**6 ЛАБОРАТОРНАЯ РАБОТА №6**

## **«Исследование возможностей работы с данными. LINQ, анонимные методы, лямбда-выражения.»**

**6.1 Цель работы**

Исследовать свойства механизма событий в С#, изучить особенности их применения и приобрести практические навыки их использования в программе.

**6.2 Индивидуальный вариант**

Система Поставок. Создать классы поставщик товараиработник склада. Когда поставщик привозит товар или работник склада принимает этот товар должны вызываться соответствующие события.

**6.3 Ход выполнения работы**

В начале лабораторной работы были изучены методические указания. Далее был написан класс шины событий, код которого содержится в листинге 6.1.

Листинг 6.1 – Код шины событий

public class EventBus

{

private readonly Dictionary<Type, List<Delegate>>\_handlers = new();

public void Subscribe<TEvent>(Delegate handler) where TEvent : EventArgs

{

var eventType = typeof(TEvent);

if (!\_handlers.ContainsKey(eventType))

\_handlers.Add(eventType, new List<Delegate>());

\_handlers[eventType].Add(handler);

}

public void Unsubscribe<TEvent>(Delegate handler) where TEvent : EventArgs

{

var eventType = typeof(TEvent);

if (\_handlers.ContainsKey(eventType) && \_handlers[eventType].Contains(handler))

\_handlers[eventType].Remove(handler);

}

public void Publish<TEvent>(object sender, TEvent eventArgs) where TEvent : EventArgs

{

Type eventType = typeof(TEvent);

if (\_handlers.ContainsKey(eventType))

{

foreach (var handler in \_handlers[eventType])

{

handler.DynamicInvoke(sender, eventArgs);

}

}

}

}

Далее были написаны классы событий отправки и принятия товара, что показано на листингах 6.2–6.3.

Листинг 6.2 – Событие отправки продукта

public class ProductSended : EventArgs

{

public string Product { get; set; }

}

Листинг 6.3 – Событие принятия продукта

public class ProductAccepted : EventArgs

{

public string Product { get; set; }

}

Также были написаны классы поставщика и работника склада, что показано на листингах 6.4–6.5.

Листинг 6.4 – Класс поставщика

public class Distributor(EventBus eventBus)

{

public void Distribute(string product)

{

Console.WriteLine($"Distributing product: {product}");

eventBus.Publish(this, new ProductSended{Product = product});

}

}

Листинг 6.5 – Класс работника

public class Worker

{

private readonly EventBus \_eventBus;

public Worker(EventBus eventBus)

{

\_eventBus = eventBus;

eventBus.Subscribe<ProductSended>(AcceptProduct);

eventBus.Subscribe<ProductAccepted>(FinishProduct);

}

~Worker()

{

\_eventBus.Unsubscribe<ProductSended>(AcceptProduct);

\_eventBus.Unsubscribe<ProductAccepted>(FinishProduct);

}

private void AcceptProduct(object sender, ProductSended product)

{

Console.WriteLine($"Product accepting: {product.Product}");

\_eventBus.Publish(this, new ProductAccepted{Product = product.Product});

}

private void FinishProduct(object sender, ProductAccepted product)

{

Console.WriteLine($"Product accepted: {product.Product}");

}

}

Далее программа была протестирована, что показано на рисунке 6.1.

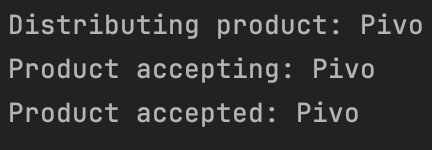


Рисунок 6.1 – Тест программы

**Выводы**

В начале выполнения работы были изучены методические указания. Далее были исследованы свойства механизма событий в С#, также были изучены особенности их применения и были приобретены практические навыки их использования в программе. В конце выполнения лабораторной работы блы написан отчет.