Handout

Certainly! Let's break down the provided C# program step by step, highlighting how it demonstrates abstraction through the use of an abstract class `Pet`, concrete classes `Dog` and `Cat`, and encapsulation via `get` and `set` accessors.

**Abstraction with Abstract Class `Pet`:**

- Abstract Class: The `Pet` class is declared as an abstract class, meaning it cannot be instantiated directly. It serves as a blueprint for other classes to inherit from.

- Common Properties: The `Pet` class contains common properties that all pets might have, such as `Name`, `Age`, and `Breed`.

- Abstract Method: The `MakeSound()` method is declared as abstract within the `Pet` class. This method represents a behavior that all pets should have, but the specific implementation may vary depending on the type of pet.

**Concrete Classes `Dog` and `Cat`:**

- Inheritance: Both `Dog` and `Cat` classes inherit from the abstract class `Pet`, thereby inheriting its properties and methods.

- Specific Implementations: Each concrete class provides specific implementations for the abstract method `MakeSound()`. For example, the `Dog` class makes a "Woof!" sound, while the `Cat` class makes a "Meow!" sound.

**Encapsulation with `get` and `set` Accessors:**

- Encapsulation: Encapsulation is achieved through the use of `get` and `set` accessors for the properties `Name`, `Age`, and `Breed` within the `Pet` class. These accessors control the access to the class's fields, ensuring that they are accessed and modified in a controlled manner.

**Main Class:**

- Instantiation: Within the `Main` method, instances of both `Dog` and `Cat` classes are created, providing specific values for their properties.

- Usage: The properties of each pet are accessed using the `get` accessor to retrieve their values and displayed to the console. Additionally, the `MakeSound()` method of each pet is invoked, demonstrating polymorphism as each pet produces its specific sound.

**Summary:**

This program demonstrates abstraction by defining a common interface (`Pet` class) for various types of pets, allowing for code reusability and flexibility in handling different pet types. Concrete classes (`Dog` and `Cat`) provide specific implementations while inheriting common properties and behaviors from the abstract class. Encapsulation ensures data integrity and controlled access to class properties, enhancing the program's maintainability and readability. Overall, the program showcases key principles of object-oriented programming, including abstraction, inheritance, encapsulation, and polymorphism.