Exercise 1: The Branch Name

Scenario: You have a Store object that has a Location property. The Location contains a BranchName (string). **Task:** Access the BranchName safely. If Store or Location is null, the result should be null.

C#

```csharp
public class Store { public Address Location { get; set; } }
public class Address { public string BranchName { get; set; } }

Store myStore = null;
// Your code here:
string name =
```

Exercise 2: The Data Formatter

Scenario: You have a Formatter class with a method Format(string input) that returns a string. **Task:** Use the null-conditional operator to call Format("hello") on a Formatter instance that might be null.

C#

```csharp
public class Formatter { public string Format(string s) => s.ToUpper(); }

Formatter processor = null;
// Your code here:
string result =
```

Exercise 3: Safe Collection Item Access

Scenario: You have an array of Customer objects. The array itself might not be initialized. **Task:** Use the null-conditional operator to access the **first element** (index 0) of the customers array.

C#

```csharp
public class Customer { public string Name { get; set; } }

Customer[] customers = null;
// Your code here:
Customer first =
```

Exercise 4: Deep String Manipulation

Scenario: You are accessing a User object's Email. You want to call the .Trim() method on that email string to remove whitespace. **Task:** Chain the operators so that you safely access the Email and then safely call .Trim().

C#

```csharp
public class User { public string Email { get; set; } }

User currentUser = GetUser(); // Might return null
// Your code here:
string cleanEmail =
```

Exercise 5: The "ToString" Safety Net

Scenario: Every object in C# has a .ToString() method. You have an object variable called data that could be null. **Task:** Get the string representation of data using the null-conditional operator so it doesn't crash if data is missing.

```csharp
object data = null;
// Your code here:
string display =
```