Тема 9. Логирование и обработка ошибок в Windows Forms.

Цель занятия:

Ознакомиться с основами логирования в Windows Forms.

Учебные вопросы:

1. Зачем нужно логирование?
2. Базовая обработка ошибок.
3. Глобальная обработка ошибок.
4. Простое логирование.

1. Зачем нужно логирование?

В процессе разработки программист видит все ошибки в режиме отладки (Debug) прямо в Visual Studio. Однако, когда программа передается пользователю, ситуация меняется.

Логирование — это автоматическая запись истории работы программы (событий, действий пользователя, ошибок) в постоянное хранилище (обычно в текстовый файл).

Ключевые причины использовать логи:

- Разбор полетов (Post-mortem): У пользователя программа может «вылететь» или вести себя странно. Без логов программист услышит только: «Я что-то нажал, и всё исчезло». Лог-файл покажет точную последовательность действий и техническую причину сбоя.
- Ошибки, которые сложно повторить: Некоторые баги проявляются только при специфических условиях (нехватка памяти, отсутствие доступа к интернету, специфическая версия Windows). Лог позволяет увидеть условия, при которых возникла проблема на «чужом» компьютере.
- Аудит действий (Безопасность): В корпоративных системах важно знать, кто и когда внес изменения в данные, удалил запись или пытался войти в систему под чужим паролем.
- Мониторинг в реальном времени: Администратор может следить за файлом лога, чтобы вовремя заметить, что сервер начал работать медленно или база данных стала часто выдавать предупреждения.

Важно: Правильное логирование экономит сотни часов на техническую поддержку приложения.

Проще попросить пользователя: «Пришлите файл log.txt», чем пытаться угадать причину ошибки по телефону.

Ситуация 1 (без логирования):

Пользователь: "Ваша программа сломалась!"

Разработчик: "Что случилось?"

Пользователь: "Не знаю, просто перестала работать"

Разработчик: 😕

Ситуация 2 (с логированием):

Пользователь: "Ваша программа сломалась!"

Разработчик: "Посмотрим логи... Вижу: 'Ошибка при сохранении файла: нет доступа к папке C:\Docs'"

Разработчик: "Проверьте права доступа к папке C:\Docs"

Пользователь: "Проверил, теперь работает!"

Пример лога (журнала):

[2024-01-15 14:30:15] INFO: ===== Программа запущена =====
[2024-01-15 14:30:15] INFO: Пользователь: Иванов
[2024-01-15 14:30:15] INFO: Версия программы: 1.0.2
[2024-01-15 14:30:15] INFO: Система: Windows 10 (64-bit)
[2024-01-15 14:32:10] INFO: Открыт документ: "Отчет за январь.docx"
[2024-01-15 14:35:22] INFO: Пользователь изменил документ
[2024-01-15 14:40:05] INFO: Нажата кнопка "Сохранить"
[2024-01-15 14:40:05] INFO: Попытка сохранить файл: "C:\Docs\Отчет за январь.docx"
[2024-01-15 14:40:05] ERROR: Ошибка при сохранении файла!
[2024-01-15 14:40:05] ERROR: Сообщение системы: "Отказано в доступе"
[2024-01-15 14:40:05] ERROR: Путь: "C:\Docs\ "
…
[2024-01-15 14:45:00] INFO: ===== Программа закрыта =====

2. Базовая обработка ошибок.

Освежим в памяти классический механизм C# и адаптируем его под специфику интерфейса Windows Forms.

Основной инструмент борьбы с ошибками в коде — блок try-catch-finally.

Он позволяет программе не «умирать» при возникновении исключения, а выполнить запасной план.

Пример:

```
try
{
    // 1. Блок «Попытка». Сюда пишем опасный код: расчеты, работа с файлами, парсинг данных.
    int age = int.Parse(textBoxAge.Text);
}
catch (FormatException ex)
{
    // 2. Блок «Перехват». Срабатывает только если в try произошла ошибка указанного типа.
    // ex — объект, содержащий детали ошибки.
    MessageBox.Show("Ошибка: Введите числовое значение!", "Внимание");
}
catch (Exception ex)
{
    // Универсальный перехватчик для всех остальных типов ошибок.
    // Важно: этот блок всегда должен быть последним в списке catch.
    MessageBox.Show("Произошла неизвестная ошибка: " + ex.Message);
}
finally
{
    // 3. Блок «Завершение» (необязательный)
    // Выполняется ВСЕГДА. Обычно здесь закрывают файлы или соединения с БД.
}
```

Специфика для Windows Forms

В визуальных приложениях обработка ошибок имеет свои особенности:

- Защита событий: Каждый обработчик кнопки (button_Click) — это потенциальная точка сбоя. Если не обернуть сложную логику внутри клика в try-catch, одно неверное действие пользователя закроет всё приложение.
- Информативность: В WinForms мы используем MessageBox для обратной связи.

Плохо: catch { } (пустой блок). Пользователь жмет кнопку, ничего не происходит, он в замешательстве.

Хорошо: Вывести сообщение о том, что именно пошло не так, и что пользователю сделать (например, «Проверьте подключение к интернету»).

Иерархия исключений:

Нужно стараться ловить конкретные ошибки (FileNotFoundException, DivideByZeroException), а не использовать общий Exception для всего.

Это помогает точнее реагировать на проблему.

Пример:

```
private void buttonLoad_Click(object sender, EventArgs e)
{
    try
    {
        // Пытаемся прочитать файл
        string content = File.ReadAllText("data.txt");
        textBoxContent.Text = content;
    }
    catch (FileNotFoundException)
    {
        MessageBox.Show("Файл data.txt не найден!\nСоздайте его или выберите другой файл.");
    }
    catch (IOException ex)
    {
        MessageBox.Show($"Ошибка чтения файла:\n{ex.Message}");
    }
    catch (Exception ex) // "На всякий случай"
    {
        MessageBox.Show($"Неизвестная ошибка:\n{ex.Message}");
    }
}
```

Когда обязательно использовать?

- При преобразовании текста из TextBox в числа (int.Parse, double.Parse).
- При работе с внешними файлами (File.Open, File.ReadAllText) и БД.
- При работе с массивами (где возможен выход за границы индекса).

Золотые правила:

- Всегда обрабатывайте ошибки ввода пользователя
- Никогда не оставляйте пустой catch-блок
- Проверяйте данные до использования (лучше предотвратить, чем обрабатывать)
- Информируйте пользователя о проблеме понятным языком

3. Глобальная обработка ошибок

Даже самый опытный программист не может обернуть каждый метод в try-catch.

Глобальная обработка это «последняя линия обороны», позволяет перехватить те ошибки, которые вы случайно пропустили, и не дать программе просто «исчезнуть» с экрана.

В Windows Forms есть специальный механизм, который позволяет поймать любое исключение, возникшее в главном потоке интерфейса.

Настраивается это в файле Program.cs.

Зачем это нужно?

- Предотвращение «тихого» падения: Без этого программа просто закрывается, оставляя пользователя в недоумении.
- Централизованное логирование: Вам не нужно писать код записи в файл в каждом catch по всей программе — достаточно написать его один раз здесь.
- Красивое уведомление: Вы можете показать пользователю вежливое окно с извинениями и инструкциями вместо системной ошибки.

Настройка в Program.cs

Чтобы включить глобальный перехват, нужно подписаться на событие ThreadException перед запуском основной формы.

```
static void Main()
{
    Application.EnableVisualStyles();
    Application.SetCompatibleTextRenderingDefault(false);

    // 1. Подписываемся на глобальную ошибку
    Application.ThreadException += new ThreadExceptionEventHandler(GlobalExceptionHandler);

    // 2. Указываем, что мы хотим именно перехватывать ошибки (CatchException)
    Application.SetUnhandledExceptionMode(UnhandledExceptionMode.CatchException);

    ApplicationConfiguration.Initialize();
    Application.Run(new Form1());
}
```

```
// 3. Метод, который сработает при любой неучтенной ошибке
Ссылок: 1
static void GlobalExceptionHandler(object sender, ThreadExceptionEventArgs e)
{
    // Показываем сообщение пользователю
    MessageBox.Show("Произошла непредвиденная ошибка. Проверьте файл log.txt",
                    "Критический сбой",
                    MessageBoxButtons.OK,
                    MessageBoxIcon.Stop);

    // Здесь обычно вызывают метод записи в лог-файл (об этом в след. пункте)
    File.AppendAllText("log.txt", $"{DateTime.Now}: {e.Exception.Message}\n");
}
```

Два типа глобальных ошибок (для справки):

- Application.ThreadException — ловит ошибки в главном потоке (UI). Это 99% всех ошибок в студенческих работах (клики по кнопкам, расчеты).
- AppDomain.CurrentDomain.UnhandledException — ловит ошибки в фоновых потоках. Если ваше приложение использует многопоточность, стоит подстраховаться и этим событием тоже.

Важный нюанс для отладки:

Когда вы запускаете программу из Visual Studio (нажав F5), она будет по-прежнему «вылетать» на строке с ошибкой.

Это нормально — отладчик имеет приоритет. Глобальный обработчик сработает в «боевых» условиях, когда вы запустите готовый .exe файл из папки проекта.

4. Простое логирование

Логирование — это не просто запись текста. Это создание хронологии событий. Чтобы лог был полезным, он должен отвечать на вопросы: «Когда?», «Что случилось?» и «Где именно?».

Как записывать данные в файл?

В .NET для самой простой записи в файл используется метод File.AppendAllText из пространства имен System.IO. Он удобен тем, что:

- Автоматически открывает файл.
- Добавляет текст в самый конец (не стирая старое).
- Закрывает файл после записи.

```
using System.IO;

// Простейшая запись
File.AppendAllText("log.txt", "Произошла ошибка\n");
```

Формат «правильной» строки лога.

Запись «Произошла ошибка» бесполезна, если в файле накопится 100 таких строк. Хорошая строка лога должна включать:

- Метку времени (DateTime.Now) — дата и точное время до секунды.
- Тип события (INFO, WARNING, ERROR).
- Текст ошибки (ex.Message).

Пример универсального метода:

```
static void WriteToLog(string message, string type = "ERROR")
{
    string logPath = "app_log.txt";
    // Формируем строку: [03.01.2026 12:00:01] [ERROR] Сообщение
    string line = $"[{DateTime.Now}] [{type}] {message}\n";

    File.AppendAllText(logPath, line);
}
```

Где хранить файл лога?

Для учебных работ: Файл создается в той же папке, где лежит .exe (папка bin/Debug).

Важный нюанс: Если программа запущена из папки Program Files, обычная запись в файл может не сработать из-за отсутствия прав администратора.

Но для лабораторных работ это обычно не является проблемой.

Итоги лекции:

- Опасный код всегда оборачиваем в try-catch.
- В catch обязательно добавляем вызов WriteToLog.
- В Program.cs настраиваем «страховочную сетку» (глобальный обработчик).
- Пользователю показываем вежливый MessageBox, а в файл пишем технические подробности.

Материалы лекций:

https://github.com/ShViktor72/Education2025

Контрольные вопросы:

- Что произойдет с приложением Windows Forms, если в коде возникнет исключение (Exception), которое не обернуто в блок try-catch?
- В каких случаях блок finally незаменим при работе с ошибками?
- Почему пустой блок catch { } считается плохой практикой программирования?
- В каком файле проекта и в каком методе настраивается глобальный перехват ошибок?
- Чем информация в лог-файле должна отличаться от информации в MessageBox, который видит пользователь?
- Назовите минимум три обязательных элемента, которые должны присутствовать в каждой строке «хорошего» лога.
- Если программа не может прочитать файл из-за отсутствия прав доступа, какая информация в логе поможет вам быстрее всего найти проблему?