1. The Generic Swapper (The Basics)

Goal: Create a utility method that can swap the values of two variables of any type.

- **Task:** Write a static class Utils with a generic method Swap<T>.
- **Requirement:** The method should take two parameters by reference (ref) and swap their values.
- **Test Case:** Try swapping two int values, then two string values.

2. The Simple Box (Constraints)

Goal: Build a container that only holds "Reference Types" (classes), not "Value Types" (like int or structs).

- **Task:** Create a class Box<T>.
- **Constraint:** Use the where T : class constraint.
- **Challenge:** Add a method void PrintType() that prints the name of the type T to the console. Why does Box<int> fail to compile?

3. Circular Buffer (Data Structures)

Goal: Implement a fixed-size Circular Buffer (or Ring Buffer) using generics.

- **Task:** Create a class CircularBuffer<T> with a private array of size N.
- **Logic:** * Push(T item): Adds an item. If the buffer is full, it overwrites the oldest item.
    - T Pop(): Returns and removes the oldest item.
- **Constraint:** Ensure the class handles any data type.

4. The Result Wrapper (Real-world API Design)

Goal: Create a standardized way for methods to return either a success value or an error message.

- **Task:** Create a class Result<T>.
- **Properties:**
  - T Data: The actual value returned on success.
  - bool IsSuccess: Whether the operation worked.
  - string ErrorMessage: Detailed info if it failed.
- **Logic:** Add two static methods: Result<T>.Success(T data) and Result<T>.Failure(string message).

5. Generic Repository Pattern (Advanced)

Goal: Simulate a database repository that works with any entity that has an Id.

- **Task:** 1. Define an interface IEntity with a property int Id. 2. Create a class Repository<T> where T must implement IEntity and have a parameterless constructor (new()).
- **Methods:** * void Add(T entity)
    - T GetById(int id) (Use a List<T> internally to store items).
- **Constraint:** where T : class, IEntity, new()

Quick Syntax Refresher

- If you get stuck on the syntax for constraints, remember the pattern:

C#

```csharp
public class MyGenericClass<T> where T : IEnumerable, new()
{
    // T must be a collection AND have a public parameterless constructor
}
```