Pet Store Application – OOP Assessment 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 Objects and Classes

Classes act as blueprints for creating objects, encapsulating both data and behaviors. Key classes in this system include:

- `PetStore` – Represents the store and manages items, employees, and customers.
- `StoreItem` – Abstract base class for all store items, including pets and supplies.
- `Pet`, `Dog`, `Cat` – Represent pets, implementing `IFeedable` for feeding actions.
- `Supply` – Represents consumable or accessory items.
- `Employee` – Models store staff with login/logout functionality.
- `Customer` – Represents store clients, tracking loyalty points.

Interfaces:

- `IPurchasable` – Defines common purchasing behavior.
- `ILoginable` – Defines login/logout behavior.
- `IFeedable` – Defines feeding behavior for pets.

Objects are instantiated from these classes, for example:

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

3 Encapsulation

Encapsulation protects internal data by controlling access through properties or methods:

- Private fields like `_name`, `_price`, and `_loyaltyPoints` safeguard internal state.

- Public properties (`Name`, `Price`, `LoyaltyPoints`) provide controlled access.
- Read-only access like `LoggedIn` in `Employee` ensures secure state management.

Encapsulation ensures that object data cannot be modified directly from outside, improving system reliability and maintainability.

4 Constructors

Constructors initialize objects with valid data and set default states:

- `StoreItem` and its derived classes ensure every object has proper `Name` and `Price`.
- `Employee` and `Customer` constructors initialize attributes such as `Name`, `Role`, `Id`, `Email`, and `LoyaltyPoints`.

Example:

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

5 Inheritance

Inheritance allows code reuse and the creation of hierarchical relationships:

- `Dog` and `Cat` inherit from `Pet`, sharing properties (`Name`, `Species`, `Price`) and methods (`Feed()`).
- `Pet` and `Supply` inherit from `StoreItem`, reusing `Name` and `Price` properties and the `Purchase()` method.

This reduces code duplication and allows derived classes to extend or override base functionality for specific behaviors.

6 Polymorphism

Polymorphism allows different objects to be treated uniformly via base classes or interfaces:

- `Purchase()` defined in `IPurchasable` is overridden in `Pet`, `Supply`, and `Customer`.
- `Feed()` defined in `IFeedable` is overridden in `Dog` and `Cat` to provide species-specific behavior.
- Methods like `ProcessPurchase` and `FeedAllPets` in `PetStore` operate on `IPurchasable` and `IFeedable` objects without needing to know their specific type.

Polymorphism simplifies code, improves flexibility, and allows new types to be integrated easily.

7 Summary

The Pet Store application demonstrates the core principles of object-oriented programming:

- **Objects:** Represent pets, supplies, employees, and customers.
- **Classes:** Provide reusable blueprints for entities.
- **Encapsulation:** Protects internal data and provides controlled access.
- **Constructors:** Ensure objects are initialized correctly.
- **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.