Correction Examen 2020

Question 1

Le principe solide violé est OCP. En effet la classe est sensée gérer un ensemble de cadeaux, l'ajout de cadeaux supplémentaires demandera de rajouter des attributs supplémentaire et de modifier les méthodes display et totalValue.

Question 2

Nous allons utiliser un interface gift qui permettra de séparer la fonctionnalité du container de celle des cadeaux.

Question 3

```
public interface Gift {
    double value();
    String text();
}
```

```java
public class SantaBackpack {
    List<Gift> gifts = new ArrayList<>();
    void display(){
        for(Gift gift : gifts)
            System.out.println(gift.text());
    }
    double totalValue(){
        double sum = 0;
        for(Gift gift : gifts)
            sum+= gift.value();
        return sum;
    }
}
```

```java
public class Chocolate implements Gift{
    double weight,value;
    @Override
    public double value() {
        return value;
    }

    @Override
    public String text() {
        return weight+"g of chocolate";
    }

    public void eat(double weight){this.weight-= weight;}
}
```

```java
public class FireTruck implements Gift{
    double value;

    @Override
    public double value() {
        return value;
    }

    @Override
    public String text() {
        return "One Firetruck";
    }

    public void play(){
        System.out.println("PinPonPinPon");
    }
}
```

Question 4

Il nous est demandé d'introduire des nouvelles évaluations des cadeaux qui dépendent d'Anne et de Bob. Le patron naturel à utiliser est

Question 5

```
public interface GiftVisitor<R> {
    public R visit(Chocolate chocolate);
    public R visit(FireTruck fireTruck);
}
```

```java
public class AnnValuation implements GiftVisitor<Double>{

    Double evaluate(List<Gift> gifts){
        double sum = 0;
        for(Gift gift : gifts)
            sum+= gift.accept(this);
        return sum;
    }

    @Override
    public Double visit(Chocolate chocolate) {
        return chocolate.value()/2;
    }

    @Override
    public Double visit(FireTruck fireTruck) {
        return fireTruck.value()*2;
    }
}
```

```java
public interface Gift {
    public Double value();
    public String text();
    <R> R accept(GiftVisitor<R> giftVisitor);
}

@Override
    public <R> R accept(GiftVisitor<R> giftVisitor) {
        return giftVisitor.visit(this);
    }
```

```java
public class CamilleEvaluation implements GiftVisitor<Void>
    double totalChocolateValue,totalFiretruckValue;

    public Double evaluate(List<Gift> gifts){
        for(Gift gift : gifts)
            gift.accept(this);
        return Math.min(totalChocolateValue,
        totalFiretruckValue);}
    @Override
    public Void visit(Chocolate chocolate) {
        totalChocolateValue+=chocolate.value();
        return null;}
    @Override
    public Void visit(FireTruck fireTruck) {
        totalFiretruckValue+=fireTruck.value();
        return null;}
}
```