

Cours de Programmation Orientée Objet

Programmation par contrat

Marc Champesme
`mailto:Marc.Champesme@lipn.univ-paris13.fr`

18 novembre 2020

1 Introduction à la librairie standard JAVA

- La classe `Object`
 - Fonctionnalités de la classe `Object`
 - Principales méthodes de la classe `Object`
 - Contrat de la méthode `equals`
 - Application à la classe `Point`
 - Contrat de la méthode `clone`
 - Retour sur la classe `Point`
 - Expression du contrat
 - Par des commentaires
 - Résumé
-
- ## 2 Programmation par contrat
- Les classes conteneur du package `java.util`

Table des matières

1 Introduction à la librairie standard JAVA

• La classe Object

- Fonctionnalités de la classe Object
- Principales méthodes de la classe Object
- Contrat de la méthode equals
- Application à la classe Point
- Contrat de la méthode clone
- Retour sur la classe Point

• Expression du contrat

- Par des commentaires
- Résumé

2 Programmation par contrat

- Les classes conteneur du package java.util

Fonctionnalités de la classe Object

- La classe Object¹ définit des fonctionnalités communes à **toutes** les classes (**toutes** les classes héritent de la classe Object).

1. Documentation officielle de la classe Object →

<http://download.oracle.com/javase/6/docs/api/>

Fonctionnalités de la classe Object

- La classe Object¹ définit des fonctionnalités communes à **toutes** les classes (**toutes** les classes héritent de la classe Object).
- Ces fonctionnalités sont définies par l'intermédiaire de méthodes possédant chacune :

1. Documentation officielle de la classe Object →

<http://download.oracle.com/javase/6/docs/api/>

Fonctionnalités de la classe Object

- La classe Object¹ définit des fonctionnalités communes à **toutes** les classes (**toutes** les classes héritent de la classe Object).
- Ces fonctionnalités sont définies par l'intermédiaire de méthodes possédant chacune :
 - Un contrat (sous forme de commentaires) précisant le cadre à respecter pour adapter le fonctionnement de ces méthodes à **toute** nouvelle classe.

1. Documentation officielle de la classe Object →

<http://download.oracle.com/javase/6/docs/api/>

Fonctionnalités de la classe Object

- La classe Object¹ définit des fonctionnalités communes à **toutes** les classes (**toutes** les classes héritent de la classe Object).
- Ces fonctionnalités sont définies par l'intermédiaire de méthodes possédant chacune :
 - Un contrat (sous forme de commentaires) précisant le cadre à respecter pour adapter le fonctionnement de ces méthodes à **toute** nouvelle classe.
 - Une implémentation basique utilisable directement par toute nouvelle classe

1. Documentation officielle de la classe Object →

<http://download.oracle.com/javase/6/docs/api/>

Principales méthodes de la classe Object

- Ce contrat définit en particulier quel doit être le comportement commun pour deux notions très importantes : l'égalité entre objets (i.e. méthode `equals`) et la duplication d'objets (i.e. méthode `clone`).

Principales méthodes de la classe Object

- Ce contrat définit en particulier quel doit être le comportement commun pour deux notions très importantes : l'égalité entre objets (i.e. méthode `equals`) et la duplication d'objets (i.e. méthode `clone`).
- `public boolean equals(Object o) → égalité entre objets`
- `protected Object clone() → clonage d'objets`

Principales méthodes de la classe Object

- Ce contrat définit en particulier quel doit être le comportement commun pour deux notions très importantes : l'égalité entre objets (i.e. méthode `equals`) et la duplication d'objets (i.e. méthode `clone`).
- `public boolean equals(Object o)` → égalité entre objets
- `protected Object clone()` → clonage d'objets
- `public int hashCode()` → pour pouvoir utiliser les objets dans des structures de données utilisant des tables de hachage (par exemple, `java.util.HashSet` ?)
- `public String toString()` → pour un affichage de débogage.

Contrat de la méthode equals : relation d'équivalence

La documentation officielle (cf. ci-dessous) impose que la méthode `equals` implémente une relation d'équivalence (i.e. réflexive, symétrique et transitive) :

The equals method implements an equivalence relation on non-null object references :

- *It is reflexive : for any non-null reference value x , $x.equals(x)$ should return true.*
- *It is symmetric : for any non-null reference values x and y , $x.equals(y)$ should return true if and only if $y.equals(x)$ returns true.*
- *It is transitive : for any non-null reference values x , y , and z , if $x.equals(y)$ returns true and $y.equals(z)$ returns true, then $x.equals(z)$ should return true.*

Contrat de la méthode equals : cas des références null et consistance

De plus :

- le cas où l'argument est null est spécifié :

For any non-null reference value x, `x.equals(null)` should return false.

Contrat de la méthode equals : cas des références null et consistance

De plus :

- le cas où l'argument est null est spécifié :

For any non-null reference value x, `x.equals(null)` should return false.

- ainsi que la notion de consistance :

It is consistent : for any non-null reference values x and y, multiple invocations of `x.equals(y)` consistently return true or consistently return false, provided no information used in equals comparisons on the objects is modified.

Contrat de la méthode equals : implémentation de la classe Object

Le contrat précise par ailleurs quelle est l'implémentation de base fournie par la classe Object :

The equals method for class Object implements the most discriminating possible equivalence relation on objects; that is, for any non-null reference values x and y , this method returns true if and only if x and y refer to the same object ($x == y$ has the value true).

En d'autres termes, avec cette implémentation, equals à la même sémantique que l'opérateur `==`. Il convient donc d'être particulièrement attentif à la nécessité de redéfinir cette méthode pour chaque nouvelle classe.

Contrat de la méthode equals : (fin)

Le contrat donne aussi des indications sur l'incidence de l'implémentation de la méthode `equals` sur l'implémentation de la méthode `hashCode` :

Note that it is generally necessary to override the `hashCode` method whenever this method is overridden, so as to maintain the general contract for the `hashCode` method, which states that equal objects must have equal hash codes.

Au delà de la méthode `hashCode`, la façon dont `equals` est implémentée a aussi une conséquence forte sur l'implémentation de la méthode `clone()` (cf. plus loin).

Retour sur la classe Point : méthode equals

```
public class Point {  
    ...  
    public boolean equals(Object o) {  
        if (!(o instanceof Point)) {  
            return false;  
        }  
        Point autrePoint = (Point) o;  
        return getX() == autrePoint.getX()  
            && getY() == autrePoint.getY()  
    }  
    ...  
}
```


Définition du contrat de hashCode dans la classe Object

Returns a hash code value for the object. This method is supported for the benefit of hash tables such as those provided by HashMap.

The general contract of hashCode is :

- *Whenever it is invoked on the same object more than once during an execution of a Java application, the hashCode method must consistently return the same integer, provided no information used in equals comparisons on the object is modified. This integer need not remain consistent from one execution of an application to another execution of the same application.*
- ...

Définition du contrat de hashCode (suite)

Relation avec la méthode equals :

- *If two objects are equal according to the equals(Object) method, then calling the hashCode method on each of the two objects must produce the same integer result.*
- *It is not required that if two objects are unequal according to the equals(java.lang.Object) method, then calling the hashCode method on each of the two objects must produce distinct integer results. However, the programmer should be aware that producing distinct integer results for unequal objects may improve the performance of hash tables.*

Définition du contrat de hashCode (fin)

Implémentation dans la classe Object :

Implementation Requirements : As far as is reasonably practical, the hashCode method defined by class Object returns distinct integers for distinct objects.

Définition du contrat de `clone` dans la classe `Object`

Creates and returns a copy of this object. The precise meaning of "copy" may depend on the class of the object. The general intent is that, for any object x ,

- *the expression : `x.clone() != x` will be true,*
- *and that the expression : `x.clone().getClass() == x.getClass()` will be true, but these are not absolute requirements.*
- *While it is typically the case that : `x.clone().equals(x)` will be true, this is not an absolute requirement.*

By convention, the returned object should be obtained by calling `super.clone`. If a class and all of its superclasses (except `Object`) obey this convention, it will be the case that `x.clone().getClass() == x.getClass()`.

Contrat de la méthode clone (suite)

By convention, the object returned by this method should be independent of this object (which is being cloned).

To achieve this independence, it may be necessary to modify one or more fields of the object returned by `super.clone` before returning it. Typically, this means copying any mutable objects that comprise the internal "deep structure" of the object being cloned and replacing the references to these objects with references to the copies. If a class contains only primitive fields or references to immutable objects, then it is usually the case that no fields in the object returned by `super.clone` need to be modified.

Implémentation de clone par la classe Object

The method clone for class Object performs a specific cloning operation.

- *First, if the class of this object does not implement the interface Cloneable, then a CloneNotSupportedException is thrown. Note that all arrays are considered to implement the interface Cloneable.*
- *Otherwise, this method creates a new instance of the class of this object and initializes all its fields with exactly the contents of the corresponding fields of this object, as if by assignment ; the contents of the fields are not themselves cloned.*

Thus, this method performs a "shallow copy" of this object, not a "deep copy" operation.

The class Object does not itself implement the interface Cloneable, so calling the clone method on an object whose class is Object will result in throwing an exception at run time.

Règles générales pour le respect du contrat de Object

- Classe non modifiable → toujours redéfinir equals, ne jamais redéfinir clone

Règles générales pour le respect du contrat de Object

- Classe non modifiable → toujours redéfinir equals, ne jamais redéfinir clone
- Classe modifiable → décider de redéfinir equals en fonction des caractéristiques de la classe (décision au cas par cas, pas de règle générale)
- Classe modifiable → si equals est redéfini, clone doit toujours être (re)défini

Règles générales pour le respect du contrat de Object

- Classe non modifiable → toujours redéfinir equals, ne jamais redéfinir clone
- Classe modifiable → décider de redéfinir equals en fonction des caractéristiques de la classe (décision au cas par cas, pas de règle générale)
- Classe modifiable → si equals est redéfini, clone doit toujours être (re)défini
- equals redéfini → redéfinir hashCode
- equals redéfini → redéfinir toString

Retour sur la classe Point : méthode clone

On suppose la classe Point modifiable avec deux attributs int x et y.

```
public class PointModifiable implements Cloneable {  
    ...  
    // equals redéfini  
    public Object clone(Object o) {  
        Object leClone = null;  
        try {  
            leClone = super.clone();  
        } catch (CloneNotSupportedException e) {  
            throw new InternalError("Erreur impossible"  
                                    + "en théorie !");  
        }  
        // Rien à ajouter: l'implémentation de Object convient  
        return leClone;  
    }  
    ...  
}
```

Table des matières

1 Introduction à la librairie standard JAVA

- La classe Object
 - Fonctionnalités de la classe Object
 - Principales méthodes de la classe Object
 - Contrat de la méthode equals
 - Application à la classe Point
 - Contrat de la méthode clone
 - Retour sur la classe Point
- Expression du contrat
 - Par des commentaires
 - Résumé

2 Programmation par contrat

- Les classes conteneur du package java.util

Par des commentaires (format javadoc)

```
/**
 * Un point non modifiable représentant une position dans
 * un espace à deux dimensions. Les coordonnées sont spécifiées
 * par des entiers. Les opérations de transformations affines
 * courantes (homothétie, symétrie, ...) sont définies et
 * retournent des instances de cette classe dont les
 * coordonnées sont arrondies à l'entier le plus proche.
 *
 * @author Marc Champesme
 * @version 1.0 (24 novembre 2006)
 */
public class Point {
    ...
}
```

Commentaires au format javadoc : méthodes

```
...  
/**  
 * Initialise un point à la position (x, y) du système  
 * de coordonnées.  
 *  
 * @param x abscisse du point  
 * @param y ordonnée du point  
 */  
public Point(int x, int y) {  
    this.x = x;  
    this.y = y;  
}  
...
```

Résumé :

- Classe + masquage d'information (visibilité) → possibilité d'associer un contrat rigoureux à chaque classe...

Résumé :

- Classe + masquage d'information (visibilité) → possibilité d'associer un contrat rigoureux à chaque classe...
- Nécessite que le contrat soit spécifié rigoureusement pour chaque classe

Résumé :

- Classe + masquage d'information (visibilité) → possibilité d'associer un contrat rigoureux à chaque classe...
- Nécessite que le contrat soit spécifié rigoureusement pour chaque classe
- ... en utilisant des commentaires (spécification informelle)

Résumé :

- Classe + masquage d'information (visibilité) → possibilité d'associer un contrat rigoureux à chaque classe...
- Nécessite que le contrat soit spécifié rigoureusement pour chaque classe
- ... en utilisant des commentaires (spécification informelle)
- ... + des assertions (invariant, pré-conditions, post-conditions)
→ spécification plus formelle (vérifiable)

Programmation par contrat

- Introduit par Bertrand MEYER dans le langage EIFFEL (1992)

Programmation par contrat

- Introduit par Bertrand MEYER dans le langage EIFFEL (1992)
- Ajout d'assertions en complément des commentaires

Programmation par contrat

- Introduit par Bertrand MEYER dans le langage EIFFEL (1992)
- Ajout d'assertions en complément des commentaires
- Assertions exprimées dans le langage de programmation

Programmation par contrat

- Introduit par Bertrand MEYER dans le langage EIFFEL (1992)
- Ajout d'assertions en complément des commentaires
- Assertions exprimées dans le langage de programmation
- Assertions exécutables. . .
- pour vérifier que le contrat est respecté

Programmation par contrat : comment ?

- Possibilité de compiler les assertions avec le programme

Programmation par contrat : comment ?

- Possibilité de compiler les assertions avec le programme
- On peut donc vérifier les assertions pendant l'exécution du programme (avec des données réelles)

Programmation par contrat : comment ?

- Possibilité de compiler les assertions avec le programme
- On peut donc vérifier les assertions pendant l'exécution du programme (avec des données réelles)
- Utile lors de la phase de développement/test du logiciel

Programmation par contrat : comment ?

- Possibilité de compiler les assertions avec le programme
- On peut donc vérifier les assertions pendant l'exécution du programme (avec des données réelles)
- Utile lors de la phase de développement/test du logiciel
- En phase de production :
 - possibilité de désactiver le contrôle des assertions...
 - totalement ou partiellement

Qu'est-ce que le contrat ?

- Analogie avec un contrat commercial

Qu'est-ce que le contrat ?

- Analogie avec un contrat commercial
- Contrat entre un fournisseur et un client

Qu'est-ce que le contrat ?

- Analogie avec un contrat commercial
- Contrat entre un fournisseur et un client
- Chacun s'engage, chacun possède des droits et obligations

Qu'est-ce que le contrat ?

- Analogie avec un contrat commercial
- Contrat entre un fournisseur et un client
- Chacun s'engage, chacun possède des droits et obligations
- Exemple : contrat de vente d'une machine à laver
 - l'acheteur (client) doit respecter le mode d'emploi (obligation du client / droit du fabricant)
 - le fabricant (fournisseur) garantie que l'appareil fonctionne correctement conformément au mode d'emploi (droit du client / obligation du fabricant)

Qu'est-ce que le contrat ?

- Et dans le cas d'un logiciel / d'une classe ?

Qu'est-ce que le contrat ?

- Et dans le cas d'un logiciel / d'une classe ?
- Fournisseur : celui qui fournit l'implémentation de la classe

Qu'est-ce que le contrat ?

- Et dans le cas d'un logiciel / d'une classe ?
- Fournisseur : celui qui fournit l'implémentation de la classe
- Client : celui qui utilise la classe dans un autre programme

Qu'est-ce que le contrat ?

- Et dans le cas d'un logiciel / d'une classe ?
- Fournisseur : celui qui fournit l'implémentation de la classe
- Client : celui qui utilise la classe dans un autre programme
- Contrat = interface de la classe (documentation + assertions)
 - l'utilisateur de la classe (i.e. client) doit respecter le mode d'emploi définit dans la documentation
 - l'implémentation (écrite par le fournisseur) doit faire ce qui est dit dans la documentation

Trois catégories d'assertions

L'invariant :

- L'invariant porte sur la classe dans sa globalité

Trois catégories d'assertions

L'invariant :

- L'invariant porte sur la classe dans sa globalité
- C'est un ensemble d'assertions qui doivent être vraies pour toute instance de la classe :

Trois catégories d'assertions

L'invariant :

- L'invariant porte sur la classe dans sa globalité
- C'est un ensemble d'assertions qui doivent être vraies pour toute instance de la classe :
 - Après exécution de chaque constructeur

Trois catégories d'assertions

L'invariant :

- L'invariant porte sur la classe dans sa globalité
- C'est un ensemble d'assertions qui doivent être vraies pour toute instance de la classe :
 - Après exécution de chaque constructeur
 - Avant exécution de chaque méthode de la classe

Trois catégories d'assertions

L'invariant :

- L'invariant porte sur la classe dans sa globalité
- C'est un ensemble d'assertions qui doivent être vraies pour toute instance de la classe :
 - Après exécution de chaque constructeur
 - Avant exécution de chaque méthode de la classe
 - Après exécution de chaque méthode de la classe

Trois catégories d'assertions (suite)

Pré-conditions et post-conditions :

- Pré-conditions et post-conditions sont spécifiques à une méthode ou constructeur de la classe

Trois catégories d'assertions (suite)

Pré-conditions et post-conditions :

- Pré-conditions et post-conditions sont spécifiques à une méthode ou constructeur de la classe
- C'est un ensemble d'assertions qui doivent être vraies pour toute instance de la classe :

Trois catégories d'assertions (suite)

Pré-conditions et post-conditions :

- Pré-conditions et post-conditions sont spécifiques à une méthode ou constructeur de la classe
- C'est un ensemble d'assertions qui doivent être vraies pour toute instance de la classe :
 - Pré-conditions : *avant* exécution de la méthode / constructeur
 - Post-conditions : *après* exécution de la méthode / constructeur

Programmation par contrat en JAVA : JML

Les assertions sont écrites en JML (Java Modelling Language) :

- JML = JAVA + extensions

Programmation par contrat en JAVA : JML

Les assertions sont écrites en JML (Java Modelling Language) :

- JML = JAVA + extensions
- Les assertions sont écrites dans les commentaires avec des tags spécifiques :

Programmation par contrat en JAVA : JML

Les assertions sont écrites en JML (Java Modelling Language) :

- JML = JAVA + extensions
- Les assertions sont écrites dans les commentaires avec des tags spécifiques :
 - Invariant : `@invariant`
 - Pré-conditions : `@requires`
 - Post-conditions : `@ensures`

Invariant (JML)

```
/**
 * Pile d'entiers de taille bornée.
 *
 * @invariant this.estVide() <==> (this.hauteur() == 0);
 * @invariant this.estPleine()
 *             <==> (this.hauteur() == this.hauteurMax());
 * @invariant this.hauteur() >= 0;
 * @invariant this.hauteur() <= this.hauteurMax();
 * @invariant this.hauteurMax() > 0;
 */
public class PileInt {
    ...
}
```

Pré-condition (JML)

```
/**  
 * Initialise une pile d'entiers vide dont la taille  
 * maximale est l'entier spécifié.  
 *  
 * @param taille la taille maximale de cette pile  
 *  
 * @requires taille > 0;  
 */  
public PileInt(int taille) {  
    ...  
}
```

Post-condition (JML)

```
/**
 * Initialise une pile d'entiers vide dont la taille
 * maximale est l'entier spécifié.
 *
 * @param taille la taille maximale de cette pile
 *
 * @requires taille > 0;
 * @ensures this.hauteurMax() == taille;
 * @ensures this.estVide();
 * @ensures !this.estPleine();
 */
public PileInt(int taille) {
    ...
}
```

Post-condition (JML)

```
/**
 * Empile l'élément spécifié sur cette pile.
 *
 * @param elt l'élément à empiler
 *
 * @requires !this.estPleine();
 * @ensures this.sommet() == elt;
 * @ensures !this.estVide();
 * @ensures this.hauteur() == \old(this.hauteur() + 1);
 */
public void empiler(int elt) {
    ...
}
```


Post-condition (JML)

```
/**
 * Dépile cette pile et renvoie l'élément dépilé.
 *
 * @return l'élément dépilé
 *
 * @requires !this.estVide();
 * @ensures \result == \old(this.sommet());
 * @ensures !this.estPleine();
 * @ensures this.hauteur() == \old(this.hauteur() - 1);
 */
public int depiler() {
    ...
}
```

Méthodes pure (JML)

```
/**
 * Renvoie le nombre d'éléments présents dans cette pile.
 *
 * @return le nombre d'éléments de cette pile
 *
 * @ensures \result >= 0;
 * @ensures \result <= this.hauteurMax();
 *
 * @pure
 */
public int hauteur() {
    ...
}
```

Règles pour l'écriture des assertions

Certaines règles à respecter dans l'écriture des assertions :

- Uniquement des attributs / méthodes public (pas d'attributs private)
- Toutes les méthodes doivent être "pure"

Table des matières

1 Introduction à la librairie standard JAVA

- La classe Object
 - Fonctionnalités de la classe Object
 - Principales méthodes de la classe Object
 - Contrat de la méthode equals
 - Application à la classe Point
 - Contrat de la méthode clone
 - Retour sur la classe Point
- Expression du contrat
 - Par des commentaires
 - Résumé

2 Programmation par contrat

- Les classes conteneur du package java.util

Les classes conteneur du package java.util

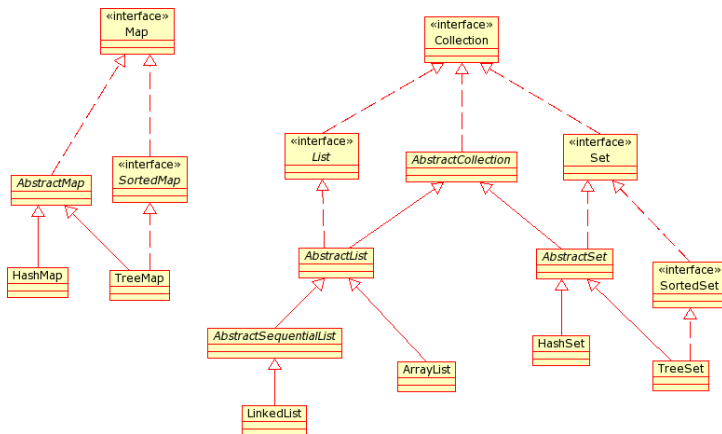


FIGURE – Hiérarchie des classes conteneur du package java.util