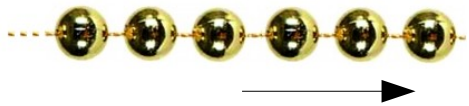
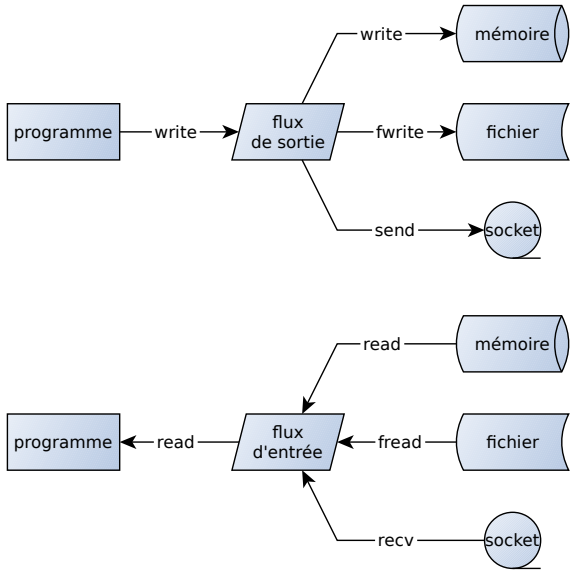
Rappels et compléments

Flux : objet transportant des données séquentiellement et en quantité indéterminée...



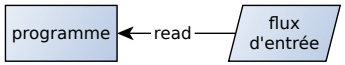
... qui permet une communication standardisée entre le programme et son environnement.



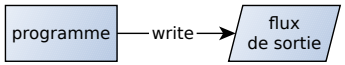
Classification des flux selon 3 axes :
1/ selon le mode d'accès
2/ selon le type des données transportées
3/ selon le mode opératoire

Axe 1 "mode d'accès"

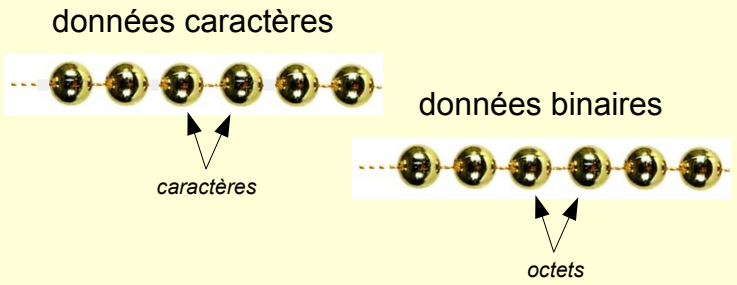
accès en lecture

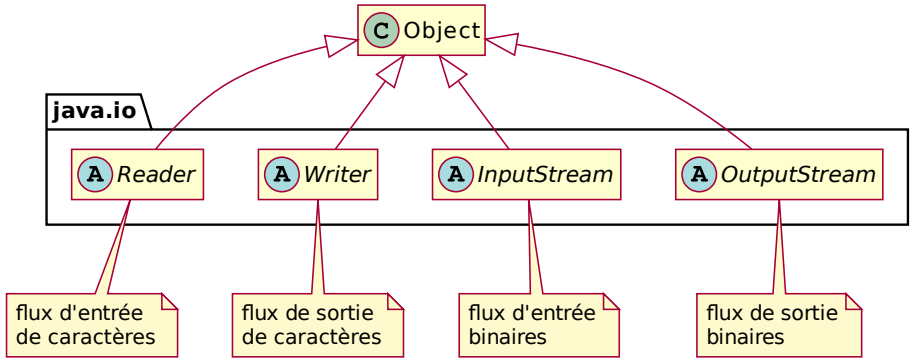


accès en écriture



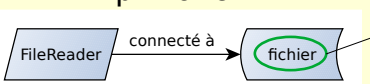
Axe 2 "type de données"





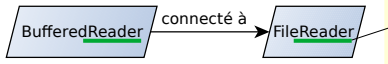
Axe 3 "mode opératoire"

primaire



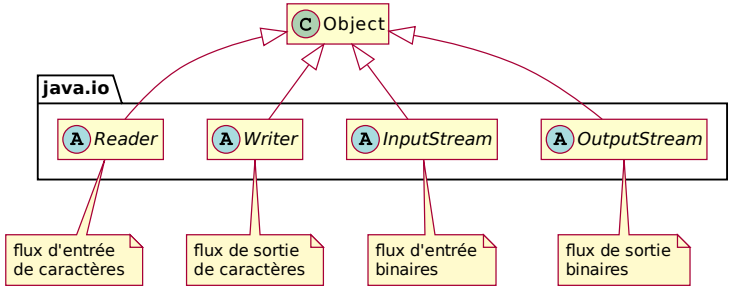
les constructeurs prennent une ressource système en argument

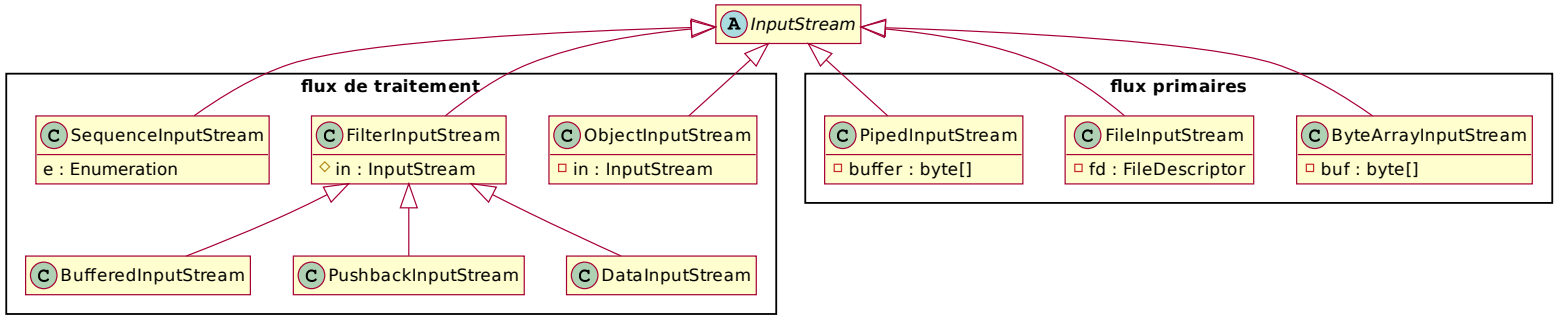
traitement

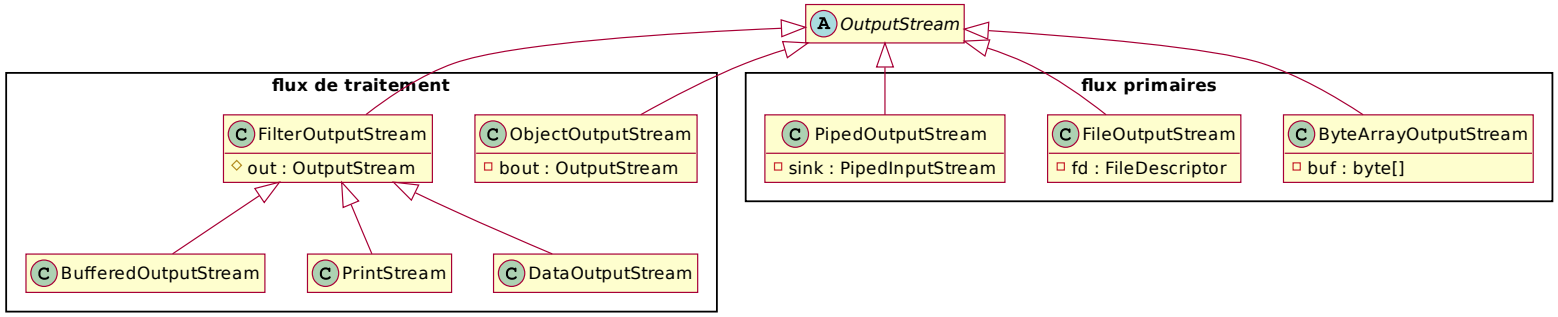


les constructeurs prennent un flux de même mode d'accès en argument

Rappels et compléments

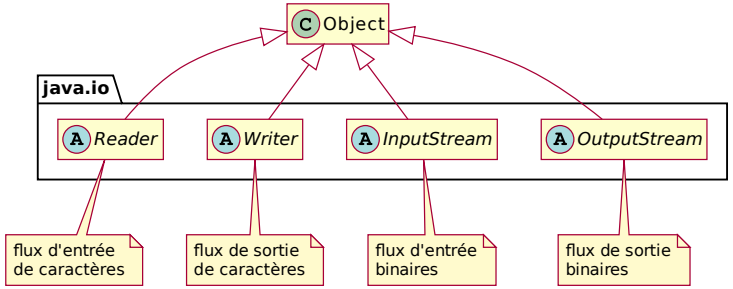


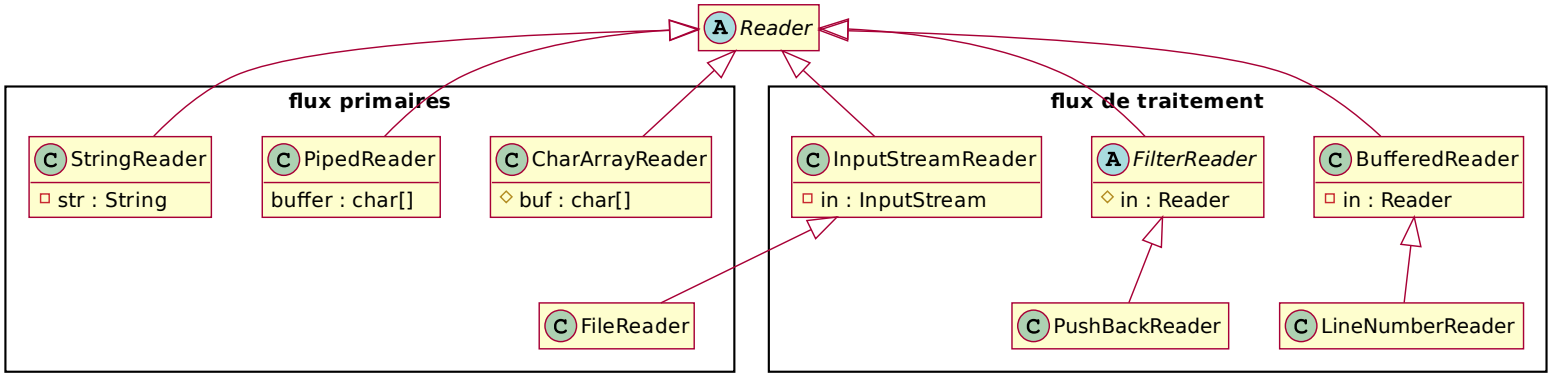


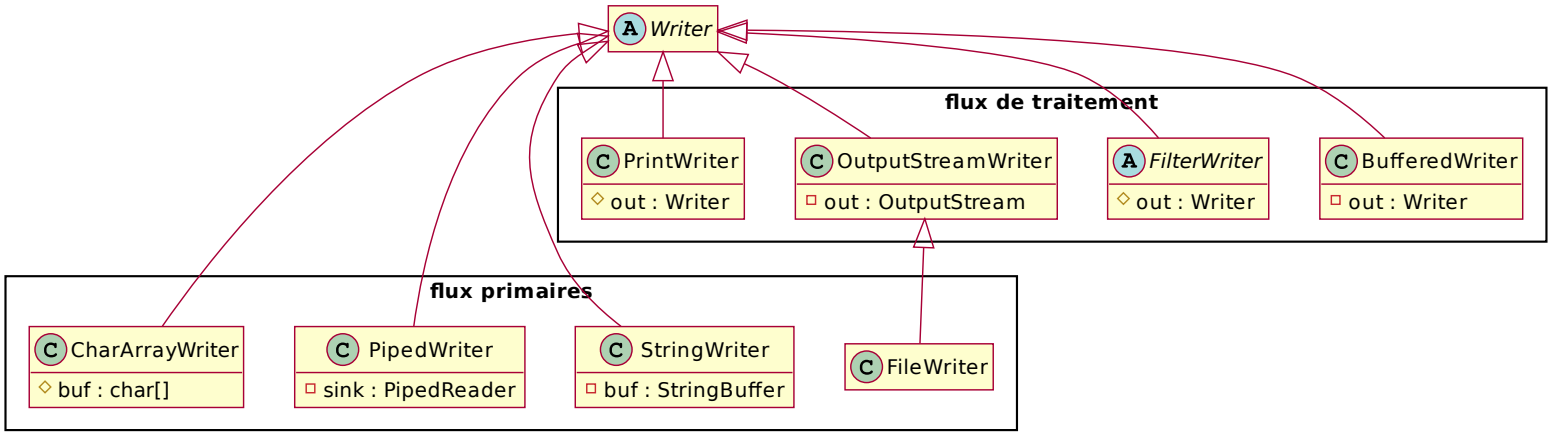


à travailler en autonomie
(lire poly + doc Java)

Rappels et compléments

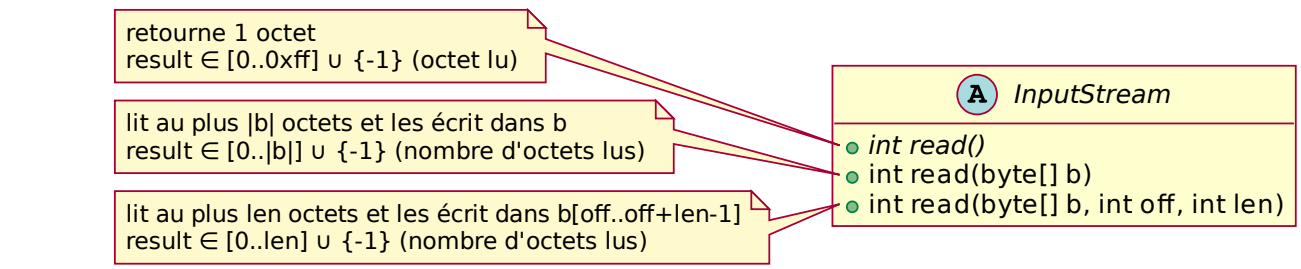


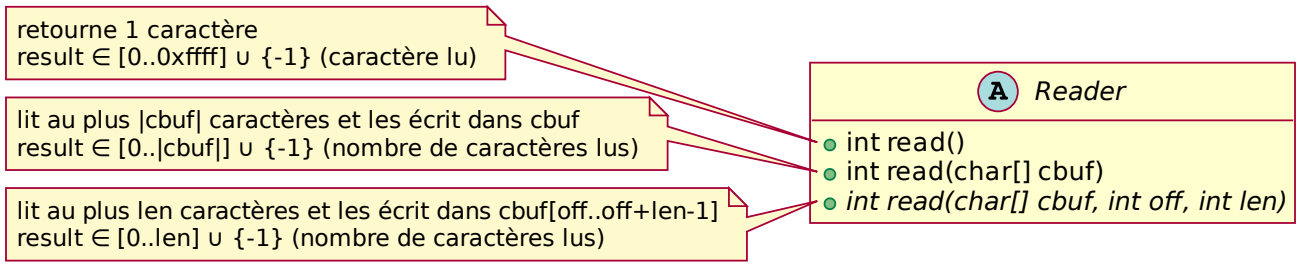


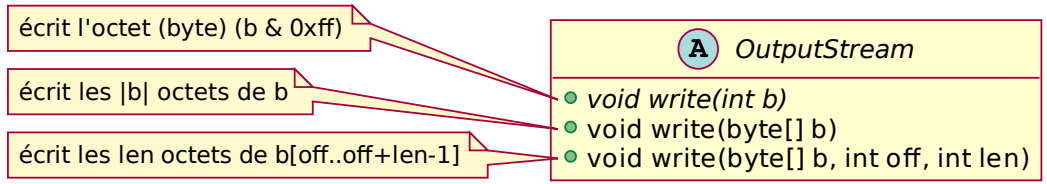


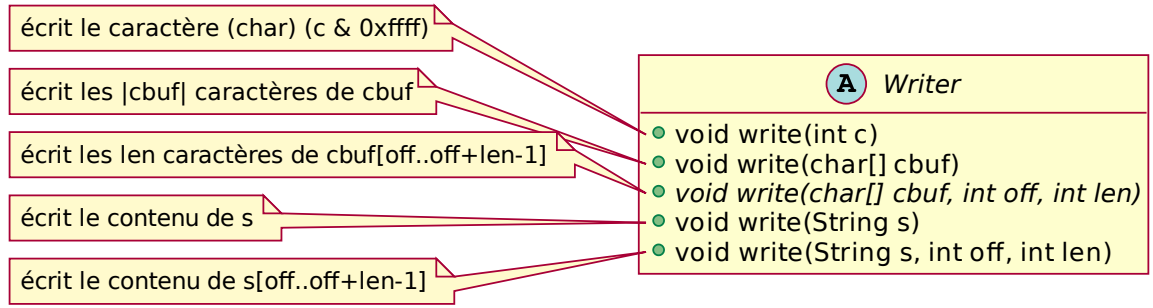
à travailler en autonomie
(lire poly + doc Java)

Rappels et compléments









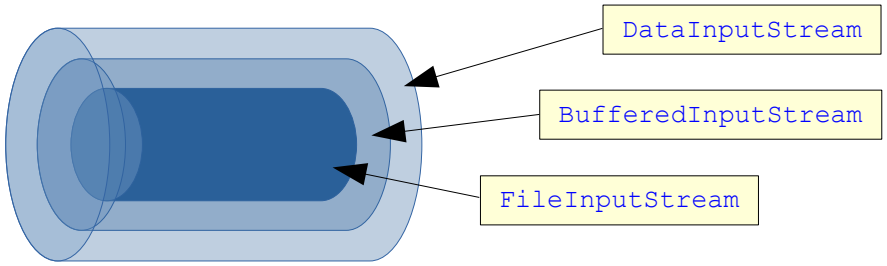
à travailler en autonomie
(lire poly + doc Java)

Rappels et compléments


```
DataInputStream dis =
  new DataInputStream(
    new BufferedInputStream(
      new FileInputStream("data.bin")));
```

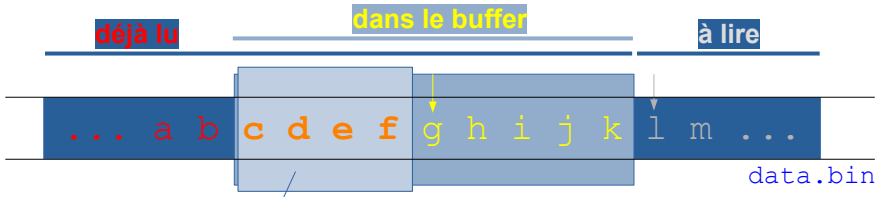
throws FNFException

création par emboîtement
avec ouverture automatique




```
int x = dis.readInt();
```

lecture de valeurs primitives
par lecture bufferisée
dans un fichier binaire ouvert en lecture



readInt

throws EOFException

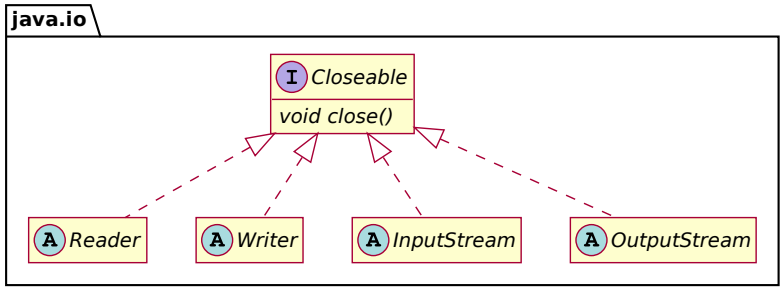
throws IOException

```
(c << 24) + (d << 16) + (e << 8) + f
```

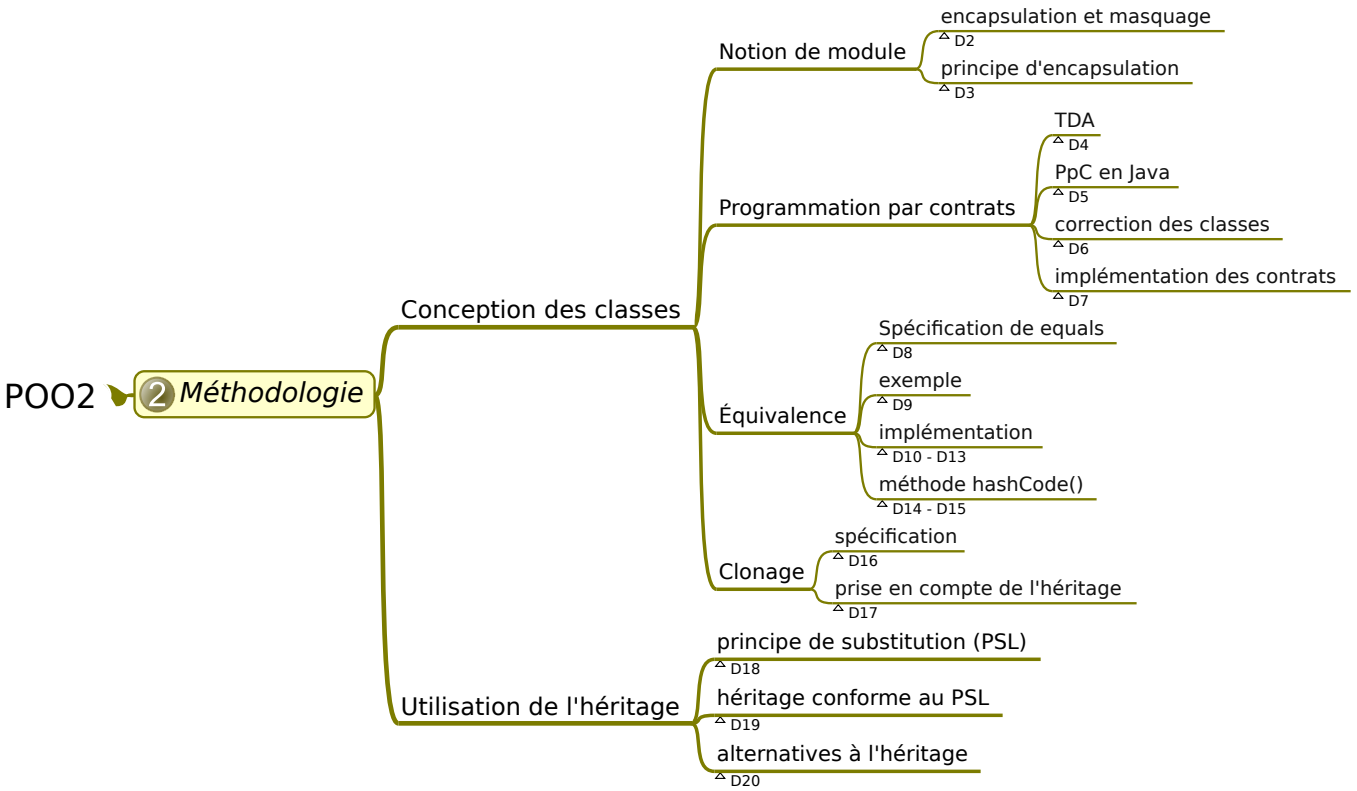
```
dis.close();
```

fermeture explicite
avec close()

throws IOException



Méthodologie



Méthodologie

Module : élément logiciel qui met en œuvre l'encapsulation des données et la rétention d'information.

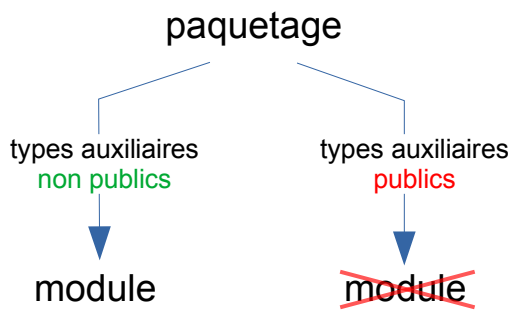
Encapsulation des données :
Lorsqu'une unité logicielle regroupe en même temps des opérations et toutes les structures de données sur lesquelles agissent ces opérations.

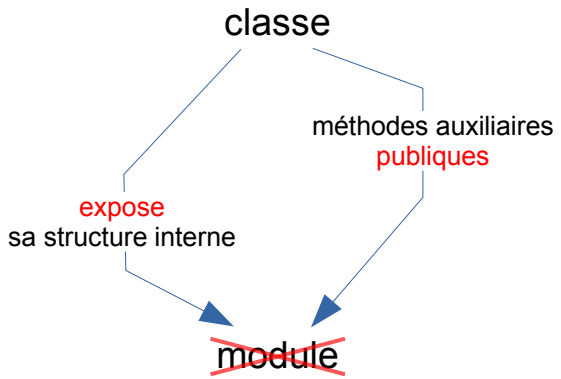
```
public class Rect {
    Point ulc;
    Point lrc;
}

public class Geometry {
    static int perimeter(Rect r) {
        return (r.lrc.x - r.ulc.x
              + r.lrc.y - r.ulc.y) * 2;
    }
}
```

→

```
package geometry;

public class Rect {
    Point ulc;
    Point lrc;
    int perimeter() {
        return (lrc.x - ulc.x
              + lrc.y - ulc.y) * 2;
    }
}
```

Rétention de l'information :
Lorsqu'on masque les choix de réalisation du logiciel à l'aide d'une politique d'accès définie au niveau du code.





Méthodologie

Principe d'encapsulation : seul l'objet lui-même a le droit d'accéder à ses propres données.

```
class Customer {
    void doSomething(Stock s, Item i) {
        int qty = s.getQty(i.ref);
        ...
    }
    void doSomethingElse() {
        Item.num = 0;
        ...
    }
}
```

```
class Item {
    static int num = 0;
    final Ref ref;
    Item() {
        num += 1;
        ref = new Ref(num);
    }
    ...
}
```

`ref.val() == num`

changement de politique de génération ?
`int num` → `Generator.nextNum()`

contrainte d'intégrité ?
∀ `ref`, `ref.val()` ≤ `Item.num`

changement de nom ?
`ref` → `reference`

```
public class Item {
    private static int num = 0;
    private final Ref ref;
    public Item() {
        ref = createNewRef();
    }
    public Ref getRef() {
        return ref;
    }
    private static Ref createNewRef() {
        num += 1;
        return new Ref(num);
    }
    ...
}
```