Pet Store Application – OOD Assignment Report

1 Overview of the System

The Pet Store application has been developed to manage a small pet retail business. It provides functionality for maintaining detailed records of pets, pet supplies, employees, and customers. The system supports:

- Maintaining a register of pets and their prices.
- Maintaining a register of pet supplies (food, toys, etc.) and their prices.
- Registering employees and enabling login/logout functionality.
- Registering returning customers and tracking loyalty points.
- Allowing customers to make purchases.
- Feeding pets via a dedicated interface to demonstrate separation of responsibilities.
- Tracking total revenue and providing inventory reporting.

The system is built using object-oriented programming principles to promote code reuse, maintainability, and clarity. Through the structured use of classes, interfaces, inheritance, and polymorphism, common functionalities are efficiently shared across different entities while allowing for specific behaviors where necessary.

2 Project Structure & Responsibilities

2.1 Program.cs

- Entry point of the application.

- Creates a `PetStore` instance and adds pets, supplies, employees, and customers.
- Demonstrates feeding pets and processing purchases.
- Displays inventory and total revenue.

2.2 PetStore.cs

- Main store class containing lists of `IPurchasable` items, employees, and customers.
- Methods:
  - `AddItem / RegisterEmployee / RegisterCustomer`
  - `ProcessPurchase` – updates revenue and loyalty points
  - `FeedAllPets` – feeds all `IFeedable` items
  - `ShowInventory` – prints items, employees, and customers

2.3 Pets.cs

- Abstract `Pet` class implements `IFeedable` and inherits from `StoreItem`.
- `Dog` and `Cat` inherit from `Pet` and override `Feed()`.
- Demonstrates inheritance, polymorphism, and interfaces.

2.4 Supply.cs

- Inherits from `StoreItem`.
- Can `Restock` and `Purchase`.
- Represents toys, food, or other supplies.

2.5 Employee.cs

- Implements `ILoginable`.
- Tracks login status and role.

2.6 Customer.cs

- Implements `IPurchasable` (price always 0 for customers).
- Tracks loyalty points and can make purchases.

2.7 StoreItem.cs

- Base class for all store items (`Pet` and `Supply`).
- Implements `IPurchasable` and provides a default `Purchase()` method.

2.8 Interfaces.cs

- Defines three interfaces: `IPurchasable`, `ILoginable`, `IFeedable`.
- Ensures separation of responsibilities and promotes polymorphism.

3 OOP Principles Demonstrated

3.1 Classes

Classes are blueprints for creating objects. They define the structure (attributes) and behavior (methods) that objects will have.

Role: Classes such as `PetStore`, `StoreItem`, `Pet`, `Supply`, `Employee`, and `Customer` define the core entities of the system and their capabilities.

3.2 Objects

Objects are instances of classes, representing concrete entities with their own state.

Role: Objects like:

```
Dog max = new Dog("Max", 250);
Customer john = new Customer("John␣Smith", "john@mail.com");
```

are the actual pets, customers, and employees that the store manages.

3.3 Encapsulation

Protects internal data by controlling access via private fields and public properties.

Role: Private fields like `name`, `price`, and `loyaltyPoints` safeguard internal state. Public properties provide controlled access, and read-only fields like `LoggedIn` in `Employee` ensure secure state management.

3.4 Constructors

Initializes objects with valid data and sets default states.

Role: Ensures each item, pet, employee, or customer has correct initial attributes such as Name, Price, Role, or LoyaltyPoints.

```
public Dog(string name, double price) :
base(name, "Dog", price) { }
```

3.5 Inheritance

Allows code reuse and hierarchical relationships between classes.

Role: `Dog` and `Cat` inherit from `Pet`, sharing properties and methods. `Pet` and `Supply` inherit from `StoreItem`, reducing code duplication and allowing specialized behavior.

3.6 Polymorphism

Enables different objects to be treated uniformly through base classes or interfaces.

Role: Overridden methods like `Purchase()` and `Feed()` provide type-specific behaviors. Methods like `ProcessPurchase` and `FeedAllPets` operate on interfaces (`IPurchasable`, `IFeedable`) without needing to know exact object types.

4 Summary

The Pet Store application demonstrates core object-oriented principles:

- **Classes:** Provide reusable blueprints for entities.
- **Objects:** Represent concrete instances of these entities.
- **Encapsulation:** Protects internal data and ensures controlled access.
- **Constructors:** Initialize objects with valid states.
- **Inheritance:** Reduces duplication and allows specialized behavior.
- **Polymorphism:** Enables uniform handling of different object types.

Overall, the system maximizes code reuse, enforces separation of concerns, and supports scalable, maintainable management of all pet store operations.