ITI 1521. Introduction à l'informatique II*

Marcel Turcotte
École de science informatique et de génie électrique
Université d'Ottawa

Version du 18 janvier 2012

Résumé

– Revue des concepts de programmation orientée objet :
  – Implémentation d'une classe.

*. Pensez-y, n'imprimez ces notes de cours que si c'est nécessaire!

Motd

– Disponibilité
  – Mardi de 13 à 14 h au STE 5-003
  – Vendredi de 15 à 16 h au STE 5-003
– Solution des laboratoires

Résumé

– La programmation orientée objet c'est beaucoup de choses ;
– Rend la conception de logiciels une activité plus concrète ;
– Facilite de développement de grosses applications ;
– C'est aussi un ensemble d'outils qui nous permettent de cacher les détails d'implémentation (encapsulation) ;
– C'est une façon d'organiser (structurer) les programmes telle que les données et les méthodes qui les transforment sont regroupées dans une même unité, l'**objet**.

```
public class Counter {

    private int value = 0;

    public int getValue() {
      return value;
    }

    public void incr() {
        value++;
    }

    public void reset() {
        value = 0;
    }
}
```

Résumé

Classe et **objet**, est-ce la même chose ?

– Les objets sont des entités crées lors de l'exécution.
– La classe spécifie le contenu des objets (modèle, spécification).

Dans l'exemple du jeu d'échec, il peut y avoir une classe décrivant les propriétés et comportements qui sont communs à toutes les pièces.

Les propriétés d'une pièce incluent : un nom, une couleur et une position, par exemple.

À l'exécution du programme, plusieurs ≪instances≫ seront créées : le roi noir en D8, la reine blanche en E1, etc.

Un des comportements d'une pièce pourrait être de se déplacer.

Résumé

Instance et un **objet**, est-ce le même concept ?

Oui, en effet instance et objet sont deux termes décrivant un seul et même concept.

Le terme instance est préférablement utilisé dans des phrases ayant la forme ≪l'instance de la classe . . .≫, on parle du rôle de l'objet.

Les objets sont des ≪exemples≫ (instances) d'une classe.

Package

Afin de comprendre les **modificateurs de visibilité**, il nous faut introduire la notion de package.

Qu'est qu'un package ?

Un package permet de regrouper plusieurs classes.

L'API (Application Programming Inteface) de java est structurée en package :

```
java.io
java.lang
java.util
java.awt
```

```
import java.io.*;
```

Package

Les packages peuvent contenir des packages (c'est pratique, mais sans conséquences).

```
java.awt
java.awt.color
```

Package

Une classe a accès aux autres classes du même package.

```
datebook
datebook.agenda
  Event
  Agenda
datebook.util
  Time
  TimeInterval
```

Par défaut, une classe n'est pas accessible aux classes des autres packages.

Package

L'organisation des fichiers et répertoires reflète souvent l'organisation des packages.

Fichier ≪datebook/util/Time.java≫ :

```
package datebook.util;

class Time {
    ...
}
```

Fichier : ≪datebook/agenda/Event.java≫

```
package datebook.agenda;

public class Event {
    ...
}
```

Package

Fichier ≪datebook/util/Time.java≫ :

```
package datebook.util;

class Time {
    ...
}
```

Fichier : ≪datebook/agenda/Event.java≫

```
package datebook.agenda;
import datebook.util.Time;

public class Event {
    private Time start;
}
```

L'utilisation de la classe **Time** à partir de la classe Event produira une erreur de compilation !

Package

Fichier ≪datebook/util/Time.java≫ :

```
package datebook.util;

public class Time {
    ...
}
```

Il faut faire un effort supplémentaire et ajouter le modificateur de visibilité **public**.

Package

Pourquoi souhaiterait-on définir une classe sans la rendre pulique ?

Elle est utile à l'implémentation des autres classes du même paquetage (package), mais ne doit pas être accessibles par les classes des autres paquetages.

Package

Les packages sont aussi très utiles afin d'éviter des conflits de nom.

Nous avons défini **datebook.util.Event** mais **java.awt.Event** existe aussi.

Déclaration d'une référence de type **java.awt.Event**.

```
java.awt.Event e;
```

Package

Si une classe n'est pas explicitement assignée à un package, elle appartient au package "unnamed package".

On utilise une déclaration de paquetage (package declaration) afin de spécifier le nom du paquetage auquel appartient la classe.

```
package datebook.util;

public class Time {
    ...
}
```

Classe

Qu'est-ce qu'on peut mettre dans une classe ?

Il y a au moins 48 variations, une approche hiérarchique sera utile.

- **variable**, **méthodes**, (**classes** imbriquées) ; (3)
- ces constructions appartiennent soit à l'**instance** ou à la **classe** (2) ;
- elles sont **public**, **package**, **private** ou **protected** ; (4)
- et elles sont **final** ou **pas**. (2)

Classe

- Comment déclare-t-on une variable de classe ?
- Comment déclare-t-on une variable d'instance ?

Il faut faire un effort afin de déclarer une variable de classe, ajout du mot réservé **static**.

Variable d'instance vs variable de classe

Comment décidez vous s'il s'agit d'une variable d'instance ou d'une variable de classe ?

Proposez un exemple illustrant bien la différence entre ces deux concepts.

Variable d'instance

Les variables d'instances définissent l'état (propriétés) d'un objet.

Qu'est-ce l'état ?

Les valeurs des variables à tout instant définissent l'état de l'objet.

Pensez au compteur ! Les secondes, minutes et heures d'un objet temps définissent sont état.

Variable de classe

Ticket, serialNumber, et lastSerialNumber.

```
public class Ticket {
    private static int lastSerialNumber = 0;
    private int serialNumber;

    public Ticket() {
        serialNumber = lastSerialNumber;
        lastSerialNumber++;
    }
    public int getSerialNumber() {
        return serialNumber;
    }
}
```

Variable de classe

java.lang.Math plusieurs exemples de variables et de méthodes de classe.

```
public class Math {

public static final double E = 2.7182818284590452354;

static int min( int a, int b ) { ... }
static double sqrt( double a ) { ... }
static double pow( double a, double b ) { ... }

public static double toDegrees(double angrad) {
    return angrad * 180.0 / PI;
}
```

Variable de classe

Comment fait-on un appel à une variable/méthode de classe ?

- À l'intérieur de la classe, on utilise simplement le nom de la méthode ;
- À partir d'une autre classe, on utilise le nom de la classe suivi du nom de la méthode, Math.sqrt(36) ;
- À partir d'une autre classe, on utilise une référence vers un objet de cette classe, ici **c** est une référence vers un **Counter**, **c.MAX_VALUE** (ou encore **Counter.MAX_VALUE**) sont deux façons d'accéder à la variable de classe **MAX_VALUE**.

```
Counter c;
c = new Counter();
if ( c.getValue() < c.MAX_VALUE / 2 ) { ... }
```

Méthode d'instance vs méthode de classe

Quelle serait l'impact de déclarer une méthode de classe comme étant une méthode d'instance ?

Quelle serait l'impact de déclarer une méthode d'instance comme étant une méthode de classe ?

Modificateurs de visibilité

On peut qualifier une variable, une méthode ou une classe imbriquée comme étant **public**, **package**, **private** ou **protected**.

Qu'est-ce que ça veut dire ?

Mot réservé final

On peut qualifier une variable, (une méthode) ou (une classe imbriquée) comme étant **final**.

Qu'est-ce que ça veut dire ?

Problème

On souhaite modéliser le temps sur une période de 24 heures.

En particulier, il nous faut représenter les **informations** suivantes :

heures : sur l'intervalle 0 . . .23 (inclusivement)
minutes : sur l'intervalle 0 . . .59 (inclusivement)
secondes : sur l'intervalle 0 . . .59 (inclusivement)

⇒ deux implémentations seront présentées.

Classe

La déclaration d'une classe (aspect déclaratif) débute par le mot réservé `class` suivi du nom de la classe (un identificateur dont la première lettre est une majuscule, on choisi en général un nom singulier).

Est-ce que cette déclaration est valide ?

```
public class Time {

}
```

Cette déclaration peut être mise dans un fichier nommé **Time.java**, puis compilé, et utilisé comme suit,

```
> javac Time.java
```

Classe

Peut-on utiliser la classe **Time** ? Comment ?

Est-valide ?

```
class Test {
    public static void main(String[] args) {
        Time t0;
    }
}
```

Classe

Est-valide ?

```
class Test {
    public static void main(String[] args) {
        Time t0;
        t0 = new Time();
    }
}
```

Hum, mais il n'y a pas de constructeur !

Attributs

La classe **Time** modélise des valeurs de temps (sur une période de 24 heures), quels sont ses attributs ?

Heures, minutes et secondes (variable d'instance ou de classe ?)

Constructeur

Java introduit automatiquement un constructeur par défaut :

```
public Time() {

}
```

Constructeur (suite)

Le constructeur par défaut existe, à moins que vous définissiez votre propre constructeur :

```
public class Time {
  private int hours;
  private int minutes;
  private int seconds;
  public Time( int h, int m, int s ) {
     hours = h;
     minutes = m;
     seconds = s;
  }
}
```

Ainsi, le second énoncé produira une erreur de compilation.

```
Time t;
t = new Time(); // fails!
```

Constructeur (suite)

Y-a-t-il des situations où on souhaiterait déclarer un constructeur **private** ?

```
public class Math {
  private Math() {}
}
```

Constructeur (suite)

```
class Test {
    public static void main( String[] args ) {
      Math m;
      m = new Math();
    }
}

// > javac Test.java
// Test.java:5: Math() has private access in
// java.lang.Math
//    m = new Math();
//            ^
// 1 error
```

Constructeur (suite)

Qu'est-ce qu'un constructeur ?

Se comporte comme une méthode d'instance ayant des propriétés spéciales :
- Ne peut être appelée qu'une seule fois et que dans le contexte "new . . ." ;
- Le constructeur porte le nom de la classe ;
- N'a pas de valeur de retour ;
- Java fournit un constructeur par défaut, mais seulement si vous ne définissez aucun constructeur.

Puisque le constructeur est appelé au moment de la création de l'objet seulement, il sert généralement à initialiser le contenu des variables d'instances.

Interface

Les variables et les méthodes publiques définissent l'**interface** de la classe.

Afin d'utiliser un objet (ou une classe) tout ce qu'il faut c'est en connaître l'interface.

L'interface d'une classe doit être définie soigneusement.

Seules les changements portant sur l'interface de la classe affecterons les autres parties du système logiciel.

Classe comme spécification d'un objet

```
class Time {

    int hours;
    int minutes;
    int seconds;

    Time (int h, int m, int s) {
        hours = h;
        minutes = m;
        secondes = s;
    }
    int getHours () {
        return hours;
    }
}
```

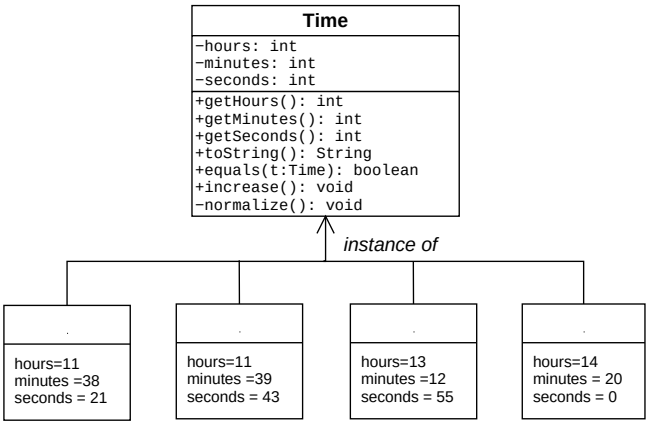
⇒ Tous les objets décrits par la classe Time auront 3 variables : hours, minutes et seconds ; 1 constructeur et méthode, getHours.

Objet

Le concept central de la programmation orientée objet est l'objet.

Un objet possède :
– des variables (état, propriétés, données),
– des méthodes (comportements, traitements, procédures).

⇒ les variables et les méthodes sont $encapsulées$ en seule entité : l'objet.



Création d'objets

On crée un objet à l'aide 1) du mot réservé **new** et 2) un appel à un constructeur (une méthode spéciale qui i) a le même nom que la classe, ii) ne retourne jamais une valeur et iii) possède 0 ou plusieurs paramètres formels).

```
a = new Time(13,0,0);
b = new Time(14,30,0);
...
n = new Time(16,45,0);
```

n objets de la classe **Time** ont été créés.

Une classe **Time** a servi a créer n objets.

Une relation 1 :n.

Création d'objets (suite)

```
a = new Time(13,0,0);
```

Lorsqu'on crée un objet, disons a, à partir d'une classe, ici Time, ont dit que a est une instance de la classe Time.

Instance n'est pas un mot de tous les jours, en particulier, aucune des définitions du dictionnaire français Larousse ne correspond, de près ou de loin au sens que l'on veut donner à instance ici.

L'une des définitions du dictionnaire anglais Webster s'en approche :

≪(. . .) instance : an individual illustrative of a category≫

⇒ . . . en quelque sorte un exemple.

Création d'objets, soyons plus précis

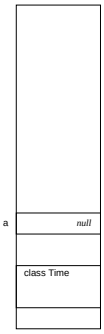
```
Time a = new Time(13,0,0);
```

Pour être plus précis, il faudrait dire que a est une variable de type référence qui pointe vers un objet de la classe **Time** !

Les variables ont des types !

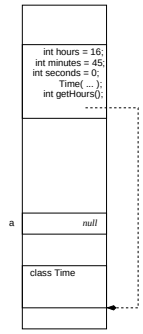
Les objets ont des classes ! (Les objets ont de la classe)

```
> Time a;
  a = new Time (16,45,0);
```



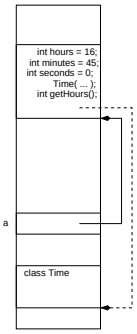
⇒ Une déclaration de variable de type référence n'alloue que la mémoire nécessaire pour une référence vers un objet.

```
  Time a;
> a = new Time (16,45,0);
```



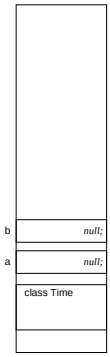
⇒ L'exécution de l'énoncé **new Time(16,45,0)** crée un objet de la classe **Time**, fait appel au constructeur **Time(int,int,int)** qui initialise les variables de l'objet ; un objet connaît sa classe, ce que nous symbolisons à l'aide d'une flèche.

```
  Time a;
> a = new Time (16,45,0);
```



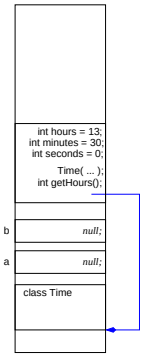
⇒ À l'exécution, l'affection, **a = ...**, met l'adresse mémoire de l'objet dans variable **a**, i.e. maintenant **a** pointe sur l'objet créé.

```
> Time a;
  a = new Time (13,30,0);
> Time b;
  b = new Time (16,45,0);
```



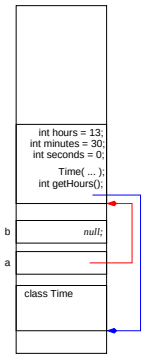
⇒ Une déclaration de variable de type référence n'alloue que la mémoire nécessaire pour une référence vers un objet.

```
  Time a;
> a = new Time (13,30,0);
  Time b;
  b = new Time (16,45,0);
```



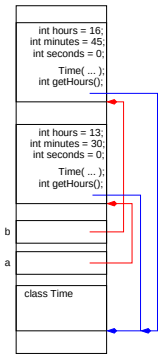
⇒ L'exécution de l'énoncé **new Time(13,30,0)** crée un objet de la classe **Time**, fait appel au constructeur **new Time(int,int,int)** qui initialise les variables de l'objet.

```
  Time a;
> a = new Time (13,30,0);
  Time b;
  b = new Time (16,45,0);
```



⇒ À l'exécution, l'affection, **a = ...**, met l'adresse mémoire de l'objet dans variable **a**, i.e. maintenant **a** pointe sur l'objet créé.

```
  Time a;
  a = new Time (13,30,0);
  Time b;
> b = new Time (16,45,0);
```



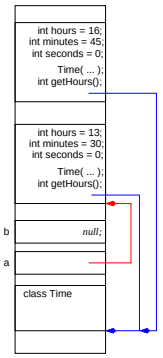
⇒ À l'exécution, l'affection, **b = ...**, met l'adresse mémoire de l'objet dans variable **b**, i.e. maintenant **b** pointe sur l'objet créé.

Solution

La programmation objet nous donne des outils afin de contrôler l'accès au données, on appelle ce principe **dissimulation d'information** (*information hiding*) :

```
class Time {
    private int hours;
    private int minutes;
    private int seconds;
}
```

```
  Time a;
  a = new Time (13,30,0);
  Time b;
> b = new Time (16,45,0);
```



⇒ L'exécution de l'énoncé **new Time(16,45,0)** crée un objet de la classe **Time**, fait appel au constructeur **new Time(int,int,int)** qui initialise les variables de l'objet.

Faiblesse de notre déclaration

La définition de notre classe est telle qu'il est possible qu'un utilisateur de la classe affecte une valeur inacceptable à une variable de l'objet.

Par exemple,

```
Time a = new Time (13,0,0);
a.hours = 55;
```

```
class Time {
    private int hours;
    private int minutes;
    private int seconds;
}

class Main {
    public static void main (String[] args) {
        Time a = new Time();
        System.out.println (a.hours);
    }
}
```

Que ce passe-t-il ?

```
Main.java:6: hours has private access in Time
        System.out.println (a.hours);
                            ^
```

Méthodes d'accès

Puisque de l'extérieur de la classe on n'a plus accès aux variables, l'objet doit posséder des méthodes qui retournent les valeurs des variables :

```
int getHours () {
  return hours;
}
```

utilisée comme suit,

```
a = new Time (13,0,0);
System.out.println (a.getHours());
-> 13
```

≪Getters≫

```
public int getHours() {
    return hours ;
}

public int getMinutes() {
    return minutes ;
}

public int getSeconds() {
    return seconds ;
}
```

≪Setters≫

Devrait-on systématiquement créer des méthodes d'accès en écriture ?

```
public void setHours( int h ) {
    hours = h;
}
```

Non ! Ça dépend du problème à résoudre. Ici, par exemple, on souhaite que la valeur d'un objet soit incrémentée d'une seconde à la fois.

Exemple d'utilisation

```
public boolean equals( Time t ) {
    return  (( hours   == t.getHours() ) &&
             ( minutes == t.getMinutes() ) &&
             ( seconds == t.getSeconds() ) ) ;
}
```

Exemple d'utilisation

```
public boolean equals( Time t ) {
    return  (( hours   == t.hours ) &&
             ( minutes == t.minutes ) &&
             ( seconds == t.seconds ) ) ;
}
```

Autre faiblesse

La définition de notre classe est telle qu'il est possible qu'un utilisateur de la classe crée un objet dont les valeurs initiales des variables sont inacceptables.

Par exemple,

```
Time a = new Time( 24,60,60 );
```

Solution

Le constructeur est une méthode qui est appelée (exécutée) au moment de la création d'un objet, c'est donc l'endroit idéal pour s'assurer que les valeurs initiales sont conformes aux spécifications.

```
public Time(int hours, int minutes, int seconds) {
    this.seconds = seconds;
    this.minutes = minutes;
    this.hours = hours;
    normalise();
}
```

de sorte que,

```
new Time (0,0,60) -> 0:1:0
new Time (0,59,60) -> 1:0:0
new Time (23,59,60) -> 0:0:0
```

normalise()

```
private void normalise () {
    int carry = seconds / 60;
    seconds = seconds % 60;
    minutes = minutes + carry ;
    carry = minutes / 60;
    minutes = minutes % 60;
    hours = (hours + carry) % 24;
}
```

On a modifié la visibilité de normalise à l'aide du mot réservé **private** parce qu'on ne veut pas que l'utilisateur de la classe utilise la méthode normalise, tout comme on ne veut pas que l'utilisateur utilise directement les variables hours, minutes, et seconds.

```
public void increase() {
    seconds++ ;
    int carry = seconds / 60;
    seconds = seconds % 60;
    minutes = minutes + carry ;
    carry = minutes / 60;
    minutes = minutes % 60;
    hours = (hours + carry) % 24;
}
```

```
public void increase() {
    seconds++ ;
    normalise();
}
```

public String toString()

Classe

Le mot réservé **class** peut être précédé d'un modificateur de visibilité.

public : toute autre classe, soit de la même unité de compilation, du même package, ou d'un autre package, peut utiliser cette classe, i.e. déclarer une variable référence de ce type.

Par défaut, i.e. lorsqu'il n'y a vas de modificateur de visibilité, l'accès est dit ≪package≫, i.e. toute classe du même ≪package≫ peut déclarer une variable de ce type (le répertoire courant constitue le package par défaut.

De plus, les mots réservés final et abstract peuvent qualifier la classe, mais nous reviendrons sur ces concepts plus tard.

Résumé

Jusqu'à maintenant nous avons considéré l'aspect déclaratif de la classe, c'est-à-dire que la classe décrit les caractéristiques communes à un ensemble d'objets.

Toutes les méthodes et variables définies jusqu'à maintenant sont des méthodes et variables d'instance.

Méthodes et variables de classe

Nous considérons maintenant l'aspect dynamique de la classe.

Un bloc mémoire est alloué pour la classe au moment de l'exécution.

Rappelons qu'une classe peut avoir plusieurs instances, mais qu'un objet n'a qu'une seule classe, et qu'il n'y a qu'une seule copie de la classe en mémoire.

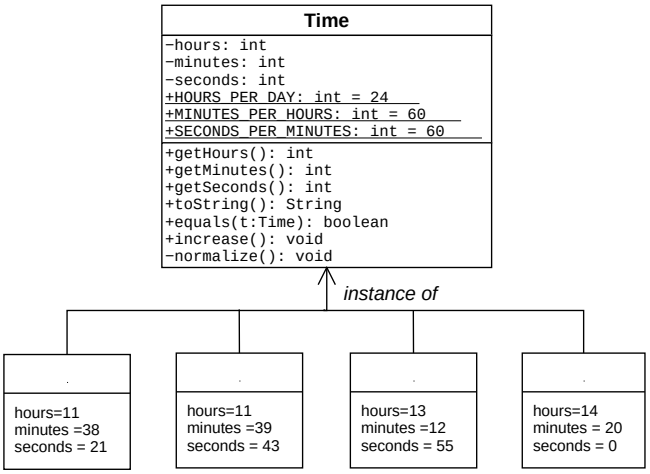
On peut associer des méthodes et des variables à une classe, en préfixant la déclaration de la méthode ou de la variable par le mot réservé **static**.

Comment détermine-t-on si une variable doit être une variable de classe ou une variable d'instance ?

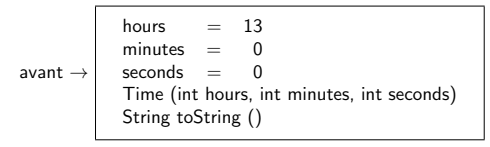
Les constantes devraient toujours être déclarée comme des variables de classe. En effet, la valeur d'une constante ne change pas lors de l'exécution d'un programme et peut donc être partagée par toutes les instances.

Exemple de variables de classe

```
class Time {

    // constantes: variables de classe (final)

    static public final int HOURS_PER_DAY      = 24;
    static public final int MINUTES_PER_HOUR   = 60;
    static public final int SECONDS_PER_MINUTE = 60;

    // variables d'instance

    private int hours;
    private int minutes;
    private int seconds;
...
}
```




```
class Time {

    static public final int HOURS_PER_DAY      = 24;
    static public final int MINUTES_PER_HOUR   = 60;
    static public final int SECONDS_PER_MINUTE = 60;

    private int hours;    // intervalle de valeurs 0 à 23 (inclusivemen
    private int minutes;  // intervalle de valeurs 0 à 59 (inclusivemen
    private int seconds;  // intervalle de valeurs 0 à 59 (inclusivemen
    ...
}

Time avant = new Time(13,0,0);
System.out.print(avant.HOURS_PER_DAY);
=> 24
System.out.print(Time.HOURS_PER_DAY);
=> 24
```

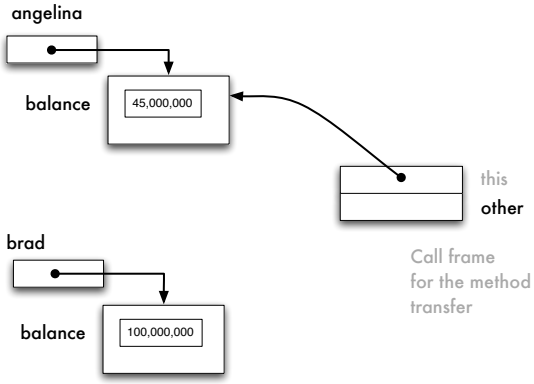


this ?

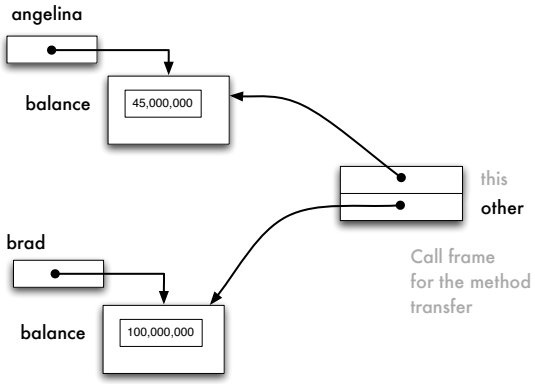
≪this≫ est une rérérence vers cet objet.

Exemple : BankAccount

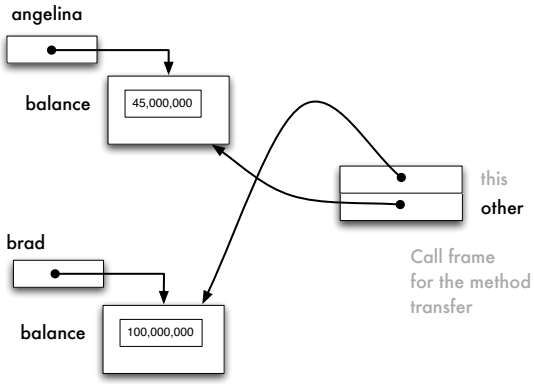
```
public class BankAccount {

  private double balance;

  // ...

  public boolean transfer( BankAccount other, double amount ) {

    if (this == other)
        return false;

    ...
  }
}
```



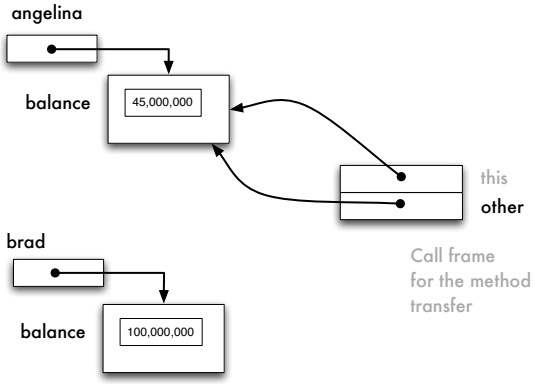
angelina.transfer(brad, 100)



angelina.transfer(brad, 100)



brad.transfer(angelina, 100)



angelina.transfer(angelina, 100)

this ?

```
public class Date {

  private int day;
  private int month;

  public Date( int day, int month ) {
     this.day = day;
     this.month = month;
  }

  // ...
}
```

Ressources

Java Language Specification
(syntaxe et sémantique du langage)

java.sun.com/docs/books/jls/second_edition/html/jTOC.doc.html

JavaTM 2 Platform, API Specification
Standard Edition, v 1.4.2
(librairies)

java.sun.com/j2se/1.5.0/docs/api