Sample Implementation for Design Patterns

The sample design patterns are implemented in Java to incorporates the Strategy, Observer, and Factory Method design patterns in the context of an online shopping system.

Strategy Pattern We will start by implementing the Strategy pattern for managing discount strategies:

```java
// Strategy interface
public interface DiscountStrategy {
    double applyDiscount(double amount);
}

// Concrete discount strategies
class FlatDiscountStrategy implements DiscountStrategy {
    private double discountPercentage;

    public FlatDiscountStrategy(double discountPercentage) {
        this.discountPercentage = discountPercentage;
    }

    public double applyDiscount(double amount) {
        return amount - (amount * discountPercentage / 100);
    }
}

class TieredDiscountStrategy implements DiscountStrategy {
    private double threshold;
    private double discountPercentage;

    public TieredDiscountStrategy(double threshold, double discountPercentage) {
        this.threshold = threshold;
        this.discountPercentage = discountPercentage;
    }

    public double applyDiscount(double amount) {
        return amount >= threshold ? amount - (amount * discountPercentage / 100) : amount;
    }
}
```

Observer Pattern Next, we will implement the Observer pattern for real-time product availability monitoring

```java
import java.util.ArrayList;
import java.util.List;

// Subject interface
interface ProductSubject {
    void attach(ProductObserver observer);
    void detach(ProductObserver observer);
    void notifyObservers(String message);
}

// Observer interface
interface ProductObserver {
    void update(String message);
}

// Concrete product subject
class ProductInventory implements ProductSubject {
    private List<ProductObserver> observers = new ArrayList<>();

    public void attach(ProductObserver observer) {
        observers.add(observer);
    }

    public void detach(ProductObserver observer) {
        observers.remove(observer);
    }

    public void notifyObservers(String message) {
        for (ProductObserver observer : observers) {
            observer.update(message);
        }
    }

    // Method to update product availability
    public void updateAvailability(String message) {
        notifyObservers(message);
    }
}

// Concrete product observer
class CustomerNotification implements ProductObserver {
    private String customerName;

    public CustomerNotification(String customerName) {
        this.customerName = customerName;
    }

    public void update(String message) {
        System.out.println("Dear " + customerName + ", " + message);
    }
}
```

Factory Method Pattern Now, let's implement the Factory Method pattern for creating instances of different product categories:

```
abstract class Product {
    private String name;
    private double price;
    private DiscountStrategy discountStrategy;

    abstract void displayInfo();

    public Product(String name, double price, DiscountStrategy discountStrategy){
        this.name = name;
        this.price = price;
        this.discountStrategy = discountStrategy;
    }

    public double getPriceAfterDiscount() {
        return discountStrategy.applyDiscount(price);
    }

    public String getName() {
        return name;
    }
}

// Concrete product types
class ApparelProduct extends Product {
    public ApparelProduct(String name, double price, DiscountStrategy discountStrategy) {
        super(name, price, discountStrategy);
        //TODO Auto-generated constructor stub
    }

    public void displayInfo() {
        System.out.println(x:"This is an apparel product.");
    }
}
```

```java

// Concrete product types
class ElectronicsProduct extends Product {
    public ElectronicsProduct(String name, double price, DiscountStrategy discountStrategy) {
        super(name, price, discountStrategy);
        //TODO Auto-generated constructor stub
    }

    public void displayInfo() {
        System.out.println(x:"This is an electronics product.");
    }
}

// Product factory
class ProductFactory {
    public Product createProduct(String type) {
        if (type.equalsIgnoreCase(anotherString:"electronics")) {
            return new ElectronicsProduct(type, price:0, discountStrategy:null);
        } else if (type.equalsIgnoreCase(anotherString:"apparel")) {
            return new ApparelProduct(type, price:0, discountStrategy:null);
        } else {
            return null;
        }
    }
}
```