Base Task – CR0 – Checkout Program at the GoodPrice Store

Your task is to implement the basic version of the checkout system for the GoodPrice store. The central element of the checkout system is the following method:

```
double getCartPrice(Cart cart)
```

This method calculates the total price of a purchase in Hungarian Forints. The return value is therefore a number representing the final amount to be paid in the case of a cash payment.

The following rules must be considered for the task:

Cart

The cart can contain two types of products:

- apple, specified in kilograms (e.g., 1.5 kg),
- banana, specified in kilograms (e.g., 3 kg),

The same product may appear multiple times in the cart; in such cases, the apple quantities must be added together, and the banana quantities must be added together.

Base Prices

The base prices of the products are as follows:

- 1 kg apple → 500 HUF
- 1 kg banana → 450 HUF

Quantity Discounts

During normal periods in the GoodPrice store, the following discounts apply:

Apple:

- If at least 5 kg is purchased → 10% discount on the total apple price.
- If at least 20 kg is purchased → 15% discount on the total apple price.

Banana:

- If at least 2 kg is purchased → 10% discount on the total banana price.

Payment

Currently, the system only supports cash payments.

Rounding Rules (for cash payment)

The final amount must be rounded to the nearest 5 HUF according to the following rules:

- If the amount ends with 0.01–2.49 → round down to the nearest 0.
- If the amount ends with 2.50–4.99 → round up to the nearest 5.
- If the amount ends with 5.01–7.49 → round down to the nearest 5.
- If the amount ends with 7.50–9.99 → round up to the nearest 0.

Examples:

- 2782 HUF → 2780 HUF
- 2783 HUF → 2785 HUF
- 2787 HUF → 2785 HUF
- 2789 HUF → 2790 HUF

Example Carts and Prices

1. Cart: 1 kg apple

Price: 1 × 500 HUF = 500 HUF
Discount: none
Rounding: not needed
Final price: 500 HUF

2. Cart: 2 kg apple

Price: 2 × 500 HUF = 1000 HUF
Discount: none
Rounding: not needed
Final price: 1000 HUF

3. Cart: 5 kg apple

Price: 5 × 500 HUF = 2500 HUF
Discount: 10% → 250 HUF

Discounted price: 2250 HUF
Rounding: 2250 HUF → 2250 HUF
Final price: 2250 HUF

4. Cart: 20 kg apple

Price: 20 × 500 HUF = 10000 HUF
Discount: 15% → 1500 HUF
Discounted price: 8500 HUF
Rounding: not needed
Final price: 8500 HUF

5. Cart: 1 kg apple, 1 kg banana

Apple: 500 HUF (no discount)
Banana: 450 HUF (no discount)
Total: 950 HUF
Rounding: 950 HUF → 950 HUF
Final price: 950 HUF

6. Cart: 2 kg banana

Price: 2 × 450 HUF = 900 HUF
Discount: 10% → 90 HUF
Discounted price: 810 HUF
Rounding: 810 HUF → 810 HUF
Final price: 810 HUF

7. Cart: 1 kg apple, 2 kg apple, 2 kg apple

Total: 1 + 2 + 2 = 5 kg apple
Price: 5 × 500 HUF = 2500 HUF
Discount: 10% → 250 HUF
Discounted price: 2250 HUF
Rounding: 2250 HUF → 2250 HUF
Final price: 2250 HUF

8. Cart: 2 kg apple, 2 kg banana

Apple: 2 × 500 = 1000 HUF (no discount)
Banana: 2 × 450 = 900 HUF → 10% discount = 810 HUF
Total: 1000 + 810 = 1810 HUF
Rounding: 1810 HUF → 1810 HUF
Final price: 1810 HUF

9. Cart: 1.01 kg apple

Price: 1.01 × 500 = 505.00 HUF
Discount: none

Rounding: 505.00 → 505 HUF (no change)
Final price: 505 HUF

10. Cart: 1.99 kg banana

Price: 1.99 × 450 = 895.50 HUF
Discount: none
Rounding: 895.50 → 895 HUF (amount ending 5.01–7.49 → down to nearest 5)
Final price: 895 HUF

11. Cart: 1.789 kg apple

Price: 1.789 × 500 = 894.50 HUF
Discount: none
Rounding: 894.50 → 895 HUF (amount ending 2.50–4.99 → up to nearest 5)
Final price: 895 HUF

12. Cart: 2.01 kg banana

Price: 2.01 × 450 = 904.50 HUF
Discount: 10% → 90.45 HUF
Discounted price: 904.50 − 90.45 = 814.05 HUF
Rounding: 814.05 → 815 HUF (amount ending 2.50–4.99 → up to nearest 5)
Final price: 815 HUF

13. Cart: 3.333 kg apple + 1.777 kg banana

Apple: 3.333 × 500 = 1666.50 HUF (no discount)
Banana: 1.777 × 450 = 799.65 HUF (no discount)
Total: 1666.50 + 799.65 = 2466.15 HUF
Rounding: 2466.15 → 2465 HUF (amount ending 5.01–7.49 → down to nearest 5)
Final price: 2465 HUF

14. Cart: 5.001 kg apple

Price: 5.001 × 500 = 2500.50 HUF
Discount: 10% → 250.05 HUF
Discounted price: 2250.45 HUF
Rounding: 2250.45 → 2250 HUF (amount ending 0.01–2.49 → down to nearest 0)
Final price: 2250 HUF

15. Cart: 3.789 kg apple and 2.777 kg banana

Apple: 3.789 × 500 = 1894.50 HUF
Banana: 2.777 × 450 = 1249.65 HUF
Discount: banana 10% (2.777 > 2 kg)
Discounted banana price: 1124.685 HUF
Total: 1894.50 + 1124.68 = 3019.185 HUF

Rounding: 3019.185 → 3020 HUF (amount ending 7.50–9.99 → up to nearest 0)
Final price: 3020 HUF

In the future, system requirements may change, for example, with the introduction of new payment methods, coupons, or seasonal discounts. These will be documented separately in Change Request (CR) form.

To be more concrete, we provide unit tests corresponding to the examples, using JUnit 5 syntax.

```
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Test;
import java.util.List;
import org.store.Store;
import org.store.Cart;
import org.store.Item;
import org.store.Product;
import org.store.ProductPrice;

class StoreCR0Tests {

    @Test
    void test_cr0_example1_1kgApple() {
        double unitPrice = 500.0;
        double quantity = 1.0;
        Store target = new Store(Product.APPLE, unitPrice);
        double expected = unitPrice * quantity;
        Cart cart = new Cart(List.of(new Item(Product.APPLE, quantity)));
        double actual = target.getCartPrice(cart);
        assertEquals(expected, actual, 0.001);
    }

    @Test
    void test_cr0_example2_2kgApple() {
        double unitPrice = 500.0;
        double quantity = 2.0;
        Store target = new Store(Product.APPLE, unitPrice);
        double expected = unitPrice * quantity;
        Cart cart = new Cart(List.of(new Item(Product.APPLE, quantity)));
        double actual = target.getCartPrice(cart);
        assertEquals(expected, actual, 0.001);
    }

    @Test
    void test_cr0_example3_5kgApple_withDiscount() {
        double unitPrice = 500.0;
        double threshold1 = 5.0;
        double discount1 = 0.1;
        double quantity = threshold1;
        Store target = new Store(Product.APPLE, unitPrice);
        target.setDiscount(Product.APPLE, threshold1, discount1);
        double expected = unitPrice * quantity * (1.0 - discount1);
        Cart cart = new Cart(List.of(new Item(Product.APPLE, quantity)));
        double actual = target.getCartPrice(cart);
        assertEquals(expected, actual, 0.001);
    }
```

```java
    @Test
    void test_cr0_example4_20kgApple_withDiscount() {
        double unitPrice = 500.0;
        double threshold1 = 5.0;
        double discount1 = 0.1;
        double threshold2 = 20.0;
        double discount2 = 0.15;
        double quantity = threshold2;
        Store target = new Store(Product.APPLE, unitPrice);
        target.setDiscount(Product.APPLE, threshold1, discount1);
        target.setDiscount(Product.APPLE, threshold2, discount2);
        double expected = unitPrice * quantity * (1.0 - discount2);
        Cart cart = new Cart(List.of(new Item(Product.APPLE, quantity)));
        double actual = target.getCartPrice(cart);
        assertEquals(expected, actual, 0.001);
    }

    @Test
    void test_cr0_example5_1kgApple_1kgBanana() {
        double unitPriceApple = 500.0;
        double quantityApple = 1.0;
        double unitPriceBanana = 450.0;
        double quantityBanana = 1.0;
        Store target = new Store(List.of(
                new ProductPrice(Product.APPLE, unitPriceApple),
                new ProductPrice(Product.BANANA, unitPriceBanana)
        ));
        Cart cart = new Cart(List.of(
                new Item(Product.APPLE, quantityApple),
                new Item(Product.BANANA, quantityBanana)
        ));
        double expected = unitPriceApple * quantityApple
                + unitPriceBanana * quantityBanana;
        double actual = target.getCartPrice(cart);
        assertEquals(expected, actual, 0.001);
    }

    @Test
    void test_cr0_example6_2kgBanana_withDiscount() {
        double unitPrice = 450.0;
        double threshold1 = 2.0;
        double discount1 = 0.1;
        double quantity = threshold1;
        Store target = new Store(Product.BANANA, unitPrice);
        target.setDiscount(Product.BANANA, threshold1, discount1);
        double expected = unitPrice * quantity * (1.0 - discount1);
        Cart cart = new Cart(List.of(new Item(Product.BANANA, quantity)));
        double actual = target.getCartPrice(cart);
        assertEquals(expected, actual, 0.001);
    }

    @Test
    void test_cr0_example7_moreApples_total15kg() {
        double unitPrice = 500.0;
        double quantity1 = 1.0;
        double quantity2 = 2.0;
        double quantity3 = 2.0;
        double totalQuantity = quantity1 + quantity2 + quantity3;
        Store target = new Store(Product.APPLE, unitPrice);
        double threshold1 = 5.0;
        double discount1 = 0.1;
```

```
        target.setDiscount(Product.APPLE, threshold1, discount1);
        Cart cart = new Cart(List.of(
                new Item(Product.APPLE, quantity1),
                new Item(Product.APPLE, quantity2),
                new Item(Product.APPLE, quantity3)
        ));
        double expected = unitPrice * totalQuantity * (1.0 - discount1);
        double actual = target.getCartPrice(cart);
        assertEquals(expected, actual, 0.001);
    }

    @Test
    void test_cr0_example8_2kgApple_2kgBanana() {
        double unitPriceApple = 500.0;
        double unitPriceBanana = 450.0;
        double discountThresholdBanana = 2.0;
        double bananaDiscount = 0.1;
        double quantityApple = 2.0;
        double quantityBanana = 2.0;
        Store target = new Store(List.of(
                new ProductPrice(Product.APPLE, unitPriceApple),
                new ProductPrice(Product.BANANA, unitPriceBanana)
        ));
        target.setDiscount(Product.BANANA, discountThresholdBanana,
bananaDiscount);
        double grossApple = unitPriceApple * quantityApple;
        double grossBanana = unitPriceBanana * quantityBanana;
        double netBanana = grossBanana * (1.0 - bananaDiscount);
        double expected = grossApple + netBanana;
        Cart cart = new Cart(List.of(
                new Item(Product.APPLE, quantityApple),
                new Item(Product.BANANA, quantityBanana)
        ));
        double actual = target.getCartPrice(cart);
        assertEquals(expected, actual, 0.001);
    }

    @Test
    void test_cr0_example9_1_01kgApple() {
        double unitPrice = 500.0;
        double quantity = 1.01;
        Store target = new Store(Product.APPLE, unitPrice);
        double expected = unitPrice * quantity;
        Cart cart = new Cart(List.of(new Item(Product.APPLE, quantity)));
        double actual = target.getCartPrice(cart);
        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;
        }
    }
```

```java
    @Test
    void test_cr0_example10_1_99kgBanana() {
        double unitPrice = 450.0;
        double quantity = 1.99;
        Store target = new Store(Product.BANANA, unitPrice);
        double gross = unitPrice * quantity;
        double expected = roundTo5(gross);
        Cart cart = new Cart(List.of(new Item(Product.BANANA, quantity)));
        double actual = target.getCartPrice(cart);
        assertEquals(expected, actual, 0.001);
    }

    @Test
    void test_cr0_example11_1_789kgApple() {
        double unitPrice = 500.0;
        double quantity = 1.789;
        Store target = new Store(Product.APPLE, unitPrice);
        double gross = unitPrice * quantity;
        double expected = roundTo5(gross);
        Cart cart = new Cart(List.of(new Item(Product.APPLE, quantity)));
        double actual = target.getCartPrice(cart);
        assertEquals(expected, actual, 0.001);
    }

    @Test
    void test_cr0_example12_2_01kgBanana_withDiscount() {
        double unitPrice = 450.0;
        double quantity = 2.01;
        double discountThreshold = 2.0;
        double discount = 0.1;
        Store target = new Store(Product.BANANA, unitPrice);
        target.setDiscount(Product.BANANA, discountThreshold, discount);
        double gross = unitPrice * quantity;
        double net = gross * (1.0 - discount);
        System.out.println(net);
        double expected = roundTo5(net);
        System.out.println(expected);
        Cart cart = new Cart(List.of(new Item(Product.BANANA, quantity)));
        double actual = target.getCartPrice(cart);
        assertEquals(expected, actual, 0.001);
    }

    @Test
    void test_cr0_example13_3_333kgApple_1_777kgBanana() {
        double unitPriceApple = 500.0;
        double unitPriceBanana = 450.0;
        double quantityApple = 3.333;
        double quantityBanana = 1.777;
        Store target = new Store(List.of(
                new ProductPrice(Product.APPLE, unitPriceApple),
                new ProductPrice(Product.BANANA, unitPriceBanana)
        ));
        double grossApple = unitPriceApple * quantityApple;
        double grossBanana = unitPriceBanana * quantityBanana;
        double gross = grossApple + grossBanana;
        double expected = roundTo5(gross);
        Cart cart = new Cart(List.of(
                new Item(Product.APPLE, quantityApple),
                new Item(Product.BANANA, quantityBanana)
        ));
        double actual = target.getCartPrice(cart);
```

```
        assertEquals(expected, actual, 0.001);
    }

    @Test
    void test_cr0_example14_5_001kgApple_withDiscount() {
        double unitPrice = 500.0;
        double quantity = 5.001;
        double discountThreshold = 5.0;
        double discount = 0.1;
        Store target = new Store(Product.APPLE, unitPrice);
        target.setDiscount(Product.APPLE, discountThreshold, discount);
        double gross = unitPrice * quantity;
        double net = gross * (1.0 - discount);
        double expected = roundTo5(net);
        Cart cart = new Cart(List.of(new Item(Product.APPLE, quantity)));
        double actual = target.getCartPrice(cart);
        assertEquals(expected, actual, 0.001);
    }

    @Test
    void test_cr0_example15_3_789kgApple_2_777kgBanana() {
        double unitPriceApple = 500.0;
        double unitPriceBanana = 450.0;
        double quantityApple = 3.789;
        double quantityBanana = 2.777;
        double discountThresholdBanana = 2.0;
        double bananaDiscount = 0.1;
        Store target = new Store(List.of(
                new ProductPrice(Product.APPLE, unitPriceApple),
                new ProductPrice(Product.BANANA, unitPriceBanana)
        ));
        target.setDiscount(Product.BANANA, discountThresholdBanana,
bananaDiscount);
        double grossApple = unitPriceApple * quantityApple;
        double grossBanana = unitPriceBanana * quantityBanana;
        double netBanana = grossBanana * (1.0 - bananaDiscount);
        double total = grossApple + netBanana;
        double expected = roundTo5(total);
        Cart cart = new Cart(List.of(
                new Item(Product.APPLE, quantityApple),
                new Item(Product.BANANA, quantityBanana)
        ));
        double actual = target.getCartPrice(cart);
        assertEquals(expected, actual, 0.001);
    }
}
```