Упражнение 2: Использование Лямбда-выражений

В этом упражнении вы объявите новый тип делегата и новый тип EventArgs для поддержки события HeartBeat. Интерфейс IMeasuringDevice и Класс MeasureDataDevice будут изменены для создания пульсации с помощью объекта BackgroundWorker. Вы зададите код для запуска в новом потоке с помощью лямбда-выражения.

В обработчике событий ReportProgress необходимо указать код для уведомления клиентского приложения другим лямбда выражением.

Событие HeartBeat в приложении WPF будет обрабатываться с помощью лямбда-выражения.

Основными задачами данного упражнения являются:

1. Откройте решение события.

2. Определите новый класс EventArgs для поддержки событий пульса.

3. Объявите новый тип делегата.

4. Обновите интерфейс IEventEnabledMeasuringDevice.

5. Добавьте событие HeartBeat и свойство HeartBeatInterval в класс MeasureDataDevice.

6. Используйте объект BackgroundWorker для создания пульса.

7. Вызвать метод StartHeartBeat, когда объект MeasureDataDevice начинает работать.

8. Dispose объекта BackgroundWorker heartBeatTimer, когда объект MeasureDataDevice разрушается.

9. Обновите конструктор для класса MeasureMassDevice.

10. Обрабатывать событие пульса в пользовательском интерфейсе.

11. Протестируйте решение.

Задача 1. Откройте решение "события"

\* Откройте решение события в Ex2\Starter.

Примечание: события, решения в папке Ex2 функционально такой же, как код, который Вы выполнили в упражнении 1, однако, он включает в себя обновленный список задач, чтобы Вы смогли выполнить это упражнение.

Задание 2: определение нового класса eventargs для поддержки событий пульсации

1 в проекте MeasuringDevice добавьте новый файл кода с именем HeartBeatEvent.cs.

2 в файле кода добавьте директиву using для переноса пространства имен System в область текущего файла.

3 Определите новый класс с именем HeartBeatEventArgs в пространстве имен MeasuringDevice. Класс должен расширять класс EventArgs (т.е. быть его наследником).

Примечание: пользовательский класс аргументов события может содержать любое количество свойств; эти свойства хранят информацию, где событие вызывается, позволяя обработчику события получать специфичные для события сведения, когда событие обрабатывается.

4 в классе HeartBeatEventArgs добавьте доступное только для чтения автоматическое свойство DateTime с именем TimeStamp.

5 добавьте конструктор в класс HeartBeatEventArgs. Конструктор должен быть без параметров, и инициализировать свойство **TimeStamp** значением даты и времени создания класса. Конструктор также должен расширять конструктор базового класса.

Задача 3. объявление нового типа делегата

• Под классом HeartBeatEventArgs объявите **public** делегат с именем HeartBeatEventHandler. Делегат должен ссылаться на метод, который не возвращает значение, но имеет следующие параметры:

a. Параметр объекта с именем sender.

b. Параметр HeartBeatEventArgs с именем args.

Задача 4: Обновление интерфейса IEventEnabledMeasuringDevice

1 в списке задач найдите задачу **TODO - Define the new event in the interface**  и дважды щелкните эту задачу. Эта задача находится в интерфейсе IEventEnabledMeasuringDevice

2 удалите этот комментарий и добавьте в интерфейс событие HeartBeat. Событие должно указать, что подписчики используют тип делегата HeartBeatEventHandler, чтобы указать метод для запуска при возникновении события.

3 Удалите комментарий **TODO - Define the HeartBeatInterval member in the interface**, а затем добавьте свойство read–only HeartBeatInterval к интерфейсу.

Задание 5: Добавить событие пульсация и свойство HeartBeatInterval к классу MeasureDataDevice

1 в списке задач найдите задачу **TODO - Add the HeartBeatInterval property** , а затем дважды щелкните эту задачу. Эта задача находится в классе MeasureDataDevice.

2 Удалите **TODO - Add the HeartBeatInterval**, и добавьте protected целочисленный член с именем namedheartBeatIntervalTime.

3 добавьте код для реализации открытого целочисленного свойства HeartBeatInterval, которое определяет интерфейс IEventEnabledMeasuringDevice. Свойство должно возвращать значение элемента heartBeatInterval при вызове метода доступа get. Свойство должно иметь метод доступа private set, чтобы конструктор мог задать свойство.

4 удалите **TODO - Add the HeartBeat event** и добавьте событие HeartBeat, которое определяет интерфейс IEventEnabledMeasuringDevice.

5 удалите **TODO - add the OnHeartBeat method** **to fire the event** , и добавьте protected virtual  void метод с именем OnHeartBeat, который не принимает параметров.

6 в методе OnHeartBeat, добавьте код для выполнения следующих действий:

a. проверка наличия подписчиков у события **HeartBeat** .

b. если событие имеет подписчиков, вызовите событие, передав текущий объект и новый экземпляр объекта HeartBeatEventArgs в качестве параметров.

Задание 6: использовать объект BackgroundWorker для имитации пульсации

1 удалите **TODO - Declare the BackgroundWorker to generate the heartbeat**, а затем определите объект private BackgroundWorker с именем heartBeatTimer.

2 удалите комментарий **TODO - Create a method to configure the BackgroundWorker using Lambda Expressions** и объявите private метод с именем StartHeartBeat, который не принимает параметры и не возвращает значение.

3 в методе StartHeartBeat добавьте код для выполнения следующих действий:

а. Создайте объекта **heartBeatTimer**  экземпляр класса BackgroundWorker.

b. Настройте объект heartBeatTimer для поддержки отмены.

c. Настройте объект heartBeatTimer для поддержки уведомления о ходе выполнения.

4 добавьте обработчик для события heartBeatTimer DoWork с помощью лямбда-выражения для определения действий, которые необходимо выполнить. Лямбда-выражение должно принимать два параметра (использовать имена и аргументы). В теле lambda выражения добавьте цикл while, который постоянно повторяется и содержит код для выполнения следующих действий:

1. Используйте статический метод **Thread.Sleep**, чтобы поставить текущий поток на сон в течение времени, которое указано в свойстве HeartBeatInterval.

б проверьте значение свойства **disposed**. Если значение true, завершите цикл.

в вызовите метод heartBeatTimer.ReportProgress, передавая ноль в качестве параметра.

Примечание: используйте + = составной оператор присваивания, чтобы указать, что метод будет обрабатывать событие DoWork , определите лямбда-выражение.

5 Добавьте обработчик для события heartBeatTimer.ReportProgress с помощью другого лямбда-выражения. В теле лямбда-выражения добавьте код для вызова метода OnHeartBeat, который вызывает событие HeartBeat.

6 В конце метода StartHeartBeat, добавьте строку кода, чтобы запустить объекта heartBeatTimer BackgroundWorker асинхронно.

Задание 7: вызвать метод StartHeartBeat, когда объект MeasureDataDevice начинает работать

1. В списке задач найдите задачу **TODO - Call StartHeartBeat() from StartCollecting method**, а затем дважды кликните на задачу. Эта задача находится в методе StartCollecting.

2. Удалите этот комментарий и добавьте строку кода для вызова метода StartHeartBeat.

Задание 8: Освобождение ресурсов объекта heartBeatTimer BackgroundWorker, когда объект MeasureDataDevice разрушается

1 В списке задач найдите **TODO - dispose of the heartBeatTimer BackgroundWorker**, а затем дважды кликните на эту задачу. Эта задача находится в методе Dispose.

2 удалите комментарий и добавьте код, чтобы убедиться, что объект Heartbeattimer BackgroundWorker не равен null. Если объект heartBeatTimer не равен null, вызовите метод Dispose объекта BackgroundWorker.

Теперь обновлен абстрактный класс MeasureDataDevice для реализации обработчиков событий с помощью лямбда выражений. Чтобы приложение могло воспользоваться этими изменениями, необходимо изменить класс MeasureMassDevice, который расширяет класс MeasureDataDevice.

Задание 9: обновление конструктора для класса MeasureMassDevice

1 Откройте файл класса MeasureMassDevice.

2 в начале класса измените сигнатуру конструктора, чтобы получить дополнительное целочисленное значение heartBeatInterval.

3 измените тело конструктора, чтобы сохранить значение элемента HeartBeatInterval в heartBeatIntervalmember.

4 под существующим конструктором удалите комментарий **TODO – Add a chained constructor that calls the previous constructor**, и добавьте второй конструктор, который принимает следующие параметры:

а. **Units**, названных deviceUnits.

b. **String** строки с именем logFileName.

5 измените конструктор new для неявного вызова существующего конструктора. Передайте значение 1000 в качестве значения параметра heartBeatInterval.

Задание 10: обрабатывать событие пульсация в пользовательском интерфейсе

1 в списке задач найдите задачу **TODO - Use a lambda expression to handle the HeartBeat event in the UI**  и дважды щелкните эту задачу. Эта задача находится в методе startCollecting\_Click файла MainWindow.cs.

2 удалите комментарий и добавьте лямбда-выражение для обработки cобытия **device**.**HeartBeat**. Лямбда выражение принимает два параметра (name them *o* and *args*). . В теле лямбда-выражения добавьте код для обновления метки theheartBeatTimeStamp текстом "HeartBeat Timestamp: timestamp", где timestamp-значение свойства **args.TimeStamp**.

Подсказка: Установите значение свойство метки **Content**, чтобы изменить текст, который отображается метка.

Задача 11: тестирование решения

1 постройте проект и исправьте все ошибки.

2 Запустите приложение.

3 Нажмите **Start** **Collecting** и убедитесь, что значения начинают отображаться как раньше. Также обратите внимание, что значение **HeartBeat** **Timestamp** обновляется один раз в секунду.

4 Нажмите кнопку **Stop** **Collecting** и убедитесь, что список RawData больше не обновляется. Обратите внимание, что метка времени продолжает обновляться, так как код не завершает мигание метки времени при остановке сбора.

5 нажмите кнопку **Dispose Object** и убедитесь, что метка времени больше не обновляется.

6 закройте приложение и вернитесь в Visual Studio.

7 Закройте Visual Studio.