Pre-Join C# Training – Assignments (Weeks 3–5)

Week 3 — C# Fundamentals

Topics: History & .NET overview • Scope & Accessibility • Namespaces & Libraries • Enums • DataTable • Date/String/Math • File I/O

Learning Goals

- Understand .NET ecosystem basics and C# compilation model.
- Apply access modifiers appropriately.
- Organize code with namespaces and leverage BCL classes.
- Use enums to model finite states.
- Manipulate tabular data via `DataTable`.
- Use date/time, string, and math APIs.
- Perform safe file read/write operations.

Assignment W3-A — "Library Check-In Utility"

Scenario: You're building a command-line tool used by a tiny community library to check-in returned books and produce a daily summary.

Requirements

1. **Project:** Console app `LibraryCheckIn`.
2. **Domain modeling:**
3. Create an enum `BookCondition { New, Good, Worn, Damaged }`.
4. Class `Book` (Id, Title, Author, Condition).
5. Use **proper access modifiers** for fields/properties/methods (justify in comments for at least two decisions).
6. **Data capture:**
7. Accept a CSV file input: `returns_YYYYMMDD.csv` with columns: `Id,Title,Author,Condition`.
8. Parse into a **DataTable** and map rows to `Book` objects.
9. **Processing:**
10. Count totals by `BookCondition`.
11. Compute **string** summaries (e.g., `${Title} by ${Author}`) and a **Math**-based small penalty score: `Damaged=+10, Worn=+3, Good=0, New=-1` then **clamp** to `[0, 100]`.
12. Add a `DateTime` stamp for processing time; format as `yyyy-MM-dd HH:mm:ss`.
13. **Output:**
14. Write a daily report file to `./out/daily_summary_YYYYMMDD.txt` containing:
    - Date/time processed
    - Total returns
    - Count by condition
    - Top 5 titles by penalty (desc)

15. **Error handling & I/O:**
16. Validate file existence; show friendly errors.
17. Use `using` statements and exception handling best practices.
18. **Namespaces & Libraries:**
19. Place types under a `LibraryCheckIn.Domain` and `LibraryCheckIn.IO` namespaces.
20. **Unit tests:**
21. Test CSV parsing edge cases (missing columns, invalid enum string).
22. Test penalty calculation and clamping.

Deliverables

- Source code, sample input CSVs, generated report examples, and unit tests.

Stretch Goals

- Accept both CSV and JSON inputs (use overloading or strategy pattern with interfaces).
- Add a simple configuration file for output directory.

Week 4 — C# Advanced I

Topics: Abstract & sealed classes, interfaces • Generics • File system APIs • Serialization (JSON/XML) • Base Class Library tour • Lambdas • Extension Methods

Learning Goals

- Design with abstraction and interfaces; understand sealed usage.
- Build generic utilities and collections.
- Work safely with the file system.
- Serialize/deserialize to JSON and XML.
- Use lambdas for concise behavior and write extension methods.

Assignment W4-A — "Pluggable Import Pipeline"

Scenario: Extend Week 3 to ingest multiple file formats via a **pluggable** pipeline with **generics**.

Requirements

1. **Project:** Class library `Ingestion.Pipeline` + console host `Ingestion.Cli`.
2. **Abstractions:**
3. `abstract class FileImporter<T>` with `IEnumerable<T> Import(string path)`.
4. Implement `CsvBookImporter : FileImporter<Book>` and `JsonBookImporter : FileImporter<Book>`.
5. Create `IReportWriter<T>` and two implementations: `TextReportWriter`, `XmlReportWriter`.
6. Mark classes `sealed` where extension is not intended; justify in XML doc comments.
7. **Generics & Extension Methods:**

8. Generic extension methods on `IEnumerable<Book>`: `TopBy<TValue>(Func<Book,TValue> keySelector, int n)` and `ToConditionCounts()`.
9. **Lambdas & LINQ:**
10. Use LINQ to filter, group, and project collections (e.g., `OrderByDescending(b => b.Penalty)`)
11. **File System:**
12. Scan an `./in` directory recursively and import all supported files.
13. Implement a `--dry-run` option.
14. **Serialization:**
15. Serialize the final summary to **JSON** and **XML** (choose one as primary, ensure parity tests).
16. **Testing:**
17. Unit tests for each importer and writer; test extension methods.
18. Mock file system interactions where possible.

Deliverables

- Source, tests, and sample data trees with nested folders.
- README with architecture diagram (ASCII or Markdown) and reasoning for `abstract` vs `sealed`.

Stretch Goals

- Add a plugin discovery mechanism via reflection (scan for `FileImporter<T>` in loaded assemblies).
- Write a small `BookSet<T>` **generic collection** with validation rules.

Assignment W4-B — "Secure Notes Vault (CLI)"

Scenario: A minimal note keeper with simple crypto to explore security basics.

Requirements

1. **Project:** Console app `SecureNotes`.
2. **Features:**
3. Create/read/update/delete notes stored as JSON in a vault folder.
4. Each note has: Id (GUID), Title, Body, CreatedAt, UpdatedAt.
5. **Security:**
6. Derive a key from a passphrase (PBKDF2) and encrypt/decrypt note bodies with **AES**.
7. Use **secure coding** practices: never log secrets, zero sensitive buffers when possible.
8. **Serialization:**
9. JSON serialization with custom converters for `DateTimeOffset`.
10. **Testing:**
11. Unit tests: encryption round-trip, invalid passphrase behavior, corrupt file handling.

Deliverables

- Code, tests, README with security considerations and threat model (1 page).

Stretch Goals

- Implement an **XML** export/import of note metadata only (no bodies).

Week 5 — C# Advanced II

Topics: LINQ (DataTable, List) • ORM overview • Security (continued) • `dynamic` type • Database CRUD with C#

Learning Goals

- Write expressive LINQ queries over in-memory collections and `DataTable`.
- Understand ORM concepts and trade-offs.
- Apply secure patterns for data access.
- Know when (and when not) to use `dynamic`.
- Build end-to-end CRUD with a DB from C#.

Assignment W5-A — "Reading Room Manager (CRUD + LINQ)"

Scenario: A small CRUD app to manage library reading rooms and reservations.

Requirements

1. **Project:** Web API (ASP.NET Core Minimal API) `ReadingRoom.Api` + test project.
2. **Entities:** `Room (Id, Name, Capacity)`, `Reservation (Id, RoomId, PatronName, Start, End, Status)` where `Status` is an enum.
3. **Database:** Use **SQLite** via EF Core (Code-First). Migrations required.
4. **API Endpoints:**
5. Rooms: `GET /rooms`, `POST /rooms`, `PUT /rooms/{id}`, `DELETE /rooms/{id}`
6. Reservations: `GET /reservations?roomId=&from=&to=`, `POST /reservations`, `PUT /reservations/{id}`, `DELETE /reservations/{id}`
7. **LINQ Queries:**
8. Top N busiest rooms in a date range.
9. Conflicting reservations finder.
10. Utilization % per room.
11. Provide equivalent queries using **DataTable** and using **List** (demonstrate both in a separate `Analytics` console project that references the API's data layer).
12. **Security:**
13. Input validation, parameterized queries (for any raw SQL), and basic rate limiting.
14. Don't expose internal exceptions; consistent problem details responses.
15. `dynamic` ** Type:**
16. Create a small module demonstrating when `dynamic` helps (e.g., JSON shape from external source) vs. a strongly-typed alternative; document pitfalls.
17. **Testing:**

18. Unit tests for services and LINQ logic; integration tests for endpoints using the WebApplicationFactory.

Deliverables

- Source, DB migrations, seed script, Postman/Bruno collection, and tests.

Stretch Goals

- Add optimistic concurrency for reservations.
- Implement simple JWT auth with roles (Admin can delete). Keep it minimal.

Assignment W5-B — "Mini ORM Show & Tell"

Purpose: Demonstrate knowledge of ORMs and alternatives.

Task

- Prepare a 6–8 minute recorded walkthrough (slides or README) that:
- Compares **EF Core** to **Dapper** and raw ADO.NET.
- Shows one query implemented in each approach (same dataset).
- Discusses performance, maintainability, security, and when you'd choose each.

Deliverables

- Slides/README, short demo code, and a link to a screen recording (optional if live).