CR1 – Handling Periods and Period-Dependent Promotions

As expected, here is CR1.
Your task is to modify the previous solution according to CR1. The GoodPrice store management has introduced the concept of **periods**, in which different base prices and quantity discounts apply.

Method Modification

The `getCartPrice` method signature now takes two parameters:

```
double getCartPrice(Cart cart, Period period)
```

The new parameter, the period, determines the prices and promotions valid at the time of purchase.

Periods

The GoodPrice store can handle multiple periods. The system must store and configure periods as well as their associated promotions.

In the current example, there are three periods:

1. **Normal period** (default, identical to the base task CR0)

Base prices:

- Apple: 500 HUF/kg
- Banana: 450 HUF/kg

Quantity discounts:

- Apple:
  - At least 5 kg → 10% discount
  - At least 20 kg → 15% discount
- Banana:
  - At least 2 kg → 10% discount

👉 The base prices and quantity discounts of the Normal period are exactly the same as those defined in the base task (CR0). This is intentional!

2. **Spring promotion period**

Base prices:

- Apple: 600 HUF/kg
- Banana: 550 HUF/kg

Quantity discounts:

- Apple:
  - At least 2 kg → 15% discount
  - At least 5 kg → 20% discount
- Banana:
  - At least 4 kg → 20% discount
  - At least 7.5 kg → 25% discount

3. **Winter promotion period**

Base prices:

- Apple: 400 HUF/kg
- Banana: 400 HUF/kg

Quantity discounts:

- Apple:
  - At least 10 kg → 5% discount
  - At least 20 kg → 10% discount
- Banana:
  - At least 5 kg → 10% discount

Development Requirements

- Periods, their prices, and their promotional rules must be configurable, so that the GoodPrice store can later expand or modify the list of periods and their contents.
- The operation of the `getCartPrice` method must always be based on the data of the given period.
- Promotions are bound to quantity thresholds and are product-specific (separate for apple and banana).

Note

The current three periods (normal, spring, winter) are only examples. In the future, the store may introduce other periods as well (e.g., autumn sale, summer clearance, etc.). The system must be prepared to handle these.

It is possible that the store management will request additional CRs in the future!

To make the task clearer, we provide the corresponding unit tests for CR1 using JUnit 5 syntax.

Java Unit Tests

```
import static org.junit.jupiter.api.Assertions.*;

import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.Test;
import java.util.List;
import org.store.*;

class StoreCR1Tests {
    static Store target;
    static Period normal, spring, winter;
    static double appleUnitPriceNormal = 500.0;
    static double bananaUnitPriceNormal = 450.0;
    static double appleThreshold1Normal = 5.0;
    static double appleDiscount1Normal = 0.1;
    static double appleThreshold2Normal = 20.0;
    static double appleDiscount2Normal = 0.15;
    static double bananaThreshold1Normal = 2.0;
    static double bananaDiscount1Normal = 0.1;

    static double appleUnitPriceSpring = 600.0;
    static double bananaUnitPriceSpring = 550.0;
    static double appleThreshold1Spring = 2.0;
    static double appleDiscount1Spring = 0.15;
    static double appleThreshold2Spring = 5.0;
    static double appleDiscount2Spring = 0.20;
    static double bananaThreshold1Spring = 4.0;
    static double bananaDiscount1Spring = 0.20;
    static double bananaThreshold2Spring = 7.5;
    static double bananaDiscount2Spring = 0.25;

    static double appleUnitPriceWinter = 400.0;
    static double bananaUnitPriceWinter = 400.0;
    static double appleThreshold1Winter = 10.0;
    static double appleDiscount1Winter = 0.05;
    static double appleThreshold2Winter = 20.0;
    static double appleDiscount2Winter = 0.10;
    static double bananaThreshold1Winter = 5.0;
    static double bananaDiscount1Winter = 0.10;

    @BeforeAll
    public static void initStore() {
        target = new Store();

        normal = new Period("Normal");
        normal.setUnitPrice(Product.APPLE, appleUnitPriceNormal);
        normal.setUnitPrice(Product.BANANA, bananaUnitPriceNormal);
        normal.setDiscount(Product.APPLE, appleThreshold1Normal,
appleDiscount1Normal);
        normal.setDiscount(Product.APPLE, appleThreshold2Normal,
appleDiscount2Normal);
        normal.setDiscount(Product.BANANA, bananaThreshold1Normal,
bananaDiscount1Normal);
```

```
        target.addPeriod(normal);

        spring = new Period("Spring");
        spring.setUnitPrice(Product.APPLE, appleUnitPriceSpring);
        spring.setUnitPrice(Product.BANANA, bananaUnitPriceSpring);
        spring.setDiscount(Product.APPLE, appleThreshold1Spring,
appleDiscount1Spring);
        spring.setDiscount(Product.APPLE, appleThreshold2Spring,
appleDiscount2Spring);
        spring.setDiscount(Product.BANANA, bananaThreshold1Spring,
bananaDiscount1Spring);
        spring.setDiscount(Product.BANANA, bananaThreshold2Spring,
bananaDiscount2Spring);
        target.addPeriod(spring);

        winter = new Period("Winter");
        winter.setUnitPrice(Product.APPLE, appleUnitPriceWinter);
        winter.setUnitPrice(Product.BANANA, bananaUnitPriceWinter);
        winter.setDiscount(Product.APPLE, appleThreshold1Winter,
appleDiscount1Winter);
        winter.setDiscount(Product.APPLE, appleThreshold2Winter,
appleDiscount2Winter);
        winter.setDiscount(Product.BANANA, bananaThreshold1Winter,
bananaDiscount1Winter);
        target.addPeriod(winter);
    }

    @Test
    void test_cr1_normal_apple_threshold1() {
        Period period = normal;
        Product product = Product.APPLE;
        double unitPrice = appleUnitPriceNormal;
        double quantity = appleThreshold1Normal;
        double discount = appleDiscount1Normal;
        double expected = roundTo5(unitPrice * quantity * (1 - discount));
        Cart cart = new Cart(List.of(new Item(product, quantity)));
        double actual = target.getCartPrice(cart, period);
        assertEquals(expected, actual, 0.001);
    }

    @Test
    void test_cr1_normal_apple_threshold2() {
        Period period = normal;
        Product product = Product.APPLE;
        double unitPrice = appleUnitPriceNormal;
        double quantity = appleThreshold2Normal;
        double discount = appleDiscount2Normal;
        double expected = roundTo5(unitPrice * quantity * (1 - discount));
        Cart cart = new Cart(List.of(new Item(product, quantity)));
        double actual = target.getCartPrice(cart, period);
        assertEquals(expected, actual, 0.001);
    }

    @Test
    void test_cr1_normal_banana_threshold1() {
        Period period = normal;
        Product product = Product.BANANA;
        double unitPrice = bananaUnitPriceNormal;
        double quantity = bananaThreshold1Normal;
        double discount = bananaDiscount1Normal;
        double expected = roundTo5(unitPrice * quantity * (1 - discount));
```

```
        Cart cart = new Cart(List.of(new Item(product, quantity)));
        double actual = target.getCartPrice(cart, period);
        assertEquals(expected, actual, 0.001);
    }

    @Test
    void test_cr1_spring_apple_threshold1() {
        Period period = spring;
        Product product = Product.APPLE;
        double unitPrice = appleUnitPriceSpring;
        double quantity = appleThreshold1Spring;
        double discount = appleDiscount1Spring;
        double expected = roundTo5(unitPrice * quantity * (1 - discount));
        Cart cart = new Cart(List.of(new Item(product, quantity)));
        double actual = target.getCartPrice(cart, period);
        assertEquals(expected, actual, 0.001);
    }

    @Test
    void test_cr1_spring_apple_threshold2() {
        Period period = spring;
        Product product = Product.APPLE;
        double unitPrice = appleUnitPriceSpring;
        double quantity = appleThreshold2Spring;
        double discount = appleDiscount2Spring;
        double expected = roundTo5(unitPrice * quantity * (1 - discount));
        Cart cart = new Cart(List.of(new Item(product, quantity)));
        double actual = target.getCartPrice(cart, period);
        assertEquals(expected, actual, 0.001);
    }

    @Test
    void test_cr1_spring_banana_threshold1() {
        Period period = spring;
        Product product = Product.BANANA;
        double unitPrice = bananaUnitPriceSpring;
        double quantity = bananaThreshold1Spring;
        double discount = bananaDiscount1Spring;
        double expected = roundTo5(unitPrice * quantity * (1 - discount));
        Cart cart = new Cart(List.of(new Item(product, quantity)));
        double actual = target.getCartPrice(cart, period);
        assertEquals(expected, actual, 0.001);
    }

    @Test
    void test_cr1_spring_banana_threshold2() {
        Period period = spring;
        Product product = Product.BANANA;
        double unitPrice = bananaUnitPriceSpring;
        double quantity = bananaThreshold2Spring;
        double discount = bananaDiscount2Spring;
        double expected = roundTo5(unitPrice * quantity * (1 - discount));
        Cart cart = new Cart(List.of(new Item(product, quantity)));
        double actual = target.getCartPrice(cart, period);
        assertEquals(expected, actual, 0.001);
    }

    @Test
    void test_cr1_winter_apple_threshold1() {
        Period period = winter;
        Product product = Product.APPLE;
```

```
        double unitPrice = appleUnitPriceWinter;
        double quantity = appleThreshold1Winter;
        double discount = appleDiscount1Winter;
        double expected = roundTo5(unitPrice * quantity * (1 - discount));
        Cart cart = new Cart(List.of(new Item(product, quantity)));
        double actual = target.getCartPrice(cart, period);
        assertEquals(expected, actual, 0.001);
    }

    @Test
    void test_cr1_winter_apple_threshold2() {
        Period period = winter;
        Product product = Product.APPLE;
        double unitPrice = appleUnitPriceWinter;
        double quantity = appleThreshold2Winter;
        double discount = appleDiscount2Winter;
        double expected = roundTo5(unitPrice * quantity * (1 - discount));
        Cart cart = new Cart(List.of(new Item(product, quantity)));
        double actual = target.getCartPrice(cart, period);
        assertEquals(expected, actual, 0.001);
    }

    @Test
    void test_cr1_winter_banana_threshold1() {
        Period period = winter;
        Product product = Product.BANANA;
        double unitPrice = bananaUnitPriceWinter;
        double quantity = bananaThreshold1Winter;
        double discount = bananaDiscount1Winter;
        double expected = roundTo5(unitPrice * quantity * (1 - discount));
        Cart cart = new Cart(List.of(new Item(product, quantity)));
        double actual = target.getCartPrice(cart, period);
        assertEquals(expected, actual, 0.001);
    }

    private double roundTo5(double amount) {
        double remainder = amount % 10.0;
        if (remainder < 2.5) {
            return amount - remainder;
        } else if (remainder < 5.0) {
            return amount - remainder + 5.0;
        } else if (remainder < 7.5) {
            return amount - remainder + 5.0;
        } else {
            return amount - remainder + 10.0;
        }
    }
}
```