# ООП в C#

## (события, chatbot: проектирование архитектуры)

Андрей Голяков

# События (events)

Событие — автоматическое уведомление о том, что произошло некоторое действие.

События действуют по следующему принципу: объект, проявляющий интерес к событию, регистрирует обработчик этого события. Когда же событие происходит, вызываются все зарегистрированные обработчики этого события. Обработчики событий обычно представлены делегатами.


# Объявление событий

События являются членами класса и объявляются с помощью ключевого слова event.

Чаще всего для этой цели используется следующая форма:

```
модификатор_доступа event делегат_события имя_события;
```

например:

```
public event WorkPerformedEventHandler WorkPerformed;
```

где, WorkPerformedEventHandler - это делегат.

# Запуск событий

Вызов события - это основополагающая операция в концепции событий.

Если событие не вызвать, то все, кто его ждут, все подписавшиеся на него слушатели, так и не узнают, что оно произошло.

Вызов события происходит тем же способом, что и вызов делегата, вы вызываете его как метод.

```
if (WorkPerformed != null) {
    WorkPerformed(8, WorkType.Work);
}
```

Прежде чем вызвать событие, необходимо посмотреть, есть хоть что-то в списке вызовов у нашего события, иначе получится, что вызывать нам совершенно нечего. Помните, что делегат это ссылка на метод. И запуская на выполнение делегат, мы на самом деле вызываем метод, однако если список вызовов пуст, произойдёт исключение.

# Запуск событий

Вызов события - это основополагающая операция в концепции событий.

Если событие не вызвать, то все, кто его ждут, все подписавшиеся на него слушатели, так и не узнают, что оно произошло.

Вызов события происходит тем же способом, что и вызов делегата, вы вызываете его как метод.

```
if (WorkPerformed != null) {
    WorkPerformed(8, WorkType.Work);
}
```

Прежде чем вызвать событие, необходимо посмотреть, есть хоть что-то в списке вызовов у нашего события, иначе получится, что вызывать нам совершенно нечего. Помните, что делегат это ссылка на метод. И запуская на выполнение делегат, мы на самом деле вызываем метод, однако если список вызовов пуст, произойдёт исключение.

Можно использовать более короткую запись:

```
WorkPerformed?.Invoke(8, WorkType.Work);
```

# Вызов событий

Надо понимать, что события - это обёртки над делегатами.

Можно вызвать событие как делегат напрямую:

```
WorkPerformedHandler del = WorkPerformed as WorkPerformedHandler;
if (del != null)
{
    del(8, WorkType.Work);
}
```

Или в более короткой записи:

```
(WorkPerformed as WorkPerformedHandler)?.Invoke(8, WorkType.Work);
```

# Пример определения и вызова события

```
public delegate int WorkPerformedHandler(int hours, WorkType workType);

public class Worker
{
    public event WorkPerformedHandler WorkPerformed; ◄---- Определеие события

    public virtual void DoWork(int hours, WorkType workType)
    {
        OnWorkPerfomed(hours, workType);
    }

    protected virtual void OnWorkPerfomed(int hours, WorkType workType)
    {
        WorkPerformed?.Invoke(hours, workType); ◄---------- Вызов события
    }
}
```

# Самостоятельная работа

Убедиться, что случайные данные плохо упаковываются архиваторами (на примере Zip-архива).

Для этого мы напишем генератор случайных данных, который выдавать запрошенное число произвольных байтов в виде массива.

Затем сохраним эти байты в бинарном виде в файл, заархивируем его и сравним размер архива с размером оригинального файла.

# Самостоятельная работа

Генерация должна происходить в классе RandomDataGenerator в единственном публичном методе

```
public byte[] GetRandomData(int dataSize, int bytesDoneToRaiseEvent)
```

- Первый параметр dataSize - размер массива в байтах
- Второй параметр bytesDoneToRaiseEvent - число байт, после очередной генерации которых надо вызвать событие RandomDataGenerated, связанное с делегатом типа:
  ```
  public delegate void RandomDataGeneratedHandler(int bytesDone, int totalBytes);
  ```
- После завершения генерации необходимо вызвать событие RandomDataGenerationDone, связанное с делегатом типа EventHandler.

В основном потоке программы подписаться на оба события.

- В обработчике события RandomDataGenerated необходимо выводить строку по примеру:
  `Generated XXX from YYY byte(s)...`
- В обработчике события RandomDataGenerationDone:
  `Generation DONE`
- Вывести получившийся массив на экран в виде Base64-строки, воспользовавшись методом Convert.ToBase64String()
- Сохранить получившийся массив в файл **в** бинарном виде.

Средствами ОС создать заархивировать файл и сравнить размер.

# Наследование EventArgs, EventHandler<T>

Стандартным способом передачи параметров в обработчик событий является объект класса EventArgs или его наследного класса.

```
public class WorkPerformedEventArgs: EventArgs
{
    public int Hours { get; set; }
    public WorkType WorkType { get; set; }
}
```

.NET включает в себя обобщённый класс EventHandler<T>, который может использоваться вместо собственного делегата:

```
public delegate void WorkPerformedHandler(object sernder, WorkPerformedEventArgs e);
public event WorkPerformedHandler WorkPerformed;
```

```
public event EventHandler<WorkPerformedEventArgs> WorkPerformed;
```

# Самостоятельная работа

Замените собственный делегат RandomDataGeneratedHandler на встроенный EventArgs<T>.

Внесите необходимые изменения в программу, чтобы она снова компилировалась и работала верно.

# Chatbot: Интерфейсы и классы

Живое обсуждение в классе возможной компоновки задач приложения.

# Домашняя работа: Chatbot

Написать интерфейсы и классы (со взаимосвязями) проекта chatbot, которые мы обсудили во время классной работы.

# Домашняя работа: События (опционально)

Написать класс FileWriterWithProgress, у которого был бы один метод
public void WriteBytes(string fileName, byte[] data, float percentageToFireEvent)

- первый параметр - это имя файла,
- второй параметр - это сам массив для записи
- третий параметр - это процентная величина для вызова периодического события о прогрессе (0 < percentageToFireEvent < 1) - см. пример ниже

Класс должен предоставлять 2 типа события

- WritingPerformed достигнут прогресс записи, кратный кратный параметру percentageToFireEvent
- WritingCompleted достигнут конец записи.

Примеры:

```
var writer = new FileWriterWithProgress();

writer.WriteBytes(data, 0.1);
// будет 11 событий - 10 событий WritingPerformed при достижении 10%, 20%, …, 100% записи
// + 1 событие WritingCompleted при завершении.

writer.WriteBytes(data, 0.15);
// будет 7 событий - 6 событий WritingPerformed при достижении 15%, 30%, …, 90% записи
//+ 1 событие WritingCompleted при завершении.
```

# Спасибо за внимание.