Handout

Absolutely! Let's delve deeper into the abstraction concept demonstrated in this example.

Abstraction is a fundamental principle in object-oriented programming that allows you to model real-world entities as simplified representations in your code. It involves hiding the complex implementation details and exposing only the essential features of an object. In the provided C# example:

**1. Interface `IPet`:**

- The `IPet` interface serves as a blueprint for all types of pets in the application.

- It defines properties that are common to all pets: Name, Age, Breed, and Color.

- By using an interface, we establish a contract that any class implementing `IPet` must provide implementations for these properties.

- This allows us to treat all pets uniformly regardless of their specific type.

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

- Both `Dog` and `Cat` classes implement the `IPet` interface, which means they must provide implementations for all properties defined in `IPet`.

- Each class provides its own implementation details for these properties.

- Despite having different implementations, both `Dog` and `Cat` can be treated as `IPet` objects, enabling polymorphic behavior.

**3. Main Method:**

- In the `Main` method, we create instances of `Dog` and `Cat`.

- We interact with these objects using the properties defined in the `IPet` interface (`Name`, `Age`, `Breed`, `Color`).

- This demonstrates how we can work with different types of pets using a common interface without needing to know the specific details of each pet type.

Overall, abstraction allows us to manage complexity by focusing on essential properties and behaviors while hiding unnecessary details. It promotes code reusability, maintainability, and flexibility by providing a clear separation between interface and implementation. In this example, abstraction enables us to build a pet selling and buying app that can handle various types of pets uniformly, facilitating easier maintenance and future extensions.