

# Java

## Méthodes

University of Geneva  
www.miralab.ch

## Méthode

2

### Définition

Fonction/procédure définie dans une classe

Comme pour les variables, on distingue

Les méthodes de classe

Invokées sans faire référence à un objet

Les méthodes d'instance

Invokées pour une instance d'objet

S'applique donc à cet objet

University of Geneva  
www.miralab.ch

LCI  
Introduction au langage C

3

## « encapsulation »

Séparation entre

le service rendu par la classe  
et son implémentation

Souvent il est judicieux de

cache les variables d'instance *private*  
en fournissant un accès par des méthodes  
qui testent la validité des paramètres fournis par  
l'utilisatrice

4

## Déclaration d'une méthode

Syntaxe similaire au C

```
type nomDeFonction(type param1,...)
{
}
```

*Mais le corps de la fonction suit toujours la  
définition*

*Pas de déclaration sans définition*

*[sauf pour les méthodes abstract c.f. héritage]*

## Méthode de classe static

```

class Compte
{ //...
    static String nomBanque = "MiraB";
    static bool initialise()
    { //...
        return true;
    }
    static String getNomBanque()
    {
        return nomBanque;
    }
}

```

## Méthode d'instance

```

class Compte
{
    private float solde;
    private boolean testSolde()
    { ... }
    public float getSolde()
    {
        if(testSolde())
            return solde; // défini après
        return 0.0f;
    }
}

```

## Autres qualificatifs

### Rappel

*public, private, static*

### *native*

Indique un code spécifique à une architecture matérielle (C/C++)

Donc non portable : attention !

### *synchronized*

Bloque l'accès à l'objet durant l'opération

Utile pour garantir les opérations en mode d'exécution concurrente

## Pas de fonction globale

Remplacée par fonction de classe

en C

en Java

```
class A {
  public static
  float puiss_n(float x,
               int n)
  {
    float puis=1.0f;
    for( ; n > 0; --n)
      puis *= x;
    return puis;
  }
}
```

```
float puiss_n(float x,
             int n)
{
  float puis=1.0f;
  for( ; n > 0; --n)
    puis *= x;
  return puis;
}
```

## Appel

9

en Java

en C

```

class B {
    static public void    void main( )
        main(String args[]) {
    {
        A.puiss_n(3.0f, 5);    }
    }
}

```

## Ordre des déclarations quelconque

10

```

class Compte
{
    private bool testSolde()
    { ... }
    public float getSolde()
    {
        if(testSolde())
            return solde; // défini après
        return 0.0f;
    }
    private float solde;
}

```

## Surcharge des méthodes

11

= définition de plusieurs méthodes de même nom

avec des paramètres différents selon  
le nombre de paramètre

méthodeCoué() , méthodeCoué(**int** un) , ...

leurs types

méthodeCoué(**int** un) , méthodeCoué(**float** un)

l'ordre des types

méthodeCoué(**int** un, **float** deux) ,

méthodeCoué(**int** un, **float** deux)

## Surcharge de méthode

12

Attention

Interdit de surcharger par type de retour de la  
fonction

**public int** méthodeCoué()

{ **return** 1; }

**public boolean** méthodeCoué() //erreur

{ **return** true; }

# Constructeur

13

## Méthode spéciale

Appelée pour créer l'objet  
Reprend le nom de la classe par convention  
Pas de type  
Qualificatifs : *public*, *private*

## Par défaut

Si aucun constructeur n'est déclaré dans la classe  
un constructeur *public* et sans paramètre  
qui ne fait que l'allocation mémoire

# Constructeur

14

```
class Compte
{
    public Compte()
    { solde = 0.0f; }
    public Compte(float solde1)
    { solde = solde1; }
    private float solde;
}
```

## Appel d'un constructeur

15

### Rappel

On utilise le mot clef *new*

```
class Comptabilité
{
    private Compte comptel;
    public Comptabilité()
    { comptel = new Compte(); }
    public Comptabilité(Compte c)
    { comptel = c; }
}
```

## Destruction des objets

16

En C++ on fait appel à l'opérateur *delete*

En *Java*

La gestion de la mémoire est contrôlée par la machine virtuelle

La destruction des objets est réalisée  
automatiquement par le *garbage collector*  
quand l'objet n'est plus utilisé  
pas forcément immédiatement



## Destruction des objets

17

Le *garbage collector* appelle la fonction *finalize()* juste avant de détruire un objet

Utile pour libérer des ressources

Typiquement des objets *native* pour libérer la mémoire de code C/C++

Méthode à définir optionnellement dans une classe

## Passage de paramètres

18

2 cas

Pour les types de base = par valeur

Pour les modifier dans une fonction il faut utiliser une classe conteneur

Pour les objets = par référence

Le changement d'un champs *public* d'un objet passé en paramètre dans une fonction modifie l'objet dans la méthode appelante

## Exemple

19

```
class Compte
{
    public Compte()
    { reset(this); }
    private void reset(Compte c)
    { c.solde = 0.0f; }
    private float solde;
}
```

University of Geneva  
www.miralab.ch

LCI  
Introduction au langage C

## Contre exemple

20

```
class Compte
{
    public Compte()
    {
        reset(this.solde); // passage par valeur!
    }
    private void reset(float s)
    { s = 0.0f; }
    private boolean reset(float s) // erreur
    { s = 0.0f; return true; }
    private float solde;
}
```

University of Geneva  
www.miralab.ch

LCI  
Introduction au langage C

21

## Paramètre constant final

Un paramètre comme une variable locale peut être qualifié(e) de constant(e)

Avec le mot clef *final*

```
public void reset(final Compte c) {}
```

Un paramètre final ne peut pas être modifié

```
Compte b = new Compte();
```

```
c = b; //erreur
```

Attention l'objet référencé peut être modifié

```
c.val = b.val; //ok
```

University of Geneva  
www.miralab.ch

LCI  
Introduction au langage C

22

## Classe conteneur

Pour pouvoir modifier un type de base passé en paramètre

On peut faire appel à une classe conteneur

```
class DoubleCont
{ public double val = -1.0;
  public DoubleCont()
  { }
  public DoubleCont(double v)
  { val = v; }
}
```

University of Geneva  
www.miralab.ch

LCI  
Introduction au langage C

23

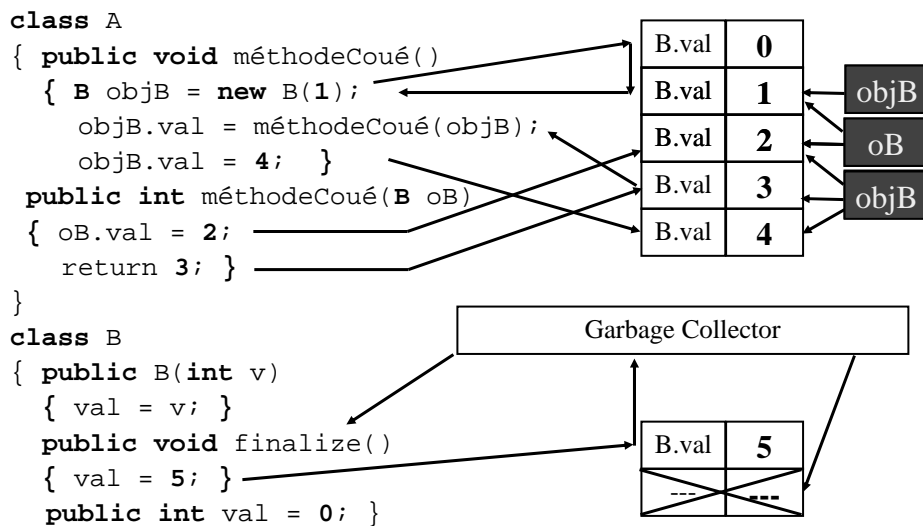
```

class Compte
{
    public Compte()
    {
        solde = new DoubleCont();
        reset(this.solde);
    }
    private void reset(DoubleCont s)
    { s.val = 0.0; }
    private DoubleCont solde;
}

```

24

## Durée de vie d'un objet



## Les exceptions

25

= interruption de l'exécution du programme  
Evènement en général lié à une erreur

Java propose une gestion de ces  
évènements à l'aide de 5 mots clefs  
*try, catch, finally, throw, throws*

## L'instruction try

26

- Sert à délimiter un bloc dans lequel une exception peut survenir
- Syntaxe :
  - `try`
  - `{`
  - `// bloc d'instructions`
  - `}`

27

## L'instruction catch

Associée à un bloc *try* { }

Sert à spécifier une action pour un type d'exception donné

Syntaxe :

```
catch (TypeException1 e)
{
    // instructions gérant l'erreur
}
```

University of Geneva  
www.miralab.ch

LCI  
Introduction au langage C

28

## L'instruction finally

Associée à un bloc *try* { }

Sert à spécifier une action à effectuer qu'il advienne ou non une exception

Syntaxe :

```
finally
{
    // instructions finales
}
```

University of Geneva  
www.miralab.ch

LCI  
Introduction au langage C

## Exemple

29

```
public Object securedPop(Pile p)
{
    Object o = null;
    try {
        o = p.pop();
    }
    catch (IOException e) { System.err.println("Erreur
        IOException: " + e.getMessage()); }
    finally {
        if (o == null)
            System.out.println("objet null !");
        return o;
    }
}
```

## L'instruction throw

30

Génère une exception sous forme d'un objet de type donné, *Throwable*...

Syntaxe :

```
throw objetException;
```

## L'instruction throws

31

Indique qu'une classe ou méthode est susceptible de générer une exception de type donné

Syntaxe :

```
class X throws objetException;
```

University of Geneva  
www.miralab.ch

LCI  
Introduction au langage C

## Exemple

32

```
public Object pop() throws  
    IOException  
{ Object obj;  
    if (size == 0)  
        throw new IOException();  
    obj = objectAt(size - 1);  
    setObjectAt(size - 1, null);  
    size--;  
    return obj;  
}
```

University of Geneva  
www.miralab.ch

LCI  
Introduction au langage C



## Résumé du cours

33

### Méthodes

de classe

d'instance

Équivalent Java d'une  
fonction globale

### Constructeur

*Garbage Collector*

### Passage de paramètre

Classe conteneur

### Durée de vie d'un objet

### Notion d'

"Encapsulation"

### Les exceptions

### Notion de surcharge