

1 Lecture 4: Files, Collections, and Application Structure

Course: Scientific Visual Programming

Topic: Data Handling and Maintainable WinForms Code

Recommended duration: 90 minutes

1.1 Learning objectives

Students should be able to read and write text files safely, use collections of objects, serialize simple data, separate UI code from business logic, and handle exceptions without hiding useful information.

1.2 1. Separating responsibilities

A form should coordinate user interaction. It should not contain every business rule, file operation, and calculation. A practical beginner architecture separates the application into a presentation layer, a service or business layer, and a data-access layer.

The presentation layer reads control values and displays results. The business layer validates rules and performs calculations. The data-access layer reads and writes files or databases. Separation makes each part easier to test and change.

1.3 2. Collections and domain classes

A domain class represents a meaningful concept. A `Student` class may contain an identifier, name, department, and grade point average. A collection such as `List<Student>` stores multiple objects.

```
public class Student
{
    public int Id { get; set; }
    public string Name { get; set; } = "";
    public string Department { get; set; } = "";
}

private readonly List<Student> students = new();
```

Use a domain class instead of parallel arrays such as `names[]`, `departments[]`, and `ids[]`. Objects keep related values together and reduce synchronization errors.

1.4 3. Reading and writing text files

The `File` and `StreamReader` classes support text-file operations. Always consider the file path, encoding, missing files, and access errors. Use a file dialog when the user selects a file rather than hard-coding a path.

```
File.WriteAllText(path, txtNotes.Text);
string content = File.Exists(path)
    ? File.ReadAllText(path)
    : "";
```

For larger files, use streaming APIs. For repeated structured records, a delimited format or JSON is usually safer than manually concatenating arbitrary text.

1.5 4. JSON serialization

JSON can represent objects and collections in a readable structured format. With `System.Text.Json`, serialization can be concise:

```
string json = JsonSerializer.Serialize(students, new JsonSerializerOptions
{
    WriteIndented = true
});
File.WriteAllText(path, json);

var loaded = JsonSerializer.Deserialize<List<Student>>(
    File.ReadAllText(path)) ?? new List<Student>();
```

The application should handle invalid or outdated files. A file written by an older version may not contain every property expected by the current version.

1.6 5. Exception handling

Exceptions indicate that an operation failed or that a programming assumption was violated. Catch exceptions where the application can respond meaningfully. For example, a file-opening operation can show a message and let the user choose another file.

```
try
{
    txtNotes.Text = File.ReadAllText(path);
}
catch (IOException ex)
{
    MessageBox.Show($"The file could not be opened: {ex.Message}");
}
```

Do not catch every exception and silently continue. Logging or rethrowing may be necessary when the application cannot safely recover.

1.7 6. Binding collections to controls

A `BindingSource` can connect a collection to a `DataGridView`, `ListBox`, or other control. Binding reduces manual synchronization between object data and displayed rows. When the application changes the collection dynamically, use an observable collection or refresh the binding intentionally.

1.8 7. Designing a grade calculator

Create a `Student` class with three assessment marks. A service method calculates the weighted grade. The form collects the marks, creates or updates a student object, and displays the result. The service method should validate that marks fall within the permitted range.

A service method might be designed as:

```

public decimal CalculateFinalMark(decimal coursework, decimal exam)
{
    if (coursework is < 0 or > 100 || exam is < 0 or > 100)
        throw new ArgumentOutOfRangeException();

    return coursework * 0.4m + exam * 0.6m;
}

```

The form can catch a validation exception and show a user-friendly message, while unit tests can verify the service independently.

1.9 8. Common mistakes

Avoid storing application data only in controls. Controls are a presentation mechanism, not a reliable data model. Avoid using relative file paths without understanding the application working directory. Do not save sensitive information as plain text. Do not perform slow file operations repeatedly on the UI thread.

1.10 Practice activity

Build a contact manager that stores contacts in a JSON file. Provide Add, Edit, Delete, Search, Load, and Save commands. Use a Contact class, a collection, a DataGridView, validation, and exception handling. Explain how the user interface would change if the JSON file were later replaced by a database.

1.11 Summary

Maintainable Windows Forms applications use domain classes, collections, structured persistence, and clear separation of responsibilities. File operations require validation and exception handling. The UI should display and collect data, while reusable logic belongs in classes that can be tested independently.

1.12 References