Génie Logiciel : Conception
Kata "Harry Potter"

Sébastien Mosser
INF-5153, Hiver 2019, Cours #1.2

UQÀM

Vendre les livres Harry Potter

Plus sorcier que ça n'en à l'air !

Spécifications

- On considère les 5 premiers livres uniquement. Chaque livre coûte 8$
- Politiques de rabais en fonction du panier du consommateur :
  - 2 livres différents : 5%
  - 3 livres différents: 10%
  - 4 livres différents : 20%
  - La série complète : 25%
- Le rabais ne s'applique que sur les livres concernées par la politique

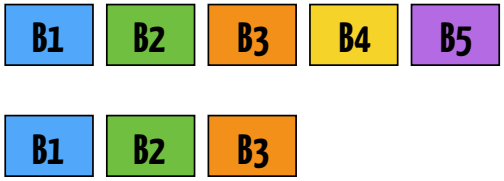
http://www.codingdojo.org/cgi-bin/index.pl?action=browse&id=KataPotter&revision=41 [3]

Exemple de calcul de prix

= ?

Exemple de calcul de prix

= ?

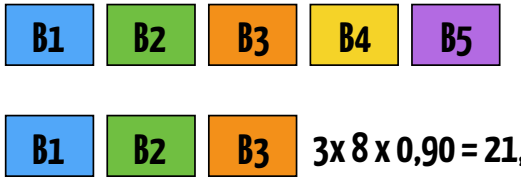


4-2

Exemple de calcul de prix

= ?

5x 8 x 0,75 = 30

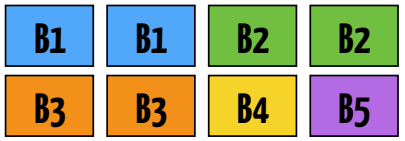


3x 8 x 0,90 = 21,6

} 51,60

4-3

Piège !



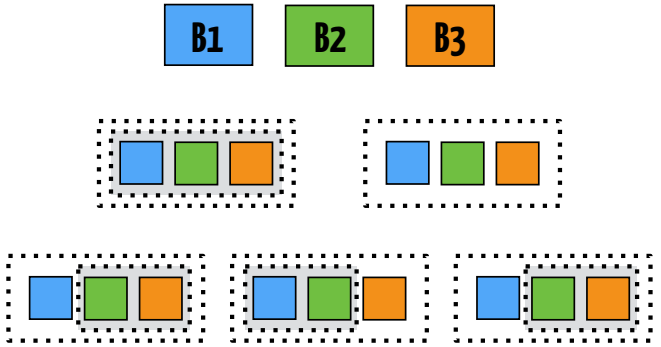
= ~~51,60~~

4 x 8 x 0,8 = 25,6

} = 2 x 25,6 = 51,20

5

Situation classiquement combinatoire



Ordre de grandeur : Nombre de Bell

6

Bell(7) = ??

7-1

Bell(7) =

877

7-2

```
public class PotterDiscountCalculator {

    private double[] discountRates;
    private int[] discounts;

    private void init() {
        discountRates = new double[]{ 1, 0.95, 0.90, 0.80, 0.75 };
        discounts = new int[5];
    }

    public double calculatePrice(List<Integer> order) {
        init(); // resetting the calculator before computing a price
        if (order == null || order.isEmpty())
            return 0.0;
        int[] booksInOrder = calculateBooksInOrder(order);
        calculateDiscounts(booksInOrder);
        optimizeDiscounts();
        double price = 0.0;
        for (int i = 0; i < discounts.length; i++)
            price += (8 * (i + 1) * discounts[i]) * discountRates[i];
        return price;
    }

    protected void optimizeDiscounts() {
        while (discounts[2] > 0 && discounts[4] > 0) {
            discounts[2]--;
            discounts[4]--;
            discounts[3] += 2;
        }
    }

    protected void calculateDiscounts(int[] booksInOrder) {
        if (booksInOrder == null)
            return;
        int differentFromZero = 0;
        for (int i = 0; i < booksInOrder.length; i++) {
            if (booksInOrder[i] > 0) {
                differentFromZero++;
                booksInOrder[i]--;
            }
        }
        if (differentFromZero > 0) {
            discounts[differentFromZero - 1] += 1;
            calculateDiscounts(booksInOrder);
        }
    }

    private int[] calculateBooksInOrder(List<Integer> order) {
        int[] result = new int[5];
        for (int book : order)
            result[book - 1]++;
        return result;
    }
}
```

Yes we can !

8

```
private double[] discountRates;
private int[] discounts;

private void init() {
    discountRates = new double[]{ 1, 0.95, 0.90, 0.80, 0.75 };
    discounts = new int[5];
}

public double calculatePrice(List<Integer> order) {
    init(); // resetting the calculator before computing a price
    if (order == null || order.isEmpty())
        return 0.0;
    int[] booksInOrder = calculateBooksInOrder(order);
    calculateDiscounts(booksInOrder);
    optimizeDiscounts();
    double price = 0.0;
    for (int i = 0; i < discounts.length; i++)
        price += (8 * (i + 1) * discounts[i]) * discountRates[i];
    return price;
}
```

http://bfindeiss.blogspot.fr/2013/07/the-katapotter-in-java-solved-in-31.html

9

```java
protected void calculateDiscounts(int[] booksInOrder) {
    if (booksInOrder == null)
        return;
    int differentFromZero = 0;
    for (int i = 0; i < booksInOrder.length; i++) {
        if (booksInOrder[i] > 0) {
            differentFromZero++;
            booksInOrder[i]--;
        }
    }
    if (differentFromZero > 0) {
        discounts[differentFromZero - 1] += 1;
        calculateDiscounts(booksInOrder);
    }
}
```

http://bfindeiss.blogspot.fr/2013/07/the-katapotter-in-java-solved-in-31.html

10

```java
private int[] calculateBooksInOrder(List<Integer> order) {
    int[] result = new int[5];
    for (int book : order)
        result[book - 1]++;
    return result;
}


protected void optimizeDiscounts() {
    while (discounts[2] > 0 && discounts[4] > 0) {
        discounts[2]--;
        discounts[4]--;
        discounts[3] += 2;
    }
}
```

http://bfindeiss.blogspot.fr/2013/07/the-katapotter-in-java-solved-in-31.html

11

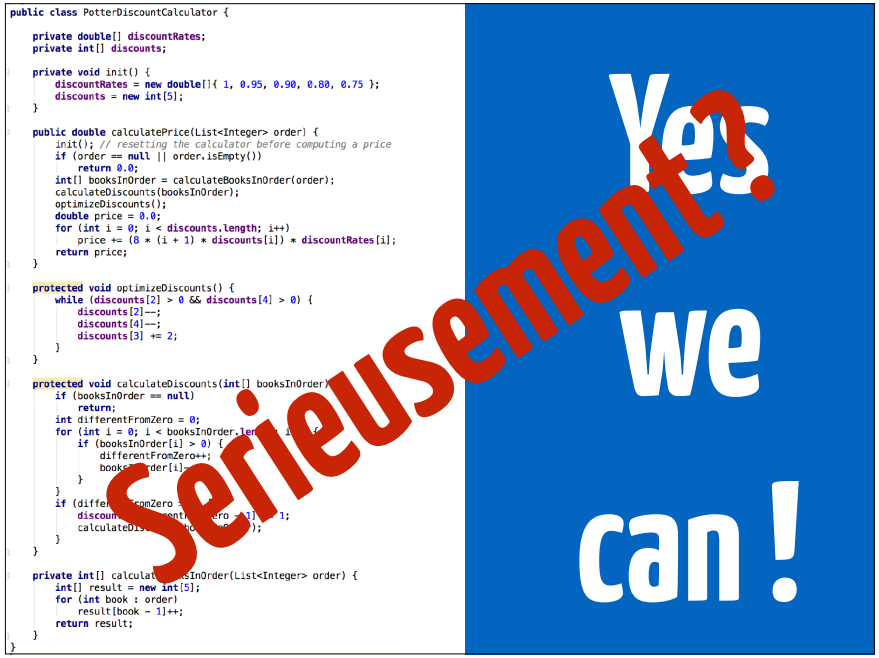
```java
@Test
public void edgeCase() {
    assertEquals( 51.20,
            calculator.calculatePrice(build(1, 1, 2, 2, 3, 3, 4, 5)), 0.01);
    assertEquals(141.20, calculator.calculatePrice(
            build(  1, 1, 1, 1, 1,
                    2, 2, 2, 2, 2,
                    3, 3, 3, 3,
                    4, 4, 4, 4,
                    5, 5, 5, 5     )), 0.01);
}


@Test
public void simpleDiscounts() {
    assertEquals(15.20, calculator.calculatePrice(build(1, 2)),          0.01);
    assertEquals(21.60, calculator.calculatePrice(build(1, 3, 5)),       0.01);
    assertEquals(25.60, calculator.calculatePrice(build(1, 2, 3, 5)),    0.01);
    assertEquals(30.00, calculator.calculatePrice(build(1, 2, 3, 4, 5)), 0.01);
}
```

12

```java
public class PotterDiscountCalculator {

    private double[] discountRates;
    private int[] discounts;

    private void init() {
        discountRates = new double[]{ 1, 0.95, 0.90, 0.80, 0.75 };
        discounts = new int[5];
    }

    public double calculatePrice(List<Integer> order) {
        init(); // resetting the calculator before computing a price
        if (order == null || order.isEmpty())
            return 0.0;
        int[] booksInOrder = calculateBooksInOrder(order);
        calculateDiscounts(booksInOrder);
        optimizeDiscounts();
        double price = 0.0;
        for (int i = 0; i < discounts.length; i++)
            price += (8 * (i + 1) * discounts[i]) * discountRates[i];
        return price;
    }

    protected void optimizeDiscounts() {
        while (discounts[2] > 0 && discounts[4] > 0) {
            discounts[2]--;
            discounts[4]--;
            discounts[3] += 2;
        }
    }

    protected void calculateDiscounts(int[] booksInOrder) {
        if (booksInOrder == null)
            return;
        int differentFromZero = 0;
        for (int i = 0; i < booksInOrder.length; i++) {
            if (booksInOrder[i] > 0) {
                differentFromZero++;
                booksInOrder[i]--;
            }
        }
        if (differentFromZero > 0) {
            discounts[differentFromZero - 1] += 1;
            calculateDiscounts(booksInOrder);
        }
    }

    private int[] calculateBooksInOrder(List<Integer> order) {
        int[] result = new int[5];
        for (int book : order)
            result[book - 1]++;
        return result;
    }
}
```

Yes we can !

13-1



13-2

Problèmes ?

14

slack

“ Bon, j'ai réfléchi un peu à ce problème et algorithmiquement, **il y a trois options** pour le résoudre : une **naïve** (ton exemple de code java) qui est exponentielle en le nombre de bouquin à acheter, une **dynamique** qui est linéaire en le nombre de bouquins à acheter (quadratique si le nombre des offres de prix est infini) et enfin une qui est **logarithmique** (amorti) en le nombre de bouquins (mais seulement avec un nombre d'offres de prix finis) et **qui sera en pratique en temps constant** (à moins d'avoir un nombre de bouquin à acheter qui ne tient pas sur un int32, mais il y a peu de chances quand même).

- Christophe Papazian

15

slack

```
from itertools import product

Bundles = list(product((0,1),repeat=5))[1:]

tags = [0,800,2*8*95,3*8*90,4*8*80,5*8*75]

def bundles(t) :
    for k in Bundles :
        if all(t[i]>=k[i] for i in range(5)) :
            yield tags[sum(k)], tuple(t[i]-k[i] for i in range(5))

def best_price(t,cache={(0,0,0,0,0):0}) :
    t=tuple(sorted(t))
    try : return cache[t]
    except KeyError :
        cache[t]=res=min(a[0]+best_price(a[1]) for a in bundles(t))
        return res

def price(l) : return best_price(tuple(l.count(i) for i in range(5)))/100
```

“Mais **je trouve ça plutôt bof**, ça passe les tests, et c'est polynomial en le nombre de bouquins en entrée, mais c'est un polynome de degré 5, donc les performances sont mauvaises en pratique. **Faut que je refléchisse pour la solution logarithmique**... mais pas ce soir” - Christophe

16

Guide de survie en Complexité

if (christophe.says("Je dois réfléchir"))
return "Fly you fools!"

=> Le problème n'est pas trivial

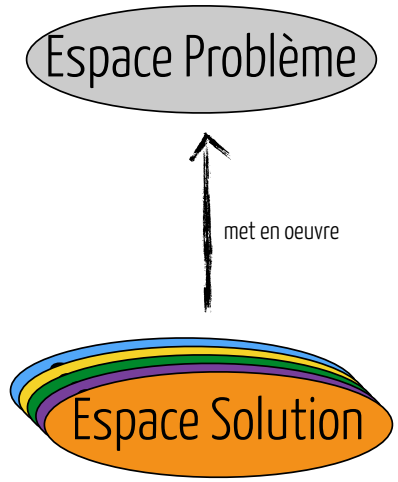
Version améliorée

slack
80 books: 6s -> 20ms

```
tags = [0,800,2*8*95,3*8*90,4*8*80,5*8*75]

def best_price(t) :
    t=tuple(sorted(t))
    m=min(t[0],t[2]-t[1])
    return tags[5]*(t[0]-m)+tags[4]*(2*m+t[1]-t[0])+
           tags[3]*(t[2]-t[1]-m)+tags[2]*(t[3]-t[2])+
           tags[1]*(t[4]-t[3])

def price(l) :
   return best_price(tuple(l.count(i) for i in range(5)))/100
```

"ok, **j'ai une version logarithmique** qui passe les tests ;P. Version python avec test, c'est linéaire par rapport au log du nombre de bouquins achetés." - Christophe

Comment encapsuler la complexité du calcul de manière à rendre le code source compréhensible et maintenable ?



Règle de 3 de la conception logicielle

21-1

Règle de 3 de la conception logicielle

Lisibilité

21-2

Règle de 3 de la conception logicielle

Lisibilité
Lisibilité

21-3

Règle de 3 de la conception logicielle

Lisibilité
Lisibilité
Lisibilité

21-4

Principe de SOLIDité en conception objet

- S : Single Responsibility
- O: Open / closed principle
- L: Liskov-compliant (substitution)
- I: Interface Segregation
- D: Dependency inversion

Principe de SOLIDité en conception objet

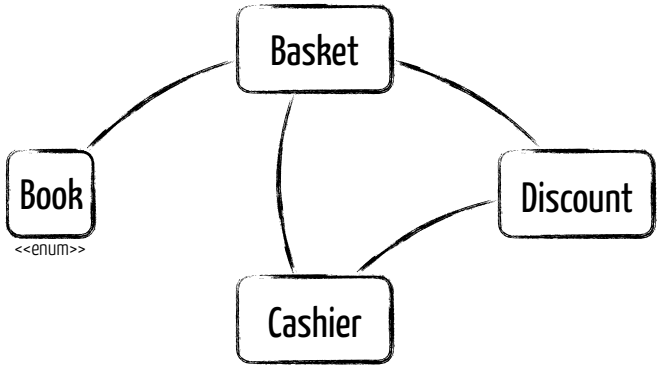
- S : Single Responsibility — **Ici**
- O: Open / closed principle — **Après**
- L: Liskov-compliant (substitution)
- I: Interface Segregation — **Encore Après**
- D: Dependency inversion

D'un code à un code "conçu"

Et si on causait objets ?

Modularisation

Modularisation



Calcul du Prix pour **b** ∈ Basket

b is empty => **0.0**

Let **discounts** = {**d** ∈ Discount | **d** can be applied to **b**}

discounts is empty => $|\mathbf{books}|_{\mathbf{b}} * \mathbf{PRICE}$

Let **candidates** = { **p** ∈ ℝ | **d** ∈ **discounts**, **p** = compute(**b**,**d**) }

=> **min(candidates)**

NAIVE

Calcul du prix avec un rabais **d** pour un panier **b**

Let **local price** = **d** applied to **b**

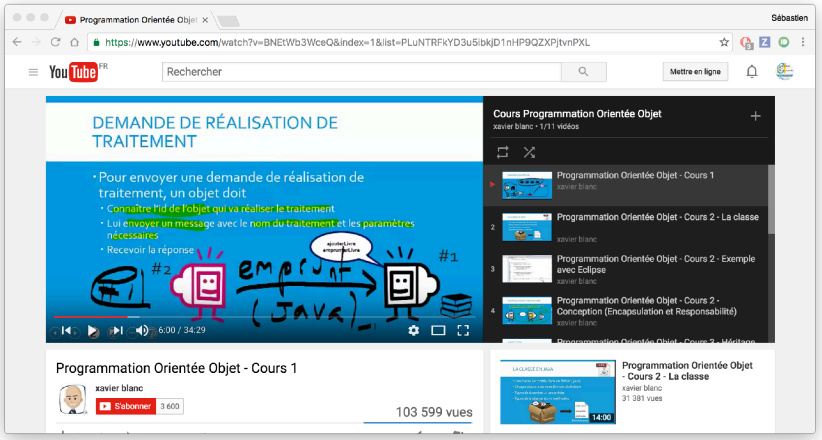
Let **remaining** = clone of **b** without the books used in **d**

=> **local price** + compute(**remaining**)

NAIVE

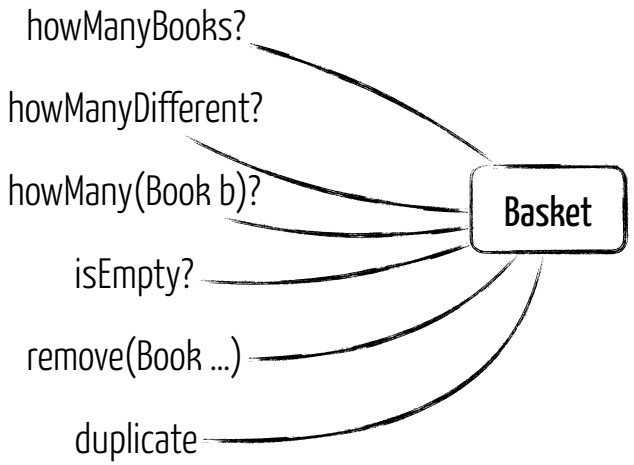
WAT ? Pas d'accesseurs ? de modificateurs ?

Est-ce que c'est seulement du Java ?



https://www.youtube.com/playlist?list=PLuNTRFkYD3u5ibkjD1nHP9QZXPjtvnPXL

Description **Comportementale**



Implémentation Interne

```
public class Basket {

    private Map<Book, Integer> contents = new EnumMap<>(Book.class);

    public Basket() {
        for(Book b: Book.values())
            contents.put(b,0);
    }

    public Basket(Book... chosen) {
        this();
        for(Book book: chosen) {
            contents.put(book, contents.get(book) + 1);
        }
    }
```


Implémentation Interne

```
public class Basket {

    private Map<Book, Integer> conte
    public Basket() {
        for(Book b: Book.
            contents.put(b,0)
    }

    public Ba
        this(
        for(Boo
            cont
    }
```

On s'en fiche !

> One of the **great leaps in OO** is to be able to answer the question "**how does this work?**" with "**I don't care**".
>
> - Alan Knight

Les tests portent sur l'interface publique

```
private static final Basket empty = new Basket();
private static final Basket complete = new Basket(BOOK1, BOOK2, BOOK3, BOOK4, BOOK5);
private static final Basket azkaban = new Basket(BOOK3, BOOK3, BOOK3);

@Test
public void basketContents() {
    for(Book b: Book.values())
        assertEquals(0, empty.howMany(b));

    for(Book b: Book.values())
        assertEquals(1, complete.howMany(b));

    for(Book b: Book.values())
        if(b == BOOK3) {
            assertEquals(3, azkaban.howMany(b));
        } else {
            assertEquals(0, azkaban.howMany(b));
        }
}
```


On tests les cas les plus critiques

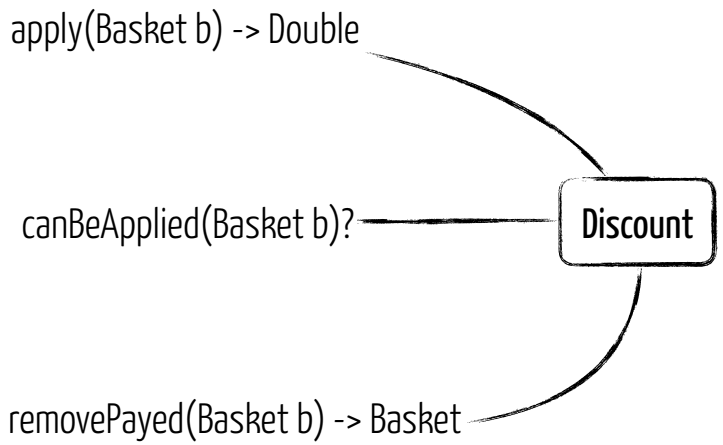
~

```
@Test
public void removeBooks() {
    Basket basket = new Basket(BOOK1, BOOK2, BOOK2);
    assertEquals(3, basket.howManyBooks());
    basket.remove(BOOK1);
    assertEquals(2, basket.howManyBooks());
    basket.remove(BOOK2);
    assertEquals(1, basket.howManyBooks());
}
```

✔

```
@Test
public void cloneBooks() {
    Basket basket = new Basket(BOOK1, BOOK2, BOOK2);
    Basket clone = basket.duplicate();
    clone.remove(BOOK2);
    assertEquals(2, basket.howMany(BOOK2));
    assertEquals(1, clone.howMany(BOOK2));
}
```


Exemple : Le cas des **Rabais**



```
public class  Discount {
    [redacted]
    public Discount(int nbBooks, double percentage) {[redacted]}
    public boolean canBeApplied(Basket b) { [redacted] }
    public double apply(Basket b) { [redacted] }
    public Basket removePayedBooks(Basket b) {
        [redacted]
    }
    private Map<Book, Integer> contents(Basket b) {
        [redacted]
    }
}
```

35-1

```
public class  Discount {

    private int nbBooks;
    private double percentage;

    public Discount(int nbBooks, double percentage) {...}

    public boolean canBeApplied(Basket b) { return b.howManyDifferent() >= nbBooks; }

    public double apply(Basket b) { return Cashier.PRICE * nbBooks * percentage; }

    public Basket removePayedBooks(Basket b) {
        Map<Book, Integer> data = contents(b);
        Book[] consumed =
                data.entrySet().stream()
                        .sorted((e1, e2) -> Integer.compare(e2.getValue(), e1.getValue()))
                        .map(Map.Entry::getKey)
                        .collect(Collectors.toList()).subList(0, nbBooks).toArray(new Book[]{});
        Basket result = b.duplicate();
        result.remove(consumed);
        return result;
    }

    private Map<Book, Integer> contents(Basket b) {
        Map<Book, Integer> data = new HashMap<>();
        for(Book book: Book.values())
            data.put(book, b.howMany(book));
        return data;
    }
}
```

35-2

```
public class  Discount {
    [redacted]
    public Discount(int nbBooks, double percentage) {[redacted]}
    public boolean canBeApplied(Basket b) { [redacted] }
    public double apply(Basket b) { [redacted] }
    public Basket removePayedBooks(Basket b) {
        [redacted]
    }
    private Map<Book, Integer> contents(Basket b) {
        [redacted]
    }
}
```

On s'en fiche !

36

Let **local price** = **d** applied to **b**

Let **remaining** = clone of **b** without the books used in **d**

=> **local price** + compute(**remaining**)

```
private double compute(Basket b, Discount d) {
    double local = d.apply(b);
    Basket remaining = d.removePayedBooks(b);
    return local + compute(remaining);
}
```

37

b is empty => **0.0**

Let **discounts** = {**d** ∈ Discount | **d** can be applied to **b**}

discounts is empty => **|books|$_b$ * PRICE**

Let **candidates** = { **p** ∈ ℝ | **d** ∈ **discounts**, **p** = compute(**b**,**d**) }

=> **min(candidates)**

```
private double compute(Basket b) {
    if(b.isEmpty()) { return 0.0; }

    List<Discount> availables = findAvailableDiscount(b);
    if(availables.isEmpty()) {
        return PRICE * b.howManyBooks();
    } else {
        return availables.stream().map(d -> compute(b, d)).min(Double::compare).get();
    }
}
```



```
private double compute(Basket b) {
    if(b.isEmpty()) { return 0.0; }

    List<Discount> availables = findAvailableDiscount(b);
    if(availables.isEmpty()) {
        return PRICE * b.howManyBooks();
    } else {
        return availables.stream().map(d -> compute(b, d)).min(Double::compare).get();
    }
}
```

b is empty => **0.0**

Let **discounts** = {**d** ∈ Discount | **d** can be applied to **b**}

discounts is empty => **|books|$_b$ * PRICE**

Let **candidates** = { **p** ∈ ℝ | **d** ∈ **discounts**, **p** = compute(**b**,**d**) }

=> **min(candidates)**

Let **local price** = **d** applied to **b**

Let **remaining** = clone of **b** without the books used in **d**

=> **local price** + compute(**remaining**)

```
private double compute(Basket b, Discount d) {
    double local = d.apply(b);
    Basket remaining = d.removePayedBooks(b);
    return local + compute(remaining);
}
```


Comparaison de performances

```
# Run complete. Total time: 00:40:41

Benchmark                    Mode  Cnt   Score    Error  Units
PotterBenchmark.ooBasics     avgt  200   0.008 ±  0.001  ms/op
PotterBenchmark.ooDiscount   avgt  200   0.012 ±  0.001  ms/op
PotterBenchmark.ooEdge       avgt  200  29.504 ±  0.487  ms/op
PotterBenchmark.rawBasics    avgt  200  ≈ 10^-4          ms/op
PotterBenchmark.rawDiscount  avgt  200  ≈ 10^-4          ms/op
PotterBenchmark.rawEdge      avgt  200  ≈ 10^-4          ms/op
```