1. Inheritance: The "Is-A" Relationship

The goal here is to establish a hierarchy where a specialized class gains the properties and methods of a general one.

- **The Smart Home Hierarchy:** Create a base class SmartDevice with properties for DeviceName and IsOn. Then, create a derived class SmartLight that adds a Brightness property. Ensure the SmartLight constructor properly initializes the base class properties.
- **The Employee Payroll:** Create a base class Employee with Name and ID. Create a derived class Manager that includes a TeamSize property. In your Main method, instantiate a Manager and access the Name property inherited from Employee.
- **The Vehicle Fleet:** Build a Vehicle class with a StartEngine() method. Create a Truck class that inherits from Vehicle and adds a CargoCapacity property. Show how a Truck object can call StartEngine() without you rewriting the code in the Truck class.

---

2. Method Overrides: Custom Behavior

Inheritance is great, but sometimes the "default" behavior isn't quite right for the child class. We use virtual and override to fix that.

- **The Sound of Music:** Create a base class Instrument with a virtual method Play(). Create two derived classes, Guitar and Drum. Override Play() in both to print "Strumming strings..." and "Beating drums..." respectively.
- **The Shape Area:** Define a base class Shape with a virtual method CalculateArea(). Create a Square class (with a Side property) and override the method to return Side^2. Use the override keyword specifically.
- **The Messaging App:** Create a base class Notification with a method Send(). Override it in a SMSNotification class to include a character limit check, and in an EmailNotification class to include a "Subject Line" in the output.

---

3. Polymorphism: One Interface, Many Forms

This is where the magic happens—treating a collection of different objects as if they were all the same base type.

- **The Animal Shelter:** Create an array of type Animal[]. Fill it with instances of Dog, Cat, and Bird (all of which inherit from Animal and override a MakeSound() method). Loop through the array and call MakeSound() on each to see polymorphism in action.
- **The Bank Account Processor:** Create a base class Account with a virtual method ApplyInterest(). Create SavingsAccount and CheckingAccount with different interest logic. Write a method ProcessAccounts(List<Account> accounts) that iterates through the list and applies interest to all, regardless of their specific type.
- **The Drawing Canvas:** Create an interface or base class IDrawable with a Draw() method. Create a List<IDrawable> containing Circle, Rectangle, and Triangle objects. Use a foreach loop to "Draw" the entire list, demonstrating that the caller doesn't need to know the specific shape to render it.