CR6 – Introducing Card Payment

Introduction

The GoodPrice store's idea box is like a bottomless bag: messages just keep piling up.
The majority of complaints sound like this: *"Why can't we pay by card?!"*

Finally, the management made the decision: yes, card payment will be introduced!
Moreover, to make customers even happier, an extra discount will be granted when paying by card.

Task

Your task is to modify the existing solution according to CR6!

It may be useful to adapt the unit tests of previous CRs to match the new interface requirements.
We recommend using the unit tests at the end of this subsection, as well as vibe coding techniques to complete the task!

Requirements

The signature of the `getCartPrice` method now accepts four parameters:

```
PriceInfo getCartPrice(Cart c, Period p, List<String> cc, PaymentMethod m)
```

The **PaymentMethod** can be of two types:

- **cash**
- **card**

Until now, only cash payments were accepted, but now card payments must also be supported.

Rules for card payment:

- The rounding rule to 5 HUF (cash-only) does **not** apply to card payments.
- For card payments, an additional **0.5% discount** is applied to the final amount.
- This is called the **card bonus**, which may later become configurable.
- The discount must be applied **after all other discounts and coupons**.
- After subtracting the card bonus, the payable amount must be **rounded to 0.1 HUF (10 fillér)** according to the following rules:

- Between 0.0–0.049 HUF → round down to the nearest 0.1 HUF
- Between 0.05–0.099 HUF → round up to the nearest 0.1 HUF

Examples and 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 StoreCR6Tests {

    static Store target;
    static Period normal;

    @BeforeAll
    static void init() {
        target = new Store();
        normal = new Period("Normal");
        // Settings known from CR0/CR1
        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);
    }

    // --- Helper: rounding to 5 HUF (cash)
    private double roundTo5(double amount) {
        double remainder = amount % 10.0;
        double base = Math.floor(amount / 10.0) * 10.0;
        if (remainder < 2.5) return base;
        if (remainder < 5.0) return base + 5.0;
        if (remainder < 7.5) return base + 5.0;
        return base + 10.0;
    }

    // --- Helper: rounding to 0.1 HUF (card)
    // 0.0–0.049 HUF → down; 0.05–0.099 HUF → up
    private double roundToTenth(double amount) {
        return Math.round(amount * 10.0) / 10.0;
    }

    // 1) Cash: behavior unchanged, 5 HUF rounding (1 kg apple)
    @Test
    void test_cr6_cash_unchanged_1kg_apple() {
        Cart cart = new Cart(List.of(new Item(Product.APPLE, 1.0))); // 500
HUF
        PriceInfo price = target.getCartPrice(cart, normal, List.of(),
PaymentMethod.CASH);
        assertEquals(roundTo5(500.0), price.getAmount(), 0.001); // 500
        assertEquals(List.of(), price.getUnusedCoupons());
    }

    // 2) Card: no 5 HUF rounding, 0.5% discount at the end, then 0.1 HUF
```

```
rounding (1 kg apple)
    @Test
    void test_cr6_card_1kg_apple_bonus_and_rounding() {
        Cart cart = new Cart(List.of(new Item(Product.APPLE, 1.0))); // 500
HUF
        PriceInfo price = target.getCartPrice(cart, normal, List.of(),
PaymentMethod.CARD);
        assertEquals(roundToTenth(500.0 * 0.995), price.getAmount(),
0.0001); // 497.5
        assertEquals(List.of(), price.getUnusedCoupons());
    }

    // 3) Card + quantity discount (2 kg bananas → 10% discount) + 0.5%
bonus
    @Test
    void test_cr6_card_with_quantity_discount_and_bonus() {
        Cart cart = new Cart(List.of(new Item(Product.BANANA, 2.0))); //
900 → 10% = 810
        double afterDiscount = 810.0;
        double card = afterDiscount * 0.995; // 805.95
        PriceInfo price = target.getCartPrice(cart, normal, List.of(),
PaymentMethod.CARD);
        assertEquals(roundToTenth(card), price.getAmount(), 0.0001); //
806.0
    }

    // 4) Card: rounding down to 0.1 HUF (1.789 kg apples)
    @Test
    void test_cr6_card_rounding_down() {
        Cart cart = new Cart(List.of(new Item(Product.APPLE, 1.789))); //
894.5
        double after = 894.5 * 0.995; // 890.0275 → 890.0
        PriceInfo price = target.getCartPrice(cart, normal, List.of(),
PaymentMethod.CARD);
        assertEquals(roundToTenth(after), price.getAmount(), 0.0001); //
890.0
    }

    // 5) Card: rounding up to 0.1 HUF (1.5 kg apples)
    @Test
    void test_cr6_card_rounding_up() {
        Cart cart = new Cart(List.of(new Item(Product.APPLE, 1.5))); // 750
        double after = 750.0 * 0.995; // 746.25 → 746.3
        PriceInfo price = target.getCartPrice(cart, normal, List.of(),
PaymentMethod.CARD);
        assertEquals(roundToTenth(after), price.getAmount(), 0.0001); //
746.3
    }

    // 6) Card + coupon: A10 on apples, then 0.5% bonus
    @Test
    void test_cr6_card_coupon_then_bonus() {
        Cart cart = new Cart(List.of(new Item(Product.APPLE, 1.0))); // 500
        PriceInfo price = target.getCartPrice(cart, normal, List.of("A10"),
PaymentMethod.CARD);
        assertEquals(roundToTenth(450.0 * 0.995), price.getAmount(),
0.0001); // 447.8
        assertEquals(List.of(), price.getUnusedCoupons());
    }

    // 7) Card + X10 (non-combinable) + other coupons: only X10 applies,
then 0.5% bonus
    @Test
    void test_cr6_card_x10_noncombinable_then_bonus() {
        Cart cart = new Cart(List.of(
                new Item(Product.APPLE, 2.0), // 1000
                new Item(Product.BANANA, 1.0) // 450
        )); // total: 1450
        PriceInfo price = target.getCartPrice(cart, normal, List.of("X10",
"A5", "B5"), PaymentMethod.CARD);
        assertEquals(roundToTenth(1450.0 * 0.90 * 0.995),
price.getAmount(), 0.0001); // 1298.5
        assertEquals(List.of("A5", "B5"), price.getUnusedCoupons());
    }

    // 8) Comparison: same cart with cash vs. card (A10 on apples)
    @Test
    void test_cr6_cash_vs_card_same_cart() {
        Cart cart = new Cart(List.of(new Item(Product.APPLE, 1.0))); // 500
        PriceInfo cash = target.getCartPrice(cart, normal, List.of("A10"),
PaymentMethod.CASH);
        assertEquals(roundTo5(450.0), cash.getAmount(), 0.001); // 450.0

        PriceInfo card = target.getCartPrice(cart, normal, List.of("A10"),
PaymentMethod.CARD);
        assertEquals(roundToTenth(450.0 * 0.995), card.getAmount(),
0.0001); // 447.8
    }

    // 9) Card: larger cart with quantity discount (3 kg bananas → 10%),
then bonus
    @Test
    void test_cr6_card_large_cart_quantity_discount() {
        Cart cart = new Cart(List.of(new Item(Product.BANANA, 3.0))); //
1350 → 10% = 1215
        double after = 1215.0 * 0.995; // 1208.925 → 1208.9
        PriceInfo price = target.getCartPrice(cart, normal, List.of(),
PaymentMethod.CARD);
        assertEquals(roundToTenth(after), price.getAmount(), 0.0001); //
1208.9
    }

    // 10) Card + "free" type coupon (e.g., A-FREE1), bonus applies to
final amount
    @Test
    void test_cr6_card_free_coupon_then_bonus() {
        Cart cart = new Cart(List.of(new Item(Product.APPLE, 0.5)));
        PriceInfo price = target.getCartPrice(cart, normal, List.of("A-
FREE1"), PaymentMethod.CARD);
        assertEquals(0.0, price.getAmount(), 0.0001);
        assertEquals(List.of(), price.getUnusedCoupons());
    }
}
```