

Rappels et compléments

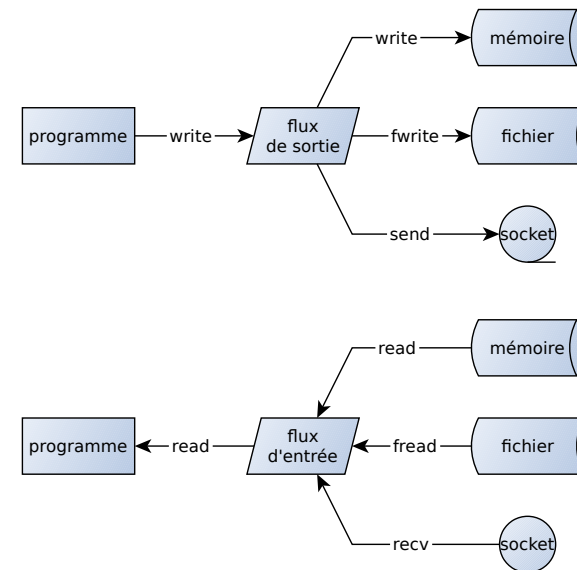
VI. Flux

1. Définition
2. Classification
 - i. Axes de classification
 - ii. Flux binaires
 - iii. Flux de caractères
3. Opérations de base
4. Exemple d'utilisation

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



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



Rappels et compléments

VI. Flux

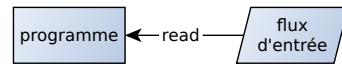
1. Définition
2. Classification
 - i. Axes de classification
 - ii. Flux binaires
 - iii. Flux de caractères
3. Opérations de base
4. Exemple d'utilisation

Classification des flux selon 3 axes :

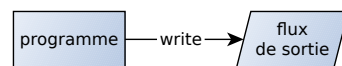
- 1/ selon le mode d'accès
- 2/ selon le type des données transportées
- 3/ selon le mode opératoire

Axe 1 "mode d'accès"

accès en lecture

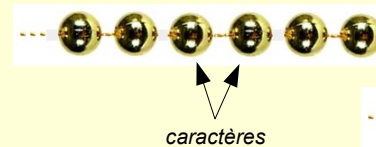


accès en écriture

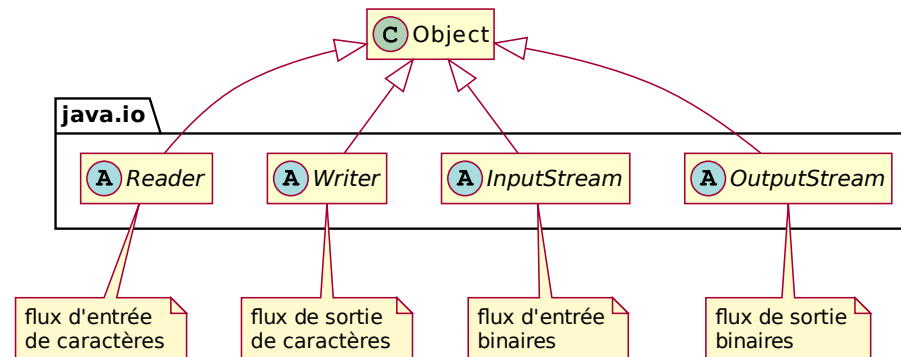
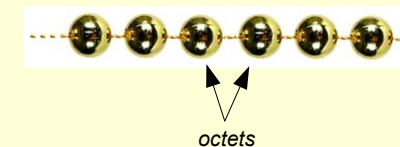


Axe 2 "type de données"

données caractères

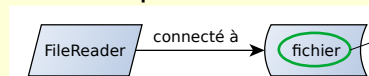


données binaires



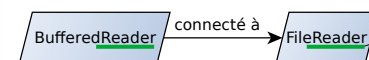
Axe 3 "mode opératoire"

primaire



les constructeurs prennent une ressource système en argument

traitement

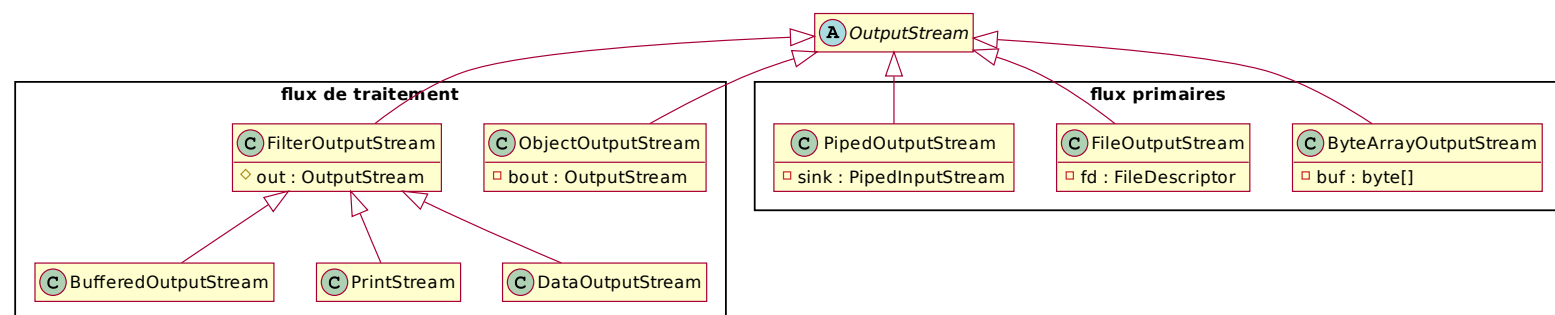
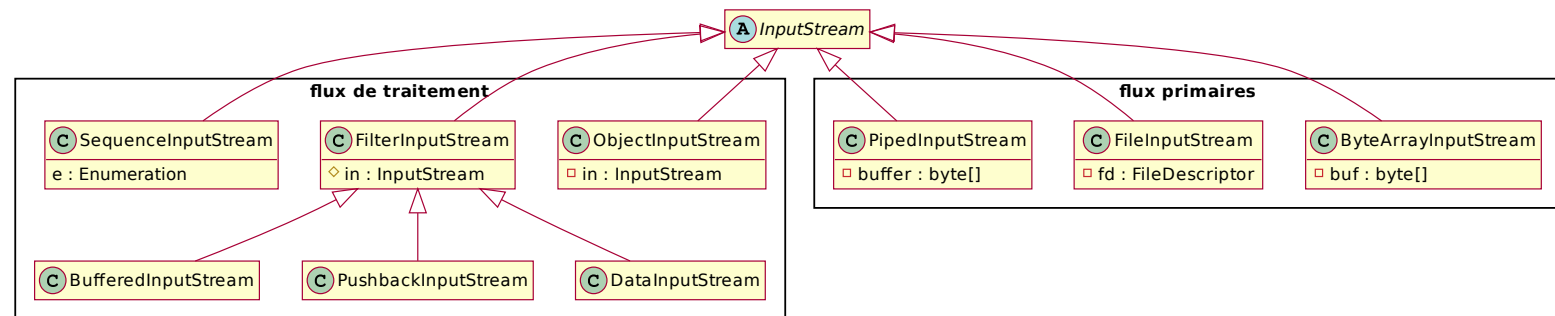
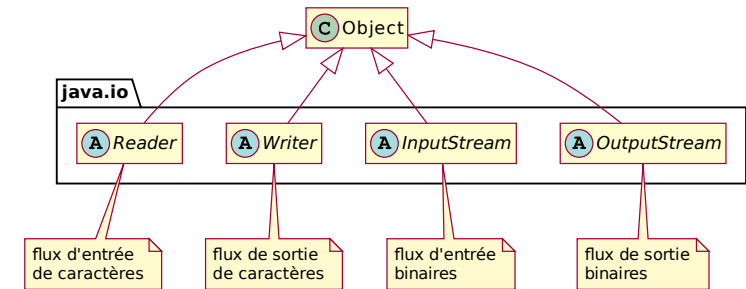


les constructeurs prennent un flux de même mode d'accès en argument

Rappels et compléments

VI. Flux

1. Définition
2. Classification
 - i. Axes de classification
 - ii. Flux binaires
 - iii. Flux de caractères
3. Opérations de base
4. Exemple d'utilisation

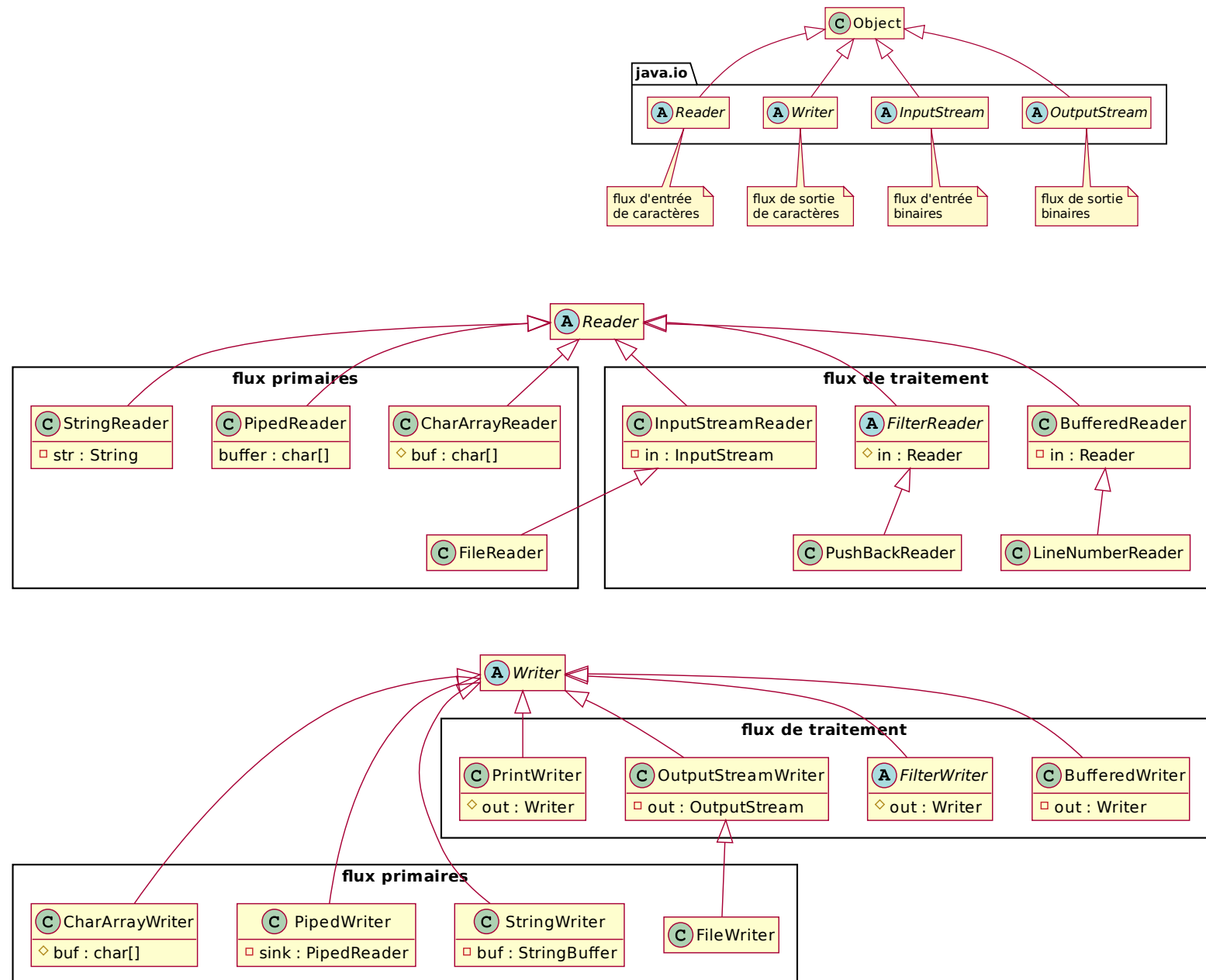


à travailler en autonomie
(lire poly + doc Java)

Rappels et compléments

VI. Flux

1. Définition
2. Classification
 - i. Axes de classification
 - ii. Flux binaires
 - iii. Flux de caractères
3. Opérations de base
4. Exemple d'utilisation



à travailler en autonomie
(lire poly + doc Java)

Rappels et compléments

VI. Flux

1. Définition
2. Classification
 - i. Axes de classification
 - ii. Flux binaires
 - iii. Flux de caractères
3. Opérations de base
4. Exemple d'utilisation

retourne 1 octet
result $\in [0..0xff] \cup \{-1\}$ (octet lu)

lit au plus $|b|$ octets et les écrit dans b
result $\in [0..|b|] \cup \{-1\}$ (nombre d'octets lus)

lit au plus len octets et les écrit dans $b[off..off+len-1]$
result $\in [0..len] \cup \{-1\}$ (nombre d'octets lus)

A *InputStream*

- `int read()`
- `int read(byte[] b)`
- `int read(byte[] b, int off, int len)`

retourne 1 caractère
result $\in [0..0xffff] \cup \{-1\}$ (caractère lu)

lit au plus $|cbuf|$ caractères et les écrit dans cbuf
result $\in [0..|cbuf|] \cup \{-1\}$ (nombre de caractères lus)

lit au plus len caractères et les écrit dans $cbuf[off..off+len-1]$
result $\in [0..len] \cup \{-1\}$ (nombre de caractères lus)

A *Reader*

- `int read()`
- `int read(char[] cbuf)`
- `int read(char[] cbuf, int off, int len)`

écrit l'octet (byte) ($b \& 0xff$)

écrit les $|b|$ octets de b

écrit les len octets de $b[off..off+len-1]$

A *OutputStream*

- `void write(int b)`
- `void write(byte[] b)`
- `void write(byte[] b, int off, int len)`

écrit le caractère (char) ($c \& 0xffff$)

écrit les $|cbuf|$ caractères de cbuf

écrit les len caractères de $cbuf[off..off+len-1]$

écrit le contenu de s

écrit le contenu de $s[off..off+len-1]$

A *Writer*

- `void write(int c)`
- `void write(char[] cbuf)`
- `void write(char[] cbuf, int off, int len)`
- `void write(String s)`
- `void write(String s, int off, int len)`

à travailler en autonomie
(lire poly + doc Java)

Rappels et compléments

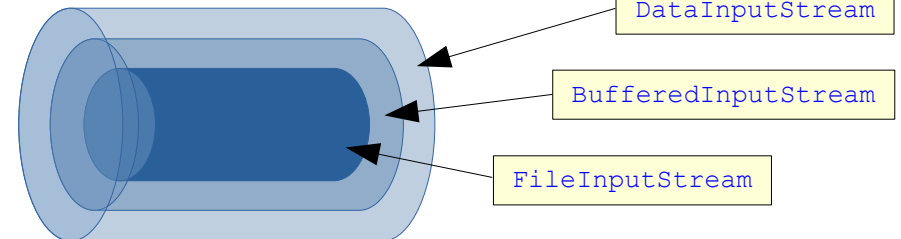
VI. Flux

1. Définition
2. Classification
 - i. Axes de classification
 - ii. Flux binaires
 - iii. Flux de caractères
3. Opérations de base
4. Exemple d'utilisation

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

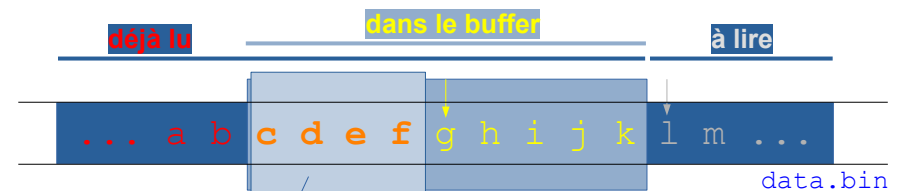
throws FileNotFoundException

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

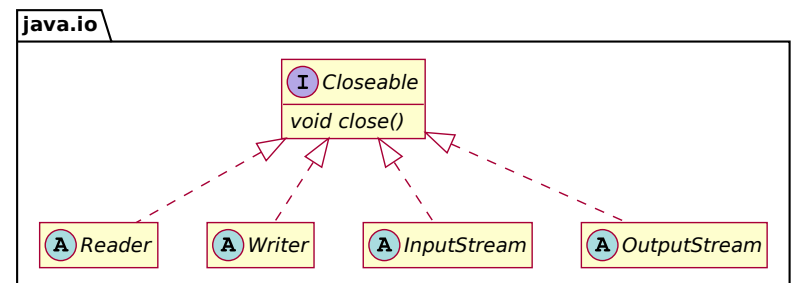
throws EOFException

throws IOException

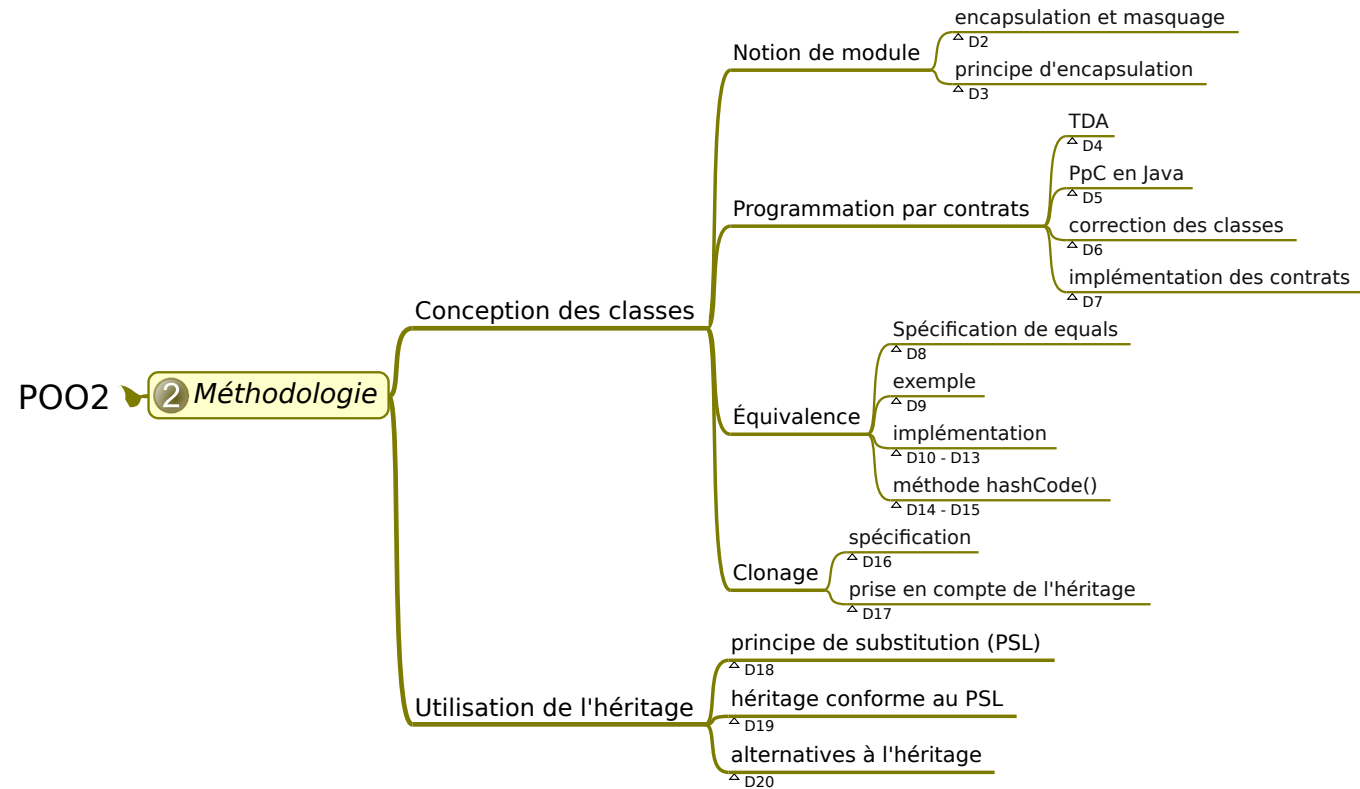
```
dis.close();
```

fermeture explicite
avec close()

throws IOException



Méthodologie



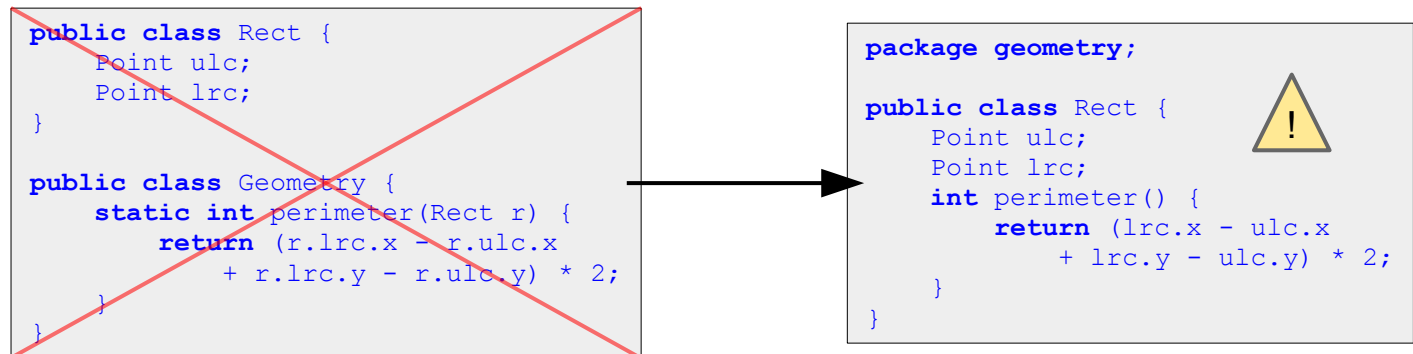
Méthodologie

- I. Conception des classes
 1. Notion de module
 - i. Encapsulation et rétention
 - ii. Principe d'encapsulation
 2. Programmation par contrat
 - i. TDA
 - ii. PpC en Java
 - iii. Correction des classes
 - iv. Implémentat° des contrats
 3. Équivalence
 - i. Spécification de equals
 - ii. Exemple
 - iii. Implémentation de equals
 - iv. Méthode hashCode
 - a. Spécification
 - b. Règles de codage
 4. Clonage
 - i. Spécification
 - ii. Prise en compte de l'héritage
- II. Utilisation de l'héritage
 1. Principe de substitution (PSL)
 2. Héritage conforme au PSL
 3. Alternatives à l'héritage

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

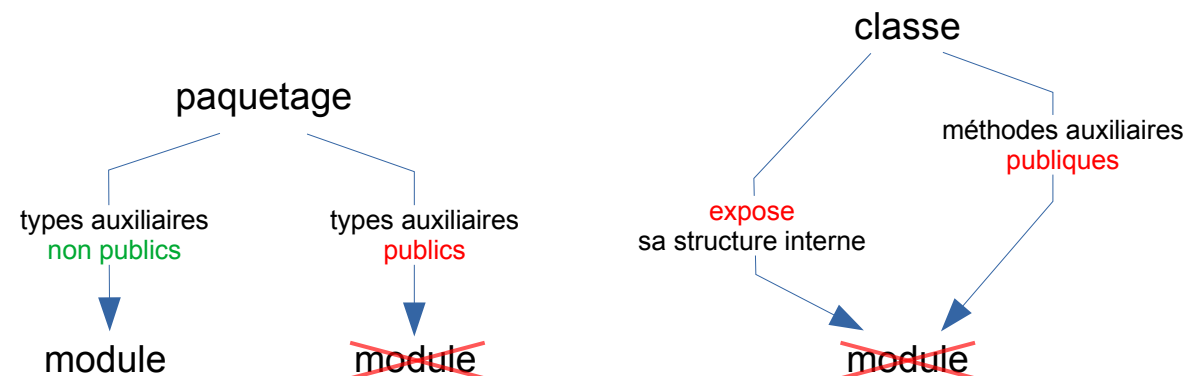
Encapsulation des données :

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



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

- I. Conception des classes
 1. Notion de module
 - i. Encapsulation et rétention
 - ii. Principe d'encapsulation
 2. Programmation par contrat
 - i. TDA
 - ii. PpC en Java
 - iii. Correction des classes
 - iv. Implémentat° des contrats
 3. Équivalence
 - i. Spécification de equals
 - ii. Exemple
 - iii. Implémentation de equals
 - iv. Méthode hashCode
 - a. Spécification
 - b. Règles de codage
 4. Clonage
 - i. Spécification
 - ii. Prise en compte de l'héritage
- II. Utilisation de l'héritage
 1. Principe de substitution (PSL)
 2. Héritage conforme au PSL
 3. Alternatives à l'héritage

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 ?
 $\forall \text{ref}, \text{ref.val}() \leq \text{Item.num}$

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);
    }
    ...
}
```