается новое значение. В этом случае в соответствующих методах или свойствах класса бросаются события.

При изменении данных объектов, ссылки на которые входят в коллекцию, значения самих ссылок не изменяются. Этот тип изменений не порождает событий. Для событий, извещающих об изменениях в коллекции, определяется свой делегат. События регистрируются в специальных классах-слушателях

Для событий предусмотреть возможность подписки и отписки от события.

Для обработки всех ошибочных ситуаций использовать конструкцию try…catch().

В Main создать два экземпляра шаблонного класса-контейнера для разных типов данных. Работа с этими объектами должна демонстрироваться на следующих операциях: добавить – просмотреть – найти – удалить – найти – просмотреть.

Отладить и выполнить полученную программу. Проверить обработку исключительных ситуаций (например, чтение из пустого стека, дублирование объектов и т.п.).

Код программы 1:

class Program

{

public static void Main(string[] args)

{

Ring<int> ringInt = new Ring<int>();

ringInt.Sort = () => ringInt.BubleSort();

ringInt.Get = (int value) => ringInt.GetByValue(value);

ringInt.GetById = (int index) => ringInt.GetByIndex(index);

ringInt.Filtered = (int value) => ringInt.FilteredByValue(value);

ringInt.ElementAdded += AddEventMethod;

ringInt.ElementRemoved += RemoveEventMethod;

Ring<string> ringString = new Ring<string>();

ringString.Sort = () => ringString.BubleSort();

ringString.Get = (string value) => ringString.GetByValue(value);

ringString.GetById = (int index) => ringString.GetByIndex(index);

ringString.Filtered = (string value) => ringString.FilteredByValue(value);

ringString.ElementAdded += AddEventMethod;

ringString.ElementRemoved += RemoveEventMethod;

Ring<double> ringDouble = new Ring<double>();

ringDouble.Sort = () => ringDouble.BubleSort();

ringDouble.Get = (double value) => ringDouble.GetByValue(value);

ringDouble.GetById = (int index) => ringDouble.GetByIndex(index);

ringDouble.Filtered = (double value) => ringDouble.FilteredByValue(value);

ringDouble.ElementAdded += AddEventMethod;

ringDouble.ElementRemoved += RemoveEventMethod;

Console.WriteLine("Work with Ring int model:\n add - veiw - find - remove - find - veiw");

// add

ringInt.push(1);

ringInt.push(6);

ringInt.push(2);

ringInt.push(3);

ringInt.push(4);

ringInt.push(5);

ringInt.Sort();

// view

ringInt.Print();

ringInt.Filtered(4).Print();

ringInt.Filter(ringInt.Even).Print();

// find

Console.WriteLine("Find:");

ringInt?.Get(6).Print();

// remove

ringInt?.Delete(1);

// find

Console.WriteLine("Find: ");

ringInt?.Get(6).Print();

// view

Console.WriteLine("");

ringInt?.Print();

Console.WriteLine("Work with Ring string model:\n add - veiw - find - remove - find - veiw");

// add

ringString.push("Hello");

ringString.push("World");

ringString.push("!!");

// view

ringString.Print();

// find

Console.WriteLine("Find:");

ringString?.Get("World")?.Print();

ringString?.GetById(1)?.Print();

// remove

ringString?.Delete("World");

// find

Console.WriteLine("Find: ");

ringString?.Get("World")?.Print();

// view

Console.WriteLine("");

ringString?.Print();

Console.WriteLine("Work with Ring double model:\n add - veiw - find - remove - find - veiw");

// add

ringDouble.push(1.1);

ringDouble.push(2.2);

ringDouble.push(3.3);

// view

ringDouble.Print();

// find

Console.WriteLine("Find:");

ringDouble?.Get(2.2).Print();

// remove

ringDouble?.Delete(2.2);

// find

Console.WriteLine("Find: ");

ringDouble?.Get(2.2)?.Print();

// view

Console.WriteLine("");

ringDouble?.Print();

Console.WriteLine("===============================================");

ringInt?.Print();

Console.WriteLine("Sort");

ringInt?.Sort();

ringInt?.Print();

Console.WriteLine("Reverse");

ringInt?.Reverse();

ringInt?.Print();

Console.WriteLine("Max");

ringInt?.Max()?.Print();

Console.WriteLine("Replace");

ringInt?.Replace(1, 10);

ringInt?.Print();

Console.WriteLine("Delete");

ringInt?.Delete(10);

ringInt?.Print();

Console.WriteLine("===============================================");

ringDouble?.Print();

Console.WriteLine("Sort");

ringDouble?.Sort();

ringDouble?.Print();

Console.WriteLine("Reverse");

ringDouble?.Reverse();

ringDouble?.Print();

Console.WriteLine("Max");

ringDouble?.Max().Print();

Console.WriteLine("Replace");

ringDouble?.Replace(1, 10);

ringDouble?.Print();

Console.WriteLine("Delete");

ringDouble?.Delete(10);

ringDouble?.Print();

Console.WriteLine("===============================================");

ringString?.Print();

Console.WriteLine("Sort");

ringString?.Sort();

ringString?.Print();

Console.WriteLine("Reverse");

ringString?.Reverse();

ringString?.Print();

Console.WriteLine("Max");

ringString?.Max().Print();

Console.WriteLine("Replace");

ringString?.Replace("Hello", "Hi");

ringString?.Print();

Console.WriteLine("Delete");

ringString?.Delete("Hi");

ringString?.Print();

Console.WriteLine("======================================");

}

private static void AddEventMethod(object sender, int i)=>Console.WriteLine($"Element {i} added");

private static void AddEventMethod(object sender, string i)=>Console.WriteLine($"Element {i} added");

private static void AddEventMethod(object sender, double i)=>Console.WriteLine($"Element {i} added");

private static void RemoveEventMethod(object sender, int i) => Console.WriteLine($"Element {i} removed");

private static void RemoveEventMethod(object sender, string i) => Console.WriteLine($"Element {i} removed");

private static void RemoveEventMethod(object sender, double i) => Console.WriteLine($"Element {i} removed");

}

class Ring<T> where T: IComparable<T>

{

internal class Node

{

private T \_data;

private Node \_next;

private Node \_prev;

public Node(T value)

{

\_data = value;

\_next = null;

\_prev = null;

}

public Node(T value, Node next, Node prev)

{

\_data = value;

\_next = next;

\_prev = prev;

}

public T getData()

{

return \_data;

}

public Node getNext()

{

return \_next;

}

public Node getPrev()

{

return \_prev;

}

public void setNext(Node Next)

{

this.\_next = Next;

}

public void setPrev(Node Prev)

{

this.\_prev = Prev;

}

public void setData(T data)

{

this.\_data = data;

}

public void Print()

{

Console.WriteLine(\_data);

}

}

private Node \_head;

private Node \_tail;

private int \_size;

public Ring()

{

\_size = 0;

\_head = null;

\_tail = null;

}

public int getSize()

{

return this.\_size;

}

public void push(T value)

{

bool hasDuplicates = FindDuplicates(value);

if (hasDuplicates)

return;

if (\_size == 0)

{

\_head = new Node(value);

\_tail = \_head;

\_head.setNext(\_tail);

\_head.setPrev(\_tail);

}

else

{

Node newNode = new Node(value, \_head, \_tail);

\_tail.setNext(newNode);

\_head.setPrev(newNode);

\_tail = newNode;

}

\_size++;

ElementAdded?.Invoke(this, value);

}

private bool FindDuplicates(T value)

{

Node current = \_head;

for (int i = 0; i < \_size; i++)

{

if (current.getData().CompareTo(value) == 0)

return true;

current = current.getNext();

}

return false;

}

public void Print()

{

Node current = \_head;

for(int i = 0; i < \_size; i++)

{

Console.WriteLine(current.getData());

current = current.getNext();

}

}

public delegate void SortDelegate();

public SortDelegate Sort;

public void BubleSort()

{

try

{

for (int i = 0; i < \_size; i++)

{

Node current = \_head;

for (int j = 0; j < \_size - i - 1; j++)

{

if (current.getData().CompareTo(current.getNext().getData()) > 0)

{

T temp = current.getData();

current.setData(current.getNext().getData());

current.getNext().setData(temp);

}

current = current.getNext();

}

}

}

catch (Exception e)

{

Console.WriteLine("May be invalid type\n" + e.Message);

return;

}

}

public void Reverse()

{

Node current = \_head;

for(int i = 0; i < \_size; i++)

{

Node temp = current.getPrev();

current.setPrev(temp.getNext().getNext());

current.setNext(temp);

temp = null;

current = current.getNext();

}

Node temp1 = \_head;

\_head = \_tail;

\_tail = temp1;

temp1 = null;

}

public void Delete(T value)

{

try

{

Node current = \_head;

for (int i = 0; i < \_size; i++)

{

if (current.getData().CompareTo(value) == 0)

{

current.getPrev().setNext(current.getNext());

current.getNext().setPrev(current.getPrev());

current = null;

if (current == \_head)

{

\_head = current.getNext();

}

if (current == \_tail)

{

\_tail = current.getPrev();

}

\_size--;

ElementRemoved?.Invoke(this, value);

return;

}

current = current.getNext();

}

}

catch (Exception e)

{

Console.WriteLine("May be invalid type\n" + e.Message);

return;

}

}

public void Delete(int index)

{

Node node = \_head;

for(int i = 0; i < index; i++)

{

node = node.getNext();

}

node.getPrev().setNext(node.getNext());

node.getNext().setPrev(node.getPrev());

if (node == \_head)

{

\_head = node.getNext();

}

if (node == \_tail)

{

\_tail = node.getPrev();

}

ElementRemoved?.Invoke(this, node.getData());

node = null;

\_size--;

}

public Node GetByValue(T value)

{

try

{

Node current = \_head;

for (int i = 0; i < \_size; i++)

{

if (current.getData().CompareTo(value) == 0)

{

return current;

}

current = current.getNext();

}

return null;

}

catch (Exception e)

{

Console.WriteLine("May be invalid type\n" + e.Message);

return null;

}

}

public Node GetByIndex(int index)

{

Node current = \_head;

for(int i = 0; i < index; i++)

{

current = current.getNext();

}

return current;

}

public Node Max()

{

try

{

Node current = \_head;

Node max = current;

for (int i = 0; i < \_size; i++)

{

if (current.getData().CompareTo(max.getData()) > 0)

{

max = current;

}

current = current.getNext();

}

return max;

}

catch (Exception e)

{

Console.WriteLine("May be invalid type\n" + e.Message);

return null;

}

}

public void Replace(T valueOfReplace, T valueToReplace)

{

try

{

Node current = \_head;

for (int i = 0; i < \_size; i++)

{

if (current.getData().CompareTo(valueOfReplace) == 0)

{

current.setData(valueToReplace);

}

current = current.getNext();

}

}

catch (Exception e)

{

Console.WriteLine("May be invalid type\n" + e.Message);

}

}

public Ring<T> FilteredByValue(T ruleValue)

{

Ring<T> temp = new Ring<T>();

Node current = \_head;

for (int i = 0; i < \_size; i++)

{

if (current.getData().Equals(ruleValue))

{

temp.push(current.getData());

}

}

return temp;

}

public Ring<T> Filter(Func<T,bool> predicate)

{

Ring<T> temp = new Ring<T>();

Node current = \_head;

for (int i = 0; i < \_size; i++)

{

if (predicate(current.getData()))

{

temp.push(current.getData());

}

current = current.getNext();

}

return temp;

}

public bool Even(T value) => value.GetHashCode() % 2 == 0;

public bool Odd(T value) => value.GetHashCode() % 2 != 0;

// delegates

public delegate Node GetbyIdDelegate(int index);

public GetbyIdDelegate GetById;

public delegate Node GetbyValueDelegate(T value);

public GetbyValueDelegate Get;

public delegate Ring<T> FilterDelegate(T rulevalue);

public FilterDelegate Filtered;

// events

//add

public delegate void ElemAddDelegate();

public event EventHandler<T> ElementAdded;

//remove

public delegate void ElemRemoveDelegate();

public event EventHandler<T> ElementRemoved;

}

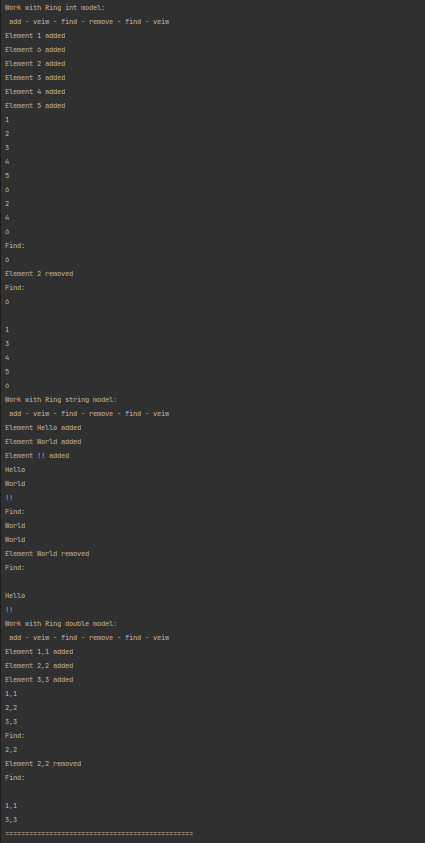
Результат работы программы представлен на рисунке 1.1 – 1.2

Рисунок 1.1 – Результат выполнения программы 1

# 

Рисунок 1.2 – Результат выполнения программы 1

Вывод: В ходе изучения темы "Делегаты, события и лямбды" в рамках лабораторной работы, были успешно достигнуты основные поставленные цели. Основная цель заключалась в ознакомлении с использованием делегатов в приложениях, научении описывать собственные события и познакомлении с механизмом обработки событий.

В ходе работы стало ясно, что делегаты представляют собой мощный инструмент для передачи функциональности между различными частями программы. Понимание их принципов использования позволяет создавать более гибкие и масштабируемые приложения, а также упрощает процесс взаимодействия между компонентами.

Освоение описания собственных событий позволяет создавать архитектуру программы, основанную на принципах асинхронного взаимодействия. Это существенно улучшает читаемость и структурированность кода, делая его более понятным и легко поддерживаемым.

Изучение механизма обработки событий позволило эффективно реагировать на изменения состояния программы и пользовательские действия. Приобретенные навыки позволят эффективно реализовывать различные функциональные возможности, такие как управление интерфейсом пользователя, обработка ввода и вывода данных и т. д.

В целом, выполнение лабораторной работы позволило углубить понимание принципов работы с делегатами, событиями и лямбдами, а также приобрести практические навыки, которые окажутся весьма полезными при разработке сложных программных продуктов.