1 Commande d'articles

Une entreprise vend des articles (*items*) en ligne. Chaque article a un poids (*weight*) en kilogramme, un prix (*price*) et un nom (*name*). Une commande (*order*) d'articles contient une liste d'articles et permet de calculer le coût total de la commande incluant les frais de ports. Il existe deux types de frais de ports selon la méthode de livraison :

- rapide (*fast*) : frais de port égal à 4 fois le poids total de la commande avec un minimum de 20€ ;
- lent (*slow*) : frais de port offert pour les commandes d'au moins 100€ et égal à 2 fois le poids total de la commande sinon.

Le code des classes `Item` et `Order` est le suivant :

```
public class Item {
    private final int price;
    private final double weight;
    private final String name;
    public Item(int price, double weight, String name) {
        this.price = price;
        this.weight = weight;
        this.name = name;
    }
    public double getWeight() { return weight; }
    public int getPrice() { return price; }
    public String getName() { return name; }
}


public class Order {
    public enum ShippingMethodKind { FAST, SLOW }
    private ShippingMethodKind shippingMethodKind;
    private List<Item> items = new ArrayList<>();
    public Order(ShippingMethodKind shippingMethodKind) {
        this.shippingMethodKind = shippingMethodKind;
    }
    public double totalWeightOfItems(){
        return items.stream().mapToDouble(Item::getWeight).sum();
    }
    public int totalCostOfItems() {
        return items.stream().mapToInt(Item::getPrice).sum();
    }
    public int totalCost(){
        return totalCostOfItems() + getShippingCost();
```

```
    }
    private int shippingCost() {
        double totalWeight = totalWeightOfItems();
        return switch (shippingMethodKind) {
            case FAST -> (int) Math.max(totalWeight * 4, 20);
            case SLOW -> totalCostOfItems() >= 100 ? 0 : (int) totalWeight * 2;
        };
    }
    public void addItem(Item item) {
        items.add(item);
    }
    public void setShippingMethodKind(ShippingMethodKind shippingMethodKind) {
        this.shippingMethodKind = shippingMethodKind;
    }
    public void print() {
        items.forEach(item -> System.out.println(item.getName() + " " + item.getPrice() +"€"));
        System.out.println("Shipping" + shippingMethodKind.name() + " " + shippingCost()+"€");
        System.out.println("total cost of " + totalCost()+"€");
    }
}
```

Question 1 : Qu'affiche l'exécution de la méthode `main` du code suivant ?

```
public class Main {
    public static void main(String[] args) {
        Item phone = new Item(1000, 0.5,"Smartphone");
        Item television = new Item(1000, 15,"Television");
        Order order = new Order(Order.ShippingMethodKind.SLOW);
        order.addItem(phone);
        order.addItem(television);
        order.print();
    }
}
```

Question 2 : Que doit-on modifier dans le code de la classe `Order` si on souhaite ajouter un nouveau type de frais de port, par exemple des frais de port, correspondant à une livraison de rapidité standard, qui serait égaux à 3 fois le poids total de la commande ? Quel(s) principe(s) SOLID ne sont pas respectés ?

On souhaite remplacer l'`enum ShippingMethodKind` par une interface `ShippingMethod` qui sera implémentée par deux classes différentes `SlowShippingMethod` et `FastShippingMethod`. L'objectif est que le code suivant ait le même comportement que le code de la question précédente.

```
public class Main {
    public static void main(String[] args) {
        Item phone = new Item(1000, 0.5,"Smartphone");
        Item television = new Item(1000, 15,"Television");
        ShippingMethod shippingMethod = new SlowShippingMethod()
        Order order = new Order(shippingMethod);
        order.addItem(phone);
        order.addItem(television);
        order.print();
    }
}
```

Question 3 : Donnez le diagramme de classes d'une nouvelle organisation du code (interface `ShippingMethod` qui sera implémentée par deux classes différentes `SlowShippingMethod` et `FastShippingMethod`) qui corrige le défaut de conception évoqué à la question 2.

Question 4 : Donnez le code de la méthode `shippingCost()` de la classe `Order` avec la nouvelle organisation du code.

Question 5 : Donnez le code de la classe `SlowShippingMethod`.

On souhaite rajouter la possibilité de faire des packs d'articles qui regroupe plusieurs articles avec une réduction de prix (rabais exprimé en pourcentage). Par exemple, on souhaitera avoir la possibilité de créer un pack comprenant une télévision et un smartphone avec un rabais (*discount*) de 20% avec le code suivant.

```
public class Main {
    public static void main(String[] args) {
        Item phone = new SingleItem(1000, 0.5,"Smartphone");
        Item television = new SingleItem(1000, 15,"Television");
        Item pack = new CompositeItem(List.of(phone, television), 20);
        ShippingMethod shippingMethod = new FastShippingMethod();
        Order order = new Order(shippingMethod);
        order.addItem(pack);
        order.print();
    }
}
```

L'exécution du code ci-dessus devra donner l'affichage suivant :

```
Pack -20% (Smartphone,Television) 1600€
Shipping FAST 62€
total cost of 1662€
```

Question 6 : Donnez le diagramme de classes d'une nouvelle organisation du code qui permettra l'exécution du code ci-dessus.

Question 7 : Donnez le code de la classe `CompositeItem`.