CR2 – Coupon Handling

The GoodPrice store has partnered with the GoodCoupon company to expand its marketing strategy with coupons. The goal is for customers to take advantage of unique discounts, thereby increasing brand loyalty and sales. The system must now also be capable of processing various types of coupons.

Method Modification

The `getCartPrice` method signature now accepts three parameters:

```
PriceInfo getCartPrice(Cart cart, Period period, List<String> coupons)
```

The return type is no longer `double`, but a new type, **PriceInfo**, which contains:

- the amount payable (floating point number, in HUF),
- the list of unused coupons.

Coupon Types

In the first stage, the system must be able to handle the following basic coupons:

Product discount coupons:

- **A5**: 5% discount on the total price of apples. Not combinable with any other `A*` coupon.
- **A10**: 10% discount on the total price of apples. Not combinable with any other `A*` coupon.
- **B5**: 5% discount on the total price of bananas. Not combinable with any other `B*` coupon.
- **B10**: 10% discount on the total price of bananas. Not combinable with any other `B*` coupon.

Free quantity coupons:

- **A-FREE1**: Up to 1 kg of apples is free (even if less than 1 kg is purchased, in which case the apple quantity is reduced to 0). Not combinable with other `A*` coupons, but does not exclude quantity discounts or coupons for other products.
- **B-FREE1**: Up to 1 kg of bananas is free (even if less than 1 kg is purchased, in which case the banana quantity is reduced to 0). Not combinable with other `B*` coupons, but does not exclude quantity discounts or coupons for other products.

These coupons can only be used once. They cannot be combined with other coupons for the same product, but can be combined with period-based discounts. The system must process coupons in the order they are provided.

Rules of Coupon Usage

- The system processes coupons in the given order.
- If a coupon cannot be applied (e.g., it would not provide a better deal than an existing discount, such as a quantity discount, or another non-combinable coupon has already been applied), it will not be used.
- Unused coupons must be returned to the customer (these are included in the corresponding list of the `PriceInfo` type).
- For non-combinable coupons, only the first applicable coupon should be considered for a given product; they can still be combined with coupons for other products.
- If the valid quantity discount provides the same or greater benefit than the coupon under consideration, then the coupon must be returned.
- Free coupons (`FREE`) do not exclude other discounts: although they are non-combinable, this applies only to other coupons, not to period-based promotions.
- Free coupons reduce the payable quantity of the product; if the reduced quantity is still sufficient for a quantity discount, then the discount still applies.

Examples

1. **Simple coupon use**
   Cart: 2 kg banana
   Period: Normal
   Coupons: B10
   Result: 810 HUF, Unused: [B10]
2. **Valid coupon**
   Cart: 1 kg apple
   Period: Normal
   Coupons: A10
   Result: 450 HUF, Unused: []
3. **Coupon invalid due to order**
   Cart: 1 kg apple
   Period: Normal
   Coupons: A5, A10
   Result: 475 HUF, Unused: [A10]
4. **Better order, better result**
   Cart: 1 kg apple
   Period: Normal
   Coupons: A10, A5
   Result: 450 HUF, Unused: [A5]
5. **Free product coupon**
   Cart: 0.5 kg apple

Period: Normal
Coupons: A-FREE1
Result: 0 HUF, Unused: []

6. **Free coupon combined with quantity discount**
   Cart: 3 kg banana
   Period: Normal
   Coupons: B-FREE1
   Result: 810 HUF, Unused: []
7. **Coupons for different products – both valid**
   Cart: 1 kg apple, 1 kg banana
   Period: Normal
   Coupons: A5, B5
   Result: 905 HUF, Unused: []
8. **Free coupon excluded by earlier percentage coupon**
   Cart: 1 kg apple
   Period: Normal
   Coupons: A10, A-FREE1
   Result: 450 HUF, Unused: [A-FREE1]
9. **Free coupon first, percentage ignored**
   Cart: 1 kg apple
   Period: Normal
   Coupons: A-FREE1, A10
   Result: 0 HUF, Unused: [A10]
10. **Free coupon reduces quantity, then percentage not allowed**
    Cart: 1.5 kg banana
    Period: Normal
    Coupons: B-FREE1, B10
    Result: 225 HUF, Unused: [B10]
11. **Coupon not better than existing discount**
    Cart: 3 kg banana
    Period: Normal
    Coupons: B5
    Result: 1215 HUF, Unused: [B5]
12. **Free coupon removes discount eligibility**
    Cart: 2.2 kg apple
    Period: Normal
    Coupons: A-FREE1
    Result: 600 HUF, Unused: []
13. **Multiple coupons for same product – only first valid**
    Cart: 1 kg banana
    Period: Normal
    Coupons: B5, B-FREE1
    Result: 430 HUF, Unused: [B-FREE1]
14. **Coupon for wrong product**
    Cart: 1 kg apple
    Period: Normal
    Coupons: B10
    Result: 500 HUF, Unused: [B10]

Development Requirements

- The `getCartPrice` method must consider coupons.
- Coupon types must be handled in an easily extendable structure (e.g., class hierarchy, type-identifier-based rules).
- For each coupon, the system must decide:
  - **use it → apply the discount**,
  - **cannot use it → return it**.

Note

The above examples show only a few common coupon types. In the future, many new forms of coupons are expected to appear. The system must remain flexible in handling them.

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

To make the requirements clearer, here are the unit tests for CR2.

Java Unit Tests

```java
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 StoreCR2Tests {
    static Store target;
    static Period normal;

    @BeforeAll
    static void init() {
        target = new Store();
        normal = new Period("Normal");
        normal.setUnitPrice(Product.APPLE, 500.0);
        normal.setUnitPrice(Product.BANANA, 450.0);
        normal.setDiscount(Product.APPLE, 5.0, 0.1);
        normal.setDiscount(Product.APPLE, 20.0, 0.15);
        normal.setDiscount(Product.BANANA, 2.0, 0.1);
        target.addPeriod(normal);
    }

    @Test
    void test_cr2_example1_b10_not_applicable() {
        Cart cart = new Cart(List.of(new Item(Product.BANANA, 2.0)));
        PriceInfo info = target.getCartPrice(cart, normal, List.of("B10"));
        assertEquals(810.0, info.getPrice(), 0.001);
        assertEquals(List.of("B10"), info.getUnusedCoupons());
    }
```

```
    @Test
    void test_cr2_example2_a10_applicable() {
        Cart cart = new Cart(List.of(new Item(Product.APPLE, 1.0)));
        PriceInfo info = target.getCartPrice(cart, normal, List.of("A10"));
        assertEquals(450.0, info.getPrice(), 0.001);
        assertEquals(List.of(), info.getUnusedCoupons());
    }

    @Test
    void test_cr2_example3_wrong_order() {
        Cart cart = new Cart(List.of(new Item(Product.APPLE, 1.0)));
        PriceInfo info = target.getCartPrice(cart, normal, List.of("A5",
"A10"));
        assertEquals(475.0, info.getPrice(), 0.001);
        assertEquals(List.of("A10"), info.getUnusedCoupons());
    }

    @Test
    void test_cr2_example4_better_order() {
        Cart cart = new Cart(List.of(new Item(Product.APPLE, 1.0)));
        PriceInfo info = target.getCartPrice(cart, normal, List.of("A10",
"A5"));
        assertEquals(450.0, info.getPrice(), 0.001);
        assertEquals(List.of("A5"), info.getUnusedCoupons());
    }

    @Test
    void test_cr2_example5_free_apple() {
        Cart cart = new Cart(List.of(new Item(Product.APPLE, 0.5)));
        PriceInfo info = target.getCartPrice(cart, normal, List.of("A-
FREE1"));
        assertEquals(0.0, info.getPrice(), 0.001);
        assertEquals(List.of(), info.getUnusedCoupons());
    }

    @Test
    void test_cr2_example6_free_and_quantity() {
        Cart cart = new Cart(List.of(new Item(Product.BANANA, 3.0)));
        PriceInfo info = target.getCartPrice(cart, normal, List.of("B-
FREE1"));
        assertEquals(810.0, info.getPrice(), 0.001);
        assertEquals(List.of(), info.getUnusedCoupons());
    }

    @Test
    void test_cr2_example7_two_different_coupons() {
        Cart cart = new Cart(List.of(
                new Item(Product.APPLE, 1.0),
                new Item(Product.BANANA, 1.0)
        ));
        PriceInfo info = target.getCartPrice(cart, normal, List.of("A5",
"B5"));
        assertEquals(905.0, info.getPrice(), 0.001);
        assertEquals(List.of(), info.getUnusedCoupons());
    }

    @Test
    void test_cr2_example8_free_excluded_by_a10() {
        Cart cart = new Cart(List.of(new Item(Product.APPLE, 1.0)));
        PriceInfo info = target.getCartPrice(cart, normal, List.of("A10",
"A-FREE1"));
```

```
        assertEquals(450.0, info.getPrice(), 0.001);
        assertEquals(List.of("A-FREE1"), info.getUnusedCoupons());
    }

    @Test
    void test_cr2_example9_free_first_then_no_merge() {
        Cart cart = new Cart(List.of(new Item(Product.APPLE, 1.0)));
        PriceInfo info = target.getCartPrice(cart, normal, List.of("A-
FREE1", "A10"));
        assertEquals(0.0, info.getPrice(), 0.001);
        assertEquals(List.of("A10"), info.getUnusedCoupons());
    }

    @Test
    void test_cr2_example10_free_then_percentage_not_applicable() {
        Cart cart = new Cart(List.of(new Item(Product.BANANA, 1.5)));
        PriceInfo info = target.getCartPrice(cart, normal, List.of("B-
FREE1", "B10"));
        double expected = roundTo5(0.5 * 450.0); // no discount
        assertEquals(expected, info.getPrice(), 0.001);
        assertEquals(List.of("B10"), info.getUnusedCoupons());
    }

    @Test
    void test_cr2_example11_coupon_worse_than_discount() {
        Cart cart = new Cart(List.of(new Item(Product.BANANA, 3.0)));
        PriceInfo info = target.getCartPrice(cart, normal, List.of("B5"));
// already has 10% discount
        double expected = roundTo5(3.0 * 450.0 * 0.9);
        assertEquals(expected, info.getPrice(), 0.001);
        assertEquals(List.of("B5"), info.getUnusedCoupons());
    }

    @Test
    void test_cr2_example12_free_removes_discount() {
        Cart cart = new Cart(List.of(new Item(Product.APPLE, 2.2)));
        PriceInfo info = target.getCartPrice(cart, normal, List.of("A-
FREE1"));
        double expected = roundTo5(1.2 * 500.0);
        assertEquals(expected, info.getPrice(), 0.001);
        assertEquals(List.of(), info.getUnusedCoupons());
    }

    @Test
    void test_cr2_example13_two_coupons_same_product() {
        Cart cart = new Cart(List.of(new Item(Product.BANANA, 1.0)));
        PriceInfo info = target.getCartPrice(cart, normal, List.of("B5",
"B-FREE1"));
        double expected = roundTo5(450.0 * 0.95);
        assertEquals(expected, info.getPrice(), 0.001);
        assertEquals(List.of("B-FREE1"), info.getUnusedCoupons());
    }

    @Test
    void test_cr2_example14_coupon_for_wrong_product() {
        Cart cart = new Cart(List.of(new Item(Product.APPLE, 1.0)));
        PriceInfo info = target.getCartPrice(cart, normal, List.of("B10"));
        assertEquals(500.0, info.getPrice(), 0.001);
        assertEquals(List.of("B10"), info.getUnusedCoupons());
    }
```

```
    // Helper function for rounding to nearest 5
    private double roundTo5(double amount) {
        double remainder = amount % 10.0;
        if (remainder < 2.5) return amount - remainder;
        if (remainder < 5.0) return amount - remainder + 5.0;
        if (remainder < 7.5) return amount - remainder + 5.0;
        return amount - remainder + 10.0;
    }
}
```