Exercise 1: The Metadata Explorer

Goal: Extract a "blueprint" of a class at runtime.

- **Task:** Write a method AnalyzeClass(object obj) that takes any object and prints:
    1. The full Name and Namespace of the class.
    2. A list of all **Public Properties** and their data types.
    3. A list of all **Public Methods** (excluding those inherited from System.Object).
- **Hint:** Use type.GetMethods(BindingFlags.Public | BindingFlags.Instance | BindingFlags.DeclaredOnly) to exclude inherited methods like ToString().

Exercise 2: The Private Detective

Goal: Access data that is encapsulated (hidden).

- **Task:** Create a class BankVault with a private string _pinCode = "1234". In your Main method:
    1. Instantiate BankVault.
    2. Use Reflection to find the private field _pinCode.
    3. Print the value to the console.
    4. Use Reflection to **change** the value to "9999" and verify the change.
- **Hint:** You must use BindingFlags.NonPublic | BindingFlags.Instance to see private fields.

Exercise 3: The Dynamic Factory

Goal: Instantiate objects when the class name is only known as a string.

- **Task:** Imagine a program that supports multiple languages. Create two classes: EnglishGreeter and SpanishGreeter, both having a method SayHello().
    1. Ask the user to type "English" or "Spanish" into the console.
    2. Based on the input, construct the class name string (e.g., "EnglishGreeter").
    3. Use Type.GetType(string) and Activator.CreateInstance() to create the object.
    4. Invoke the SayHello() method dynamically.

Exercise 4: Universal Property Cloner

Goal: Deeply understand how to read and write values dynamically.

- **Task:** Write a generic method CloneProperties<T>(T source, T target).
    1. Loop through every public property of the source object.
    2. Read the value from source.
    3. Write that same value into the corresponding property of target.
- **Constraint:** Do not hardcode any property names. It must work for any class passed to it.

Exercise 5: The Method Runner (With Parameters)

Goal: Handle method overloads and parameter passing via Reflection.

- **Task:** Create a class Calculator with a method Add(int a, int b).
    1. Get the MethodInfo for Add.
    2. Create an object[] array containing the numbers 10 and 20.
    3. Invoke the method using the array as the arguments.
    4. Capture the return value of the Invoke call and print it.
- **Challenge:** What happens if you pass a string instead of an int in your object array? Try it and handle the exception.