Práctica 4

sábado, 25 de mayo de 2024 12:10

```
public interface Product{
    public BigDecimal getPrice();
    public void checkMinItemPrice(BigDecimal minItemPrice);
    long countItemsBelowMinPrice(BigDecimal minPrice,Set<Product> visitedProducts);
    default long countItemsBelowMinPrice(BigDecimal minPrice){
            Set<Product> visited=new HashSet<>();
            return countItemsBelowMinPrice(minPrice,visited);
    }
    void accept(ProductVisitor visitor);
}
public interface ProductVisitor{
    void visit(Item item);
    void visit(Item item);
}

public class Pack implements Product{
    private List<Product> products;

    public Pack(){
        this.products=new ArrayList<>();
}
public void addProduct(Product product){
    if(product==null){
      throw new NullPointerException("Product is null");
    }
    products.add();
}

@Override
public BigDecimal getPrice(){
    return products.stream()
           .map(Product::getPrice)
           .reduce(BigDecimal.ZERO,BigDecimal::add)
           .setScale(2,RoundingMode.HALF_UP);
}
public List<Product> getProducts(){
    return Collection.unmodifiableList(products);
}
@Override
public void checkMinItemPrice(BigDecimal minItemPrice){
    products.forEach(product->product.checkMinItemPrice);

}

@Override
public long countItemsBelowMinPrice(BigDecimal minPrice,Set<Product> visited){
    visited.add(this);
```

```
        visited.add(this);

        long count = 0;
        for (Product product : products)
            if (!visited.contains(product)) {
                count += product.countItemsBelowMinPrice(minPrice, visited);
            }
        }
        return count;
    }

    @Override
    public void accept(ProductVisitor visitor) {
        visitor.visit(this);
        for (Product product : products) {
            product.accept(visitor);
        }
    }
}
public class Item implements Product {
    private BigDecimal price;

    public Item(BigDecimal price) {
        checkGreaterThanZero(price);
        this.price = price.setScale(2, RoundingMode.HALF_UP);
    }
    private void checkGreaterThanZero(BigDecimal price) {
        if (price.compareTo(BigDecimal.ZERO) <= 0) {
            throw new IllegalArgumentException("Price must be greater than zero");
        }
    }

    @Override
    public BigDecimal getPrice() {
        return this.price;
    }

    public void setPrice(BigDecimal newPrice) {
        newPrice = newPrice.setScale(2, RoundingMode.HALF_UP);
        checkGreaterThanZero(newPrice);
        this.price = newPrice;
    }

    @Override
    public void checkMinItemPrice(BigDecimal minItemPrice) {
        checkGreaterThanZero(minItemPrice);
        if (this.price.compareTo(minItemPrice) < 0) {
            this.price = minItemPrice.setScale(2, RoundingMode.HALF_UP);
            }
        }

        @Override
        public long countItemsBelowMinPrice(BigDecimal minPrice, Set<Product> visitedProducts){
                return this.price.compareTo(minPrice)<0?1:0;

        }
        public void accept(ProductVisitor visitor){
            visitor.visit(this);

        }
    }
        public class MinPriceUpdaterVisitor implements ProductVisitor{
            private BigDecimal minPrice;

            private MinPriceUpdaterVisitor(BigDecimal minPrice){
                if(minPrice.compareTo(BigDecimal.ZERO)<=0){
                    throw new IllegalArgumentException("Minimum price must be greater than zero");

                }
            this.minPrice = minPrice.setScale(2,RoundingMode.HALF_UP);
        }
        @Override
        public void visit(Item item){
            if(item.getPrice().compareTo(minPrice)<0){
                item.setPrice(minPrice);

            }
        }


        @Override
        public void visit(Pack pack){
            for(Product product : pack.getProducts()){
                product.accept(this);
            }
        }
        public static void updatePrices(Product product, BigDecimal minPrice){
            MinPriceUpdaterVisitor visitor = new MinPriceUpdaterVisitor(minPrice);
            product.accept(visitor);

        }
    }
    public class CountItemsBelowMinPriceVisitor implements ProductVisitor{
        private BigDecimal minPrice;
        private int count;
        private Set<Product> visited;

        private CountItemsBelowMinPriceVisitor(BigDecimal minPrice){
            if(minPrice.compareTo(BigDecimal.ZERO)<=0){
                throw new IllegalArgumentException("Minimum price must be greater than zero")
            }
            this.minPrice = minPrice;
        this.minPrice = [illegible];
        this.count = 0;
        this.visited = new HashSet<>();
    }

    @Override
    public void visit(Item item) {
        if (!visited.contains(item)) {
            visited.add(item);
            if (item.getPrice().compareTo(minPrice) < 0) {
                count++;
            }
        }
    }


    @Override
    public void visit(Pack pack) {
        if (!visited.contains(pack)) {
            visited.add(pack);
            for (Product product : pack.getProducts()) {
                product.accept(this);
            }
        }
    }
    public int getCount() {
        return count;
    }
    public static int countItemsBelowMinPrice(Product product, BigDecimal minPrice) {
        CountItemsBelowMinPriceVisitor visitor = new CountItemsBelowMinPriceVisitor(minPrice);
        product.accept(visitor);
        return visitor.getCount();
    }
}
```