ФГАОУ ВО «КФУ им. В.И. Вернадского»

Физико-технический институт (структурное подразделение)

|  |
| --- |
|  |

Кафедра компьютерной инженерии и моделирования

Скибинский Дмитрий Константинович

отчет по практической работе №6  
по дисциплине **«ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ»**

Направление подготовки:

09.03.04 "Программная инженерия"

Оценка - 100



Симферополь, 2024

**Практическая работа №6  
Тема: Делегаты, события, лямбда выражения,**

**многопоточное и асинхронное программирование**

**Цель работы:** Научиться на практике использованию делегатов, лямбда-

выражений и событий. Научиться создавать собственные события.

**Описание ключевых понятий:**

**Делегат**— это тип, который представляет ссылку на методы с определенной сигнатурой. Он используется для передачи методов в качестве аргументов другим методам.

**Мультикастовый делегат** — это делегат, который может ссылаться на несколько методов одновременно. Он поддерживает операции добавления (+=) и удаления (-=) методов.

**Callback** — это метод, который передается другому методу и вызывается после завершения определенной операции.

+= — добавляет метод в цепочку делегатов.

-= — удаляет метод из цепочки делегатов.

**Лямбда-выражение** — это анонимная функция, которая может использоваться для создания делегатов или выражений запросов.

**Событие** — это механизм, который позволяет классу уведомлять другие классы о возникновении определенного действия.

**EventHandler** — это предопределенный делегат, который используется для обработки событий. Он принимает два параметра: объект-отправитель (sender) и объект EventArgs.

**Sender** — объект, который инициирует событие.

**Receivers** — объекты, которые подписываются на событие и обрабатывают его.

**EventArgs** — базовый класс для передачи данных в событиях. Его потомки (например, MouseEventArgs, KeyEventArgs) используются для передачи специфических данных.

**Thread** — это объект, представляющий поток выполнения. Потоки позволяют выполнять код параллельно.

**lock** — синхронизирует доступ к разделяемым ресурсам. **Join** — ожидает завершения потока.

**Wait** и **Pulse** — используются для синхронизации потоков с помощью объектов блокировки.

**Mutex** — примитив синхронизации, который может использоваться для защиты ресурсов в разных процессах. **Semaphore** — ограничивает количество потоков, которые могут одновременно обращаться к ресурсу.

**TryEnter** — пытается получить блокировку. **Priority** — устанавливает приоритет потока. **Background** — указывает, является ли поток фоновым.

**Start** — запускает поток. **IsAlive** — проверяет, выполняется ли поток.

**Barrier** — синхронизирует потоки, заставляя их ждать друг друга на определенном этапе.

**TPL** (Task Parallel Library) — библиотека для параллельного программирования.

**Task** — представляет асинхронную операцию.

**Parallel** — предоставляет методы для параллельного выполнения циклов.

**async** и **await** — ключевые слова для асинхронного программирования.

**Перед выполнением лабораторной работы изучена следующая литература:**

1. Изучить презентации лектора курса: «Делегаты», «События», «Многопоточное

программирование» (материалы доступны в "облаке" на Mail.ru и в Moodle КФУ).

2. Сайт Metanit.com

3. Справочник по C#. Корпорация Microsoft.

http://msdn.microsoft.com/ru-ru/library/618ayhy6.aspx

4. Биллиг В.А. Основы программирования на C#. Интернет-университет

информационных технологий. http://www.intuit.ru/studies/courses/2247/18/info

5. Павловская Т. Программирование на языке высокого уровня C#.

http://www.intuit.ru/studies/courses/629/485/info

6. Руководство по программированию на C#. Корпорация Microsoft.

http://msdn.microsoft.com/ru-ru/library/67ef8sbd.aspx

7. Корпорация Microsoft. C#. Спецификация языка. (Приложение А, Комментарии к

документации).

**Задание 1. Использование событий**

Возьмем код программы из Лабораторной работы 5, с использованием интерфейса. Задача предполагает автоматический подсчет значений и уведомление клиента (пользовательского интерфейса) о новых измерениях.

Поэтому мы можем убрать не нужные кнопки из пользовательского интерфейса для получения результата, так как он будет показываться автоматически с изменениями значений

Изображение выглядит как текст, снимок экрана, программное обеспечение, дисплей

Автоматически созданное описание

Добавим интерфейс

Изображение выглядит как текст, Шрифт, Мультимедийное программное обеспечение, программное обеспечение

Автоматически созданное описание

Новый интерфейс IEventEnabledMeasuringDevice является расширением существующего интерфейса IMeasuringDevice. Он добавляет поддержку событий, что позволяет классам, реализующим этот интерфейс, уведомлять другие части приложения о новых измерениях.

IEventEnabledMeasuringDevice наследует все методы и свойства из IMeasuringDevice. Событие NewMeasurementTaken типа EventHandler позволяет уведомлять подписанные методы о том, что устройство произвело новое измерение.

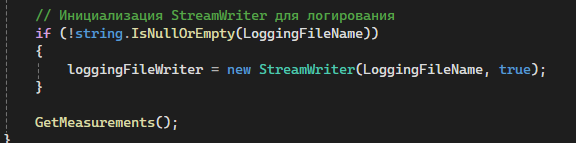
Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание**BackgroundWorker** используется для асинхронного сбора данных от устройства. Сначала инициализируем его

Добавим обработчик

Изображение выглядит как текст, снимок экрана, программное обеспечение, дисплей

Автоматически созданное описание

Сделаем инициализацию логирования

Добавим логирование в методе

Изображение выглядит как текст, снимок экрана, программное обеспечение, Шрифт

Автоматически созданное описаниеДобавим закрытие файла лога

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Освободим ресурсы

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Сделай файл логирования

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Автоматически созданное описание

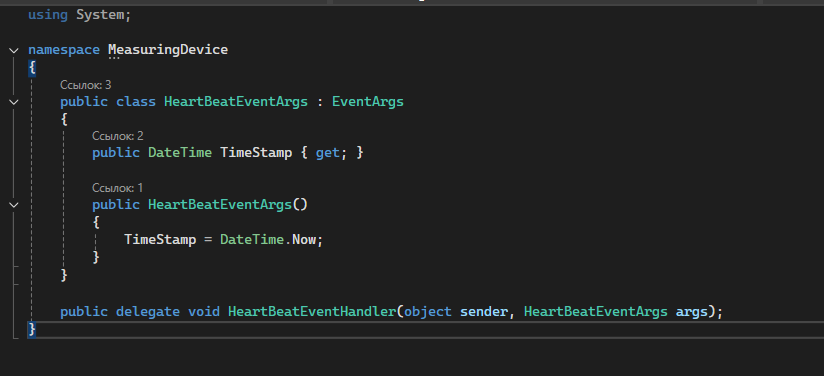
Изображение выглядит как текст, линия, снимок экрана

Автоматически созданное описаниеТеперь наша программа автоматически подсчитывает значения Metric Value и Imperial Value

**Задание 2.** Использование лямбда-выражений

Задание 2 предполагает добавление поддержки события **HeartBeat** в приложение. Это событие будет генерироваться с определенным интервалом, чтобы уведомлять клиента о том, что устройство "живое". Будем использовать **лямбда-выражения** для обработки событий и **BackgroundWorker** для генерации пульса.

Этот класс будет использоваться для передачи данных о времени пульса.

Этот делегат будет использоваться для обработки события HeartBeat

Изображение выглядит как текст, программное обеспечение, Мультимедийное программное обеспечение, снимок экрана

Автоматически созданное описаниеВ интерфейс добавим событие HeartBeat и свойство HeartBeatInterval в интерфейс

Добавим **HeartBeat**и**HeartBeatInterval**вкласс**MeasureDataDevice**

Изображение выглядит как текст, снимок экрана, программное обеспечение

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описаниеКонструктор класса MeasureLengthDevice отвечает за инициализацию объекта класса. Конструктор принимает параметр heartBeatInterval, который определяет интервал времени (в миллисекундах) между генерацией события HeartBeat. Это значение передается в базовый класс через свойство HeartBeatInterval.

Изображение выглядит как текст, линия, снимок экрана

Автоматически созданное описаниеПример работы программы

**Задание 3. Создание многопоточных приложений и асинхронных**

**методов.**

Изображение выглядит как снимок экрана, компьютер, программное обеспечение, дисплей

Автоматически созданное описаниеСделаем приложение, которое моделирует движение двух шариков (Ball) на холсте (Canvas). Шарики могут перемещаться, отскакивать от границ холста и сталкиваться друг с другом.

**Метод StartButton\_Click**

Этот метод обрабатывает нажатие кнопки "Start". Он инициализирует шарики, добавляет их на холст и запускает их движение. Происходит валидация введенных значений скорости. Инициализация шариков и добавления их на холст. Запуск движения шаров. Используется асинхронный запуск с помощью Task.Run. Движение шариков выполняется параллельно с использованием Parallel.For.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

**Метод MoveBall(Ball ball)**

Этот метод отвечает за перемещение шарика и обновление его позиции на холсте.

**Цикл движения:** Метод работает в бесконечном цикле, пока флаг isRunning равен true. Внутри цикла вызывается метод ball.Move(), который обновляет координаты шарика.

**Обновление позиции на холсте:** Для обновления позиции шарика на холсте используется Dispatcher.Invoke. Это необходимо, так как обновление UI должно выполняться в основном потоке.

Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

Автоматически созданное описание**Задержка:** Используется Thread.Sleep(30) для замедления движения шариков. Это заменяет Task.Delay, так как Parallel.For работает с потоками, а не с задачами.

**Метод UpdateBallPosition(Ball ball)**

Этот метод обновляет позицию шарика на холсте.

**Поиск элемента Ellipse:** Метод находит элемент Ellipse, соответствующий шарику, по цвету.

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание**Обновление позиции:** Если элемент найден, его позиция на холсте обновляется с использованием методов Canvas.SetLeft и Canvas.SetTop.

По поводу физической составляющей программы. Используется отражения от сторон поля

Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

Автоматически созданное описание

Изображение выглядит как текст, Шрифт, снимок экрана, белый

Автоматически созданное описаниеПо закону

Измеряем расстояние между шарами

Изображение выглядит как текст, Мультимедийное программное обеспечение, снимок экрана

Автоматически созданное описаниеРасчитыванием сталкивание шаров

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, Шрифт, алгебра

Автоматически созданное описание

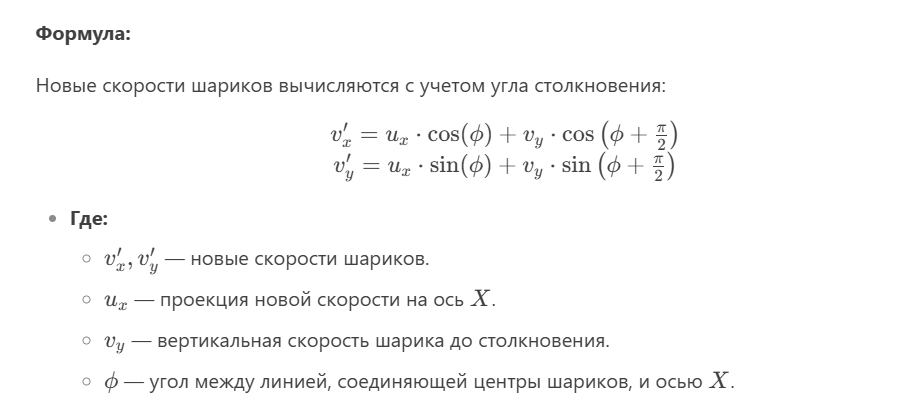
Вычисляем новые скорости шаров после столкновения

Изображение выглядит как снимок экрана, текст, Шрифт

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, Шрифт, документ

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описаниеИзображение выглядит как текст, снимок экрана, алгебра, Шрифт

Автоматически созданное описаниеОбновление скоростей

Мьютекс используется в методе MoveBall для синхронизации доступа к списку шариков (balls).

Если не использовать мьютекс, то два шарика могут одновременно пытаться изменить свои координаты или взаимодействовать друг с другом, что приведет к некорректному поведению. Например:

* Один шарик может перекрыть другой, неправильно вычислив расстояние между ними.
* Координаты шариков могут быть повреждены из-за одновременного доступа.

В методе StartButton\_Click вы запускаете движение шариков асинхронно с помощью Task.Run

Если выполнять движение шариков в основном потоке, то приложение "зависнет", пока шарики двигаются. Использование Task.Run позволяет выполнять движение в фоновом потоке.

**Параллельное выполнение:** Вы используете Parallel.For, чтобы запустить движение каждого шарика в отдельном потоке. Это ускоряет выполнение, так как шарики двигаются независимо друг от друга.

**async и await** для запуска движения шариков асинхронно

**Ответы на вопросы:**

**1. Что такое delegate?**

**Delegate** — это тип, который представляет ссылку на методы с определенной сигнатурой. Он используется для передачи методов в качестве аргументов другим методам.

**2. Что такое функции обратного вызова (callback)?**

**Функция обратного вызова (callback)** — это функция, которая передается другому методу и вызывается после завершения определенной операции.

**3. Как строятся цепочки делегатов?**

Цепочки делегатов создаются с помощью операторов += (добавление метода в цепочку) и -= (удаление метода из цепочки).

**4. Что такое анонимная функция? Приведите пример.**

**Анонимная функция** — это функция, которая не имеет имени и объявляется "на лету". В C# анонимные функции могут быть реализованы с помощью лямбда-выражений или анонимных методов.

Action<string> anonymousFunction = (message) => Console.WriteLine(message);

anonymousFunction("Hello from anonymous function!");

**5. Напишите пример лямбда-выражения.**

**Лямбда-выражение** — это компактный способ записи анонимной функции.

**Пример:**

Func<int, int, int> add = (x, y) => x + y;

Console.WriteLine(add(5, 3)); // Вывод: 8

**6. Связывание обработчика с событием.**

Обработчик события связывается с событием с помощью оператора +=.

**7. Отключение обработчика. Динамическое связывание событий с их обработчиками.**

Обработчик отключается с помощью оператора -=. Динамическое связывание и отключение обработчиков позволяет управлять событиями во время выполнения программы.

**8. Как реализуется блокировка потока?**

Блокировка потока реализуется с помощью ключевого слова lock или объектов синхронизации, таких как Mutex и Semaphore.

**9. Что такое мьютекс и семафор?**

**Мьютекс (Mutex):**

1)Используется для синхронизации доступа к общим ресурсам.

2)Позволяет только одному потоку входить в критическую секцию.

**Семафор (Semaphore):**

Позволяет ограниченному числу потоков входить в критическую секцию.

**10. В чем преимущества использования новой парадигмы многопоточности в C# с использованием библиотеки TPL?**

TPL (Task Parallel Library) — это библиотека для параллельного программирования в C#. Ее преимущества:

1.Упрощение многопоточности: TPL автоматически управляет потоками, что упрощает написание параллельного кода.

2. Высокая производительность: TPL оптимизирует использование потоков, что повышает производительность.

3. Поддержка асинхронности: TPL поддерживает асинхронное программирование с использованием Task и async/await.

**11. Что такое Parallel.For?**

Parallel.For — это метод из TPL, который позволяет выполнять цикл параллельно.

**12. Что означают термины async и await?**

**async** — ключевое слово, которое указывает, что метод является асинхронным.

**await** — оператор, который приостанавливает выполнение метода до завершения асинхронной операции.

**Представлены 4 проекта, реализованных в Visual Studio Community 2022. Проекты представлены преподавателю в электронной форме, продемонстрирована их работоспособность, разъяснены детали программного кода.**