Exercise 1: The Basics

Goal: Manually perform boxing and unboxing.

- Create an integer variable and assign it the value 101.
- Box this integer into an object variable.
- Unbox the object back into a **new** integer variable.
- Print both the object and the new integer to the console.

Exercise 2: The "Hidden" Boxing Trap

Goal: Identify boxing in common API calls.

- C# string.Format or Console.WriteLine often box value types behind the scenes because they accept object parameters.
- **Task:** Write a line of code using Console.WriteLine that prints an int, a double, and a bool in a single string (e.g., "Values: 10, 5.5, True").
- **Question:** How many boxing operations occurred in that single line of code?

Exercise 3: The Invalid Cast Exception

Goal: Understand the strictness of unboxing.

- Create a double variable with the value 9.99.
- Box it into an object.
- Try to unbox that object directly into an int variable.
- **Task:** Surround this in a try-catch block to handle the specific exception that occurs. (Note: You cannot unbox to a different type even if a conversion exists; you must unbox to the *exact* original type first).

Exercise 4: Collection Performance Challenge

Goal: Contrast ArrayList vs. List<T>.

- Create an ArrayList and a List<int>.
- Write a loop that adds 10,000 integers to the ArrayList.
- Write a second loop that adds 10,000 integers to the List<int>.
- **Task:** Use System.Diagnostics.Stopwatch to measure which one is faster and explain why in a code comment.

Exercise 5: Boxing with Interfaces

Goal: Understand how structs behave when cast to interfaces.

- Define a simple interface IWorkable with a method DoWork().
- Define a struct Worker : IWorkable that implements that method.
- In your Main method, create an instance of the Worker struct.
- Cast that struct to the IWorkable interface.
- **Task:** Research or test whether casting a **struct** to an **interface** causes boxing. (Hint: Look at where the interface reference must live in memory).