CR7 – Introducing Gift Bags and Coupons

Introduction

The financial officers of the GoodPrice store sighed: it turned out that the 0.5% card bonus had been too generous. Therefore, management temporarily sets the card bonus to 0.0% (we'll see later). However, marketing never sleeps: *"If there's no bonus, at least there should be free paper bags… and let's also give gift coupons!"* – this was the conclusion of the weekly marketing meeting. The implementation awaits you!

Task

Your task is to modify the existing solution according to CR7.
This CR is the final element of the series – if you have reached this point, you can be proud of yourself!

Recommendations:

- Because of the new signature introduced in CR6, it may be worth aligning earlier unit tests with the new interface.
- Use the unit tests at the end of this section and the vibe-driven coding tools.

Affected Interfaces

Since CR6, the computational entry point is:

```
ArInfo getCartPrice(Cart c, Period p, List<String> coupons, PaymentMethod
m)
```

where `PaymentMethod` can take two values: `CASH`, `CARD`.

Changes in the ArInfo Structure

The former two fields:

- amount payable (double, in HUF),
- list of unused coupons (List),

are extended with the following:

- number of gift paper bags (int),
- list of gift coupons (List).

Payment Rules

- **Cash:** 5 HUF rounding remains (CR0 rule).
- **Card:**
  - card bonus = 0.0% (CR7),
  - no 5 HUF rounding,
  - final amount must be rounded to 10 fillér (0.0–4.9 fillér → down, 5.0–9.9 fillér → up).

New Gift Rules

Paper Bags

- The card bonus is 0.0%.
- For every 5 kg of purchased products (apples + bananas combined), the customer receives one gift paper bag. Up to 5 kg no gift is given, only full 5 kg multiples qualify.
- Important: although the **A-FREE1** and **B-FREE1** coupons reduce the payable quantity, this is only a logical reduction. The basis for gift bags is the **physical weight**.
- The number of paper bags must be returned in the `ArInfo` structure.
- The list of gift coupons must also be returned in the `ArInfo` structure. If no gift coupon is given, this list is empty.
- The received quantity must be returned in the `ArInfo.giftBagCount` field.

Gift Coupons

- If the purchase total is greater than or equal to 20,000 HUF, then for every 20,000 HUF the customer receives one gift coupon.
- Example: if the payable amount is 58,000 HUF, then 2 gift coupons are given.
- Coupons are drawn according to the following probability:
  - **A10 or B10:** 20% chance.
  - **A-FREE1 or B-FREE1:** 10% chance.
  - **A5-MAX10 or B5-MAX10:** 5% chance.
  - **X5:** 5% chance.
  - **A5-MAX15 or B5-MAX15:** 2% chance.
  - **X10:** 2% chance.
  - **X5-MAX10:** 1% chance.
  - **KUPON-2000-ULTRAMAX:** 1% chance.
  - **A5 or B5:** remaining probability.
- These chances may change in the future.
- The coupon types must be returned in `ArInfo.giftCoupons`. If no coupon is earned, the list is empty.

Examples

- **Bag threshold**
  Cart: 4.9 kg apples + 0.1 kg bananas → 5.0 kg
  Bags: 1 (based on physical weight).
  Gift coupon: depends on total (1 for each 20,000 HUF).
- **Multiple multiples**
  Cart: 6.8 kg apples + 3.5 kg bananas → 10.3 kg → 2 bags.
- **FREE coupon does not reduce physical weight**
  Cart: 5.2 kg apples; Coupons: [A-FREE1]
  Bags: 1 (not 0), because physical weight is 5.2 kg.
- **Gift coupon count**
  Cash, large cart → total 50,600 HUF → 2 coupons (drawn from the list).
- **Gift coupon threshold**
  Cash: ~20,305 HUF → 1 coupon.
  Card: ~19,999.9 HUF → 0 coupons (CR7 bonus is 0%, only 10 fillér rounding applies).
- **ULTRAMAX zeroing**
  Cart: 3 kg apples; Coupon: [KUPON-2000-ULTRAMAX] → 0 HUF
  Gift coupons: 0 (since < 20,000 HUF).
  Bags: 0 (physical weight 3.0 kg).
- **Large physical weight, many bags**
  Cart: 60.2 kg apples + 63.2 kg bananas → 123.4 kg → 24 bags.
  Gift coupons: according to total / 20,000 HUF.

Development Requirements (Summary)

- Extend `ArInfo`:
- `int giftBagCount;`
- `List<String> giftCoupons;`
- Card logic: bonus 0.0%, rounding to 10 fillér (as in CR6, but without bonus).
- Bag calculation: `floor((apples_kg + bananas_kg) / 5.0)` – based on physical weight.
- Gift coupon calculation: `floor(total / 20000.0)`; coupon types drawn with given distribution.
- Deterministic testing: recommended to inject RNG or use seed for reproducible tests.
- Backward compatibility: earlier CR logic (quantity discounts, coupons, MAX, X, ULTRAMAX, rounding) remains unchanged.

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 java.util.Set;
import org.store.*;

class StoreCR7Tests {
    static Store target;
    static Period normal;

    // Allowed gift coupon type codes (per spec)
    static final Set<String> ALLOWED_GIFTS = Set.of(
            "A10", "B10",
            "A-FREE1", "B-FREE1",
            "A5-MAX10", "B5-MAX10",
            "X5", "X10", "X5-MAX10",
            "A5-MAX15", "B5-MAX15",
            "KUPON-2000-ULTRAMAX",
            "A5", "B5"
    );

    @BeforeAll
    static void init() {
        target = new Store();
        normal = new Period("Normal");
        // CR0/CR1 baseline configuration
        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) per CR0/CR1 rule
    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) per CR6 rule (CR7 keeps this
but with 0% bonus)
    private double roundTo10Filler(double amount) {
        return Math.round(amount * 10.0) / 10.0;
    }

    // 1) Gift paper bag: first full 5 kg → 1 bag (based on PHYSICAL
weight!)
    @Test
    void test_cr7_1_bag_threshold_5kg() {
        // 4.9 kg apples + 0.1 kg bananas = 5.0 kg total → 1 bag
        Cart cart = new Cart(List.of(
                new Item(Product.APPLE, 4.9),
                new Item(Product.BANANA, 0.1)
        ));
        PriceInfo info = target.getCartPrice(cart, normal, List.of(),
PaymentMethod.CASH);
        assertEquals(1, info.getGiftBagCount());
    }

    // 2) Gift paper bags: multiple multiples (10.3 kg → 2 bags)
    @Test
    void test_cr7_2_bags_multiple_multiples() {
        // Total weight = 6.8 + 3.5 = 10.3 → floor(10.3/5)=2
        Cart cart = new Cart(List.of(
                new Item(Product.APPLE, 6.8),
                new Item(Product.BANANA, 3.5)
        ));
        PriceInfo info = target.getCartPrice(cart, normal, List.of(),
PaymentMethod.CASH);
        assertEquals(2, info.getGiftBagCount());
    }

    // 3) "FREE" coupon only reduces payable quantity logically,
    //    gift bag calculation uses PHYSICAL weight (unchanged).
    @Test
    void test_cr7_3_bags_free_coupon_does_not_reduce_physical_weight() {
        // 5.2 kg apples with A-FREE1: payable may drop by up to 1 kg,
        // but physical weight is still 5.2 → 1 bag
        Cart cart = new Cart(List.of(new Item(Product.APPLE, 5.2)));
        PriceInfo info = target.getCartPrice(cart, normal, List.of("A-
FREE1"), PaymentMethod.CASH);
        assertEquals(1, info.getGiftBagCount());
    }

    // 4) Gift coupon count: 58,000 HUF total → 2 gift coupons (drawn from
allowed set)
    @Test
    void test_cr7_4_gift_coupon_count_58000() {
        // Construct a large total near the example:
        // Apples 100 kg → 100*500=50,000 → -15% (>=20kg) = 42,500
        // Bananas 20 kg → 20*450=9,000 → -10% (>=2kg) = 8,100
        // Sum: 50,600 HUF → cash rounding keeps 50,600
        Cart cart = new Cart(List.of(
                new Item(Product.APPLE, 100.0),
                new Item(Product.BANANA, 20.0)
        ));
        PriceInfo info = target.getCartPrice(cart, normal, List.of(),
PaymentMethod.CASH);

        int expectedCount = (int) Math.floor(info.getAmount() / 20000.0);
        assertEquals(expectedCount, info.getGiftCoupons().size());
        // All coupon codes must be from the allowed set

assertTrue(info.getGiftCoupons().stream().allMatch(ALLOWED_GIFTS::contains)
);
        // In this specific setup the example expects 2 coupons at ~50,600
HUF
        assertEquals(2, info.getGiftCoupons().size());
    }

    // 5) Gift coupon threshold:
    //    Cash: ~20,305 HUF → 1 coupon.
    //    Card: ~19,999.9 HUF → 0 coupons (CR7 card bonus is 0%, only 0.1
HUF rounding applies).
    @Test
    void test_cr7_5_gift_coupon_threshold_values() {
        // Construct around 20,000 HUF with CASH:
        // Apples 23 kg → 23*500=11,500 → -15% = 9,775
        // Bananas 26 kg → 26*450=11,700 → -10% = 10,530
        // Sum: 20,305 → cash rounding: 20,305
        Cart cashCart = new Cart(List.of(
                new Item(Product.APPLE, 23.0),
                new Item(Product.BANANA, 26.0)
        ));
        PriceInfo cashInfo = target.getCartPrice(cashCart, normal,
List.of(), PaymentMethod.CASH);

        int expectedCashCoupons = (int) Math.floor(cashInfo.getAmount() /
20000.0);
        assertEquals(expectedCashCoupons,
cashInfo.getGiftCoupons().size());
        assertEquals(1, cashInfo.getGiftCoupons().size());

assertTrue(cashInfo.getGiftCoupons().stream().allMatch(ALLOWED_GIFTS::conta
ins));

        // Construct a well-below-20000 case with CARD (bonus=0.0% in CR7):
        Cart cardCart = new Cart(List.of(
                new Item(Product.APPLE, 10.0),    // apples: 10kg → 10%
(>=5kg) → 10*500=5000 → 4500
                new Item(Product.BANANA, 20.0)    // bananas: 20kg → 10%
(>=2kg) → 20*450=9000 → 8100
        ));
        PriceInfo cardInfo = target.getCartPrice(cardCart, normal,
List.of(), PaymentMethod.CARD);
        assertTrue(cardInfo.getAmount() < 20000.0);
        assertEquals(0, cardInfo.getGiftCoupons().size());
    }

    // 6) Gift coupon list is empty if final total is 0 HUF (zeroed by
ULTRAMAX)
    @Test
    void test_cr7_6_ultramax_zeroes_and_no_gift_coupons() {
        // A cart that ULTRAMAX can zero
        Cart cart = new Cart(List.of(new Item(Product.APPLE, 3.0))); //
3*500=1500
        PriceInfo info = target.getCartPrice(
                cart, normal, List.of("KUPON-2000-ULTRAMAX"),
PaymentMethod.CASH
        );
        assertEquals(0.0, info.getAmount(), 0.0001);
        assertEquals(List.of(), info.getGiftCoupons());
        // Bags are still based on physical weight (3.0 kg → 0 bags)
        assertEquals(0, info.getGiftBagCount());
    }

    // 7) Gift coupons – every drawn code must be from the allowed set
    @Test
    void test_cr7_7_gift_coupons_from_allowed_set() {
        // Large total to get multiple coupons
        Cart cart = new Cart(List.of(
                new Item(Product.APPLE, 200.0),  // 200*500=100000 → -15% =
85000
                new Item(Product.BANANA, 50.0)   // 50*450=22500 → -10% =
20250
        ));
        PriceInfo info = target.getCartPrice(cart, normal, List.of(),
PaymentMethod.CASH);
        int expected = (int) Math.floor(info.getAmount() / 20000.0);
        assertEquals(expected, info.getGiftCoupons().size());

assertTrue(info.getGiftCoupons().stream().allMatch(ALLOWED_GIFTS::contains)
);
    }

    // 8) CR7: card bonus = 0.0% (CR6 had 0.5%) → now card has NO extra
```

```
discount.
    //    Compare cash vs. card: difference can only come from rounding.
    @Test
    void test_cr7_8_card_bonus_zeroed() {
        // Pick a value where rounding differs:
        // 1.333 kg apples → 1.333*500 = 666.5
        Cart cart = new Cart(List.of(new Item(Product.APPLE, 1.333)));
        PriceInfo cash = target.getCartPrice(cart, normal, List.of(),
PaymentMethod.CASH);
        PriceInfo card = target.getCartPrice(cart, normal, List.of(),
PaymentMethod.CARD);

        // Cash: 5 HUF rounding (666.5 → 665.0 per CR0 rule)
        assertEquals(roundTo5(666.5), cash.getAmount(), 0.001);

        // Card: NO 0.5% bonus in CR7; only 0.1 HUF rounding → stays 666.5
        assertEquals(roundTo10Filler(666.5), card.getAmount(), 0.0001);

        // Ensure there's truly no 0.5% deduction (which would yield 666.5
* 0.995)
        assertNotEquals(roundTo10Filler(666.5 * 0.995), card.getAmount(),
0.0001);
    }

    // 9) Large cart: many bags and a matching number of gift coupons
    @Test
    void test_cr7_9_many_bags_and_gift_coupons() {
        // Total physical weight = 60.2 + 63.2 = 123.4 → floor(123.4/5)=24
bags
        Cart cart = new Cart(List.of(
                new Item(Product.APPLE, 60.2),
                new Item(Product.BANANA, 63.2)
        ));
        PriceInfo info = target.getCartPrice(cart, normal, List.of(),
PaymentMethod.CASH);
        assertEquals(24, info.getGiftBagCount());
        int expected = (int) Math.floor(info.getAmount() / 20000.0);
        assertEquals(expected, info.getGiftCoupons().size());

assertTrue(info.getGiftCoupons().stream().allMatch(ALLOWED_GIFTS::contains)
);
    }

    // 10) Mixed: ULTRAMAX + large physical weight
    //     Final amount may be low (→ 0 gift coupons), but bags are granted
by physical weight.
    @Test
    void test_cr7_10_ultramax_and_heavy_cart() {
        Cart cart = new Cart(List.of(
                new Item(Product.APPLE, 8.0),   // 8*500=4000 → -10%
(>=5kg) = 3600
                new Item(Product.BANANA, 7.0)   // 7*450=3150 → -10%
(>=2kg) = 2835
        ));
        // Two ULTRAMAX coupons: after discounts 3600+2835=6435
        // 1st ULTRAMAX → 4435; 2nd → 2435; cash rounding → 2435
        PriceInfo info = target.getCartPrice(
                cart, normal,
                List.of("KUPON-2000-ULTRAMAX", "KUPON-2000-ULTRAMAX"),
                PaymentMethod.CASH
        );
        assertTrue(info.getAmount() >= 0.0);
        // Physical weight: 8 + 7 = 15 kg → 3 bags
        assertEquals(3, info.getGiftBagCount());
        // Gift coupons only if >= 20000 → none here
        assertEquals(0, info.getGiftCoupons().size());
    }
}
```