Programmer en JAVA

par Tama (tama@via.ecp.fr)

Plan

1. Présentation de Java
2. Les bases du langage
3. Concepts avancés
4. Documentation
5. Index des mots-clés
6. Les erreurs fréquentes en Java
7. Différences entre Java et C++

1. Présentation de Java

1. Qu'est-ce que Java ?
2. Les outils pour Java
3. Le concept fondamental de Java
4. Première application et compilation

1.1. Qu'est-ce que Java ?

- Langage de programmation ayant la syntaxe du C++
- Langage *portable* (s'exécute sur une *machine virtuelle*)
- Langage *orienté objet*
- Langage compilé

1.2. Les outils pour Java

- Le Java Development Kit de Sun (incluant une machine virtuelle Java), disponible sur www.sun.com
- Un éditeur spécialisé (vivement recommandé), comme Netbeans ou Eclipse.

1.3. Le concept fondamental de Java

- Tout est une classe
- Classe = structure logique pour créer des objets
- Objets dotés :
  - de propriétés
  - de méthodes et de fonctions

1.4. Première application…

```
public class Bonjour {
    public static void main (String[]
args) {
    System.out.println("Bonjour");
    }
}
```

Ce programme affiche « Bonjour »…

1.4. … et compilation

Pour faire fonctionner l'exemple précédant :

- Mettre ce code dans un fichier « Bonjour.java »
- Compiler le code par la commande : **`javac Bonjour.java`**
- Lancer l'application et vérifier le résultat : **`java Bonjour`**

2. Les bases du langage

1. Blocs et commentaires
2. Variables, constantes et types
3. Les opérateurs
4. Structures conditionnelles
5. Les classes
6. Structures itératives

2.1. Blocs et commentaires

Le code est écrit dans des blocs :

- délimités par { }
- qui peuvent être imbriqués

Les commentaires :

- Sur une seule ligne : `// mon commentaire`
- Sur plusieurs lignes : `/* */`

2.1. Blocs et commentaires

- Exemple :

```
// Une classe générique décrivant un
   animal
public class Animal {
   public Animal() {}
   public void faireQqc() {}
}
```

2.2. Variables, constantes et types

- Conventions à respecter :
  - Commencer par une lettre ou par _
    - bonne habitude à prendre : commencer par une minuscule
  - Se composer uniquement de lettres, chiffres et de _
- Une variable ou constante :
  - doit obligatoirement être déclarée
  - est d'un certain type, qui indique sa nature

2.2. Variables, constantes et types

- La plupart des types sont des objets

- Exception : les types primitifs :
  - ne sont pas des objets
  - commencent par une minuscule (**`int`**, **`char`**, **`boolean`**, …)
  - ont leurs équivalents objet : **`Integer`**, **`Char`**, **`Boolean`**, …

2.2. Variables, constantes, … (ex)

```
// Une classe générique décrivant un animal
public class Animal {
	private int age;
	private int poids;

	public Animal() {}
	public void faireQqc() {}
	public int obtenirAge() { return age; }
	public int obtenirPoids() { return
poids; }
}
```

2.3. Les opérateurs

- Arithmétiques : +, -, *, /, %
- Assignation : =, +=, *=, …
- Incrémentation : ++, --
- Comparaison : <, >, ==, <=, >=, !=
- Booléens : ||, &&, !
- Bit à bit : &, |, ^
- Rotation : <<, >>, …

2.3. Variables, constantes, … (ex)

```
// Une classe générique décrivant un animal
public class Animal {
	private int age = 0;
	private int poids = 500;

	public Animal() {}
	public void faireQqc() {}
	public int obtenirAge() { return age; }
	public int obtenirPoids() { return poids;
}
	public void grandir() { age += 1;}
	public void grossir() { poids += 500;}
}
```

2.4. Structures conditionnelles

```
public boolean estAdulte() {
   if ((age == 0) || (age == 1)) {
       return false;
   } else {
       return true;
   }
}

// Plus simple !
public boolean estAdulte() {
   return (age > 1) ? true : false;
}

// Encore plus simple...
public boolean estAdulte() { return (age > 1); }
```

2.4. Structures conditionnelles

- Instruction « switch » pour éviter les boucles « if » en cascade.

```
public boolean estAdulte() {
   switch (age) {
       case 0:
       case 1:
              return false;
              break;
       default:
              return true;
              break;
   }
}
```

2.5. Utilisation des classes

- Créer un représentant d'une classe = instancier la classe
- Avec le mot-clé « new »

```
Animal rufus = new Animal();
rufus.grandir();
rufus.grossir();
```

2.6. Instructions de base : boucle for

- Permet de faire des itérations avec un compteur.

```
/* Corps du programme
On va créer un tableau d'animaux */
Animal[] animaux = new Animal()[5];

for (int i = 0; i < animaux.length; i++) {
  animaux[i].grandir();
}
```

2.6. Instructions de base : boucle while

- Permet de répéter une action tant qu'une condition n'est pas remplie.

```
// Cette méthode fait grandir un animal
// jusqu'à ce qu'il devienne adulte
public void devenirAdulte() {
  while (!estAdulte()) {
     grandir();
  }
}
```

Récapitulatif : la classe Animal

Variables

- private int age
- private int poids

Méthodes

- public Animal()
- public void faireQqc()
- public int obtenirAge()
- public int obtenirPoids()
- public void grandir()
- public void grossir()
- public boolean estAdulte()
- public void devenirAdulte()

3. Concepts avancés

1. La surcharge de fonctions
2. Constructeurs et destructeurs
3. L'héritage
4. Les classes abstraites
5. Les interfaces
6. Les classes : divers
7. Les packages
8. La gestion des erreurs

3.1. La surcharge de fonctions

- Utiliser une même fonctionnalité dans différents contextes

```
...
public void grandir() { age += 1;}
public void grandir(int nbAnnees) {
  age += nbAnnees;
}
...
rufus.grandir();
rufus.grandir(5);
```

3.2. Le constructeur

- Même nom que la classe
- Ne renvoie aucune donnée
- Activé automatiquement à la création d'un objet

```
...
public Animal() { age = 0; poids = 1000; }
...
Animal rufus = new Animal();
// Rufus pèse 1 kg
```

3.3. L’héritage

- Améliorer / enrichir des classes préexistantes
- On peut surcharger les méthodes parentes

```
public class Chien extends Animal {
  public void faireQqc() {
      System.out.println("Ouaf !");
  }
}

Chien rufus = new Chien();
rufus.grandir(); // Rufus est un animal !
rufus.faireQqc(); // Affiche Ouaf !
```

3.3. L'héritage

- Le mot-clé protected :
  - rendre les variables accessibles aux classes filles
  - pas à l'extérieur.

```
public class Animal {
   protected int age;
   protected int poids;
   ...
}

public class Chien extends Animal {
   public boolean estVieux() { return (age > 20); }
}
```

Récapitulatif : la classe Animal

Variables

- protected int age
- protected int poids

Méthodes

- public Animal()
- public void faireQqc()
- public int obtenirAge()
- public int obtenirPoids()
- public void grandir()
- public void grossir()
- public boolean estAdulte()
- public void devenirAdulte()

Récapitulatif : la classe Chien

Variables

- protected int age
- protected int poids

Méthodes

- Venant de la classe Animal :
  - public void faireQqc()
  - public int obtenirAge()
  - public int obtenirPoids()
  - public void grandir()
  - public void grossir()
  - public boolean estAdulte()
  - public void devenirAdulte()
- Nouvelles fonctions :
  - public Chien()
  - public boolean estVieux()

3.4. Les classes abstraites

- Avec le mot-clé abstract
- Ces classes ne peuvent être instanciées

```
public abstract class Animal {
   ...
   public abstract void faireQqc();
}

// La ligne suivante provoque une erreur
Animal rufus = new Animal();
```

Récapitulatif : la classe abstraite Animal

Variables

- protected int age
- protected int poids

Méthodes

- public Animal()
- public **abstract** void faireQqc()
- public int obtenirAge()
- public int obtenirPoids()
- public void grandir()
- public void grossir()
- public boolean estAdulte()
- public void devenirAdulte()

3.4. Les classes abstraites

- Pour être utilisable, une sous-classe doit implémenter les méthodes abstraites

```
public class Chien extends Animal {
   ...
   public void faireQqc() { ... }
}

// On peut créer un chien
Chien rufus = new Chien();
```

3.5. Les interfaces

- Une classe ne peut dériver que d'une seule autre classe …
- Utilité et puissance des interfaces :
  - « aspect » homogène depuis l'extérieur
  - interface = classe purement abstraite
  - une classe peut implémenter plusieurs interfaces

3.5. Les interfaces

```
public interface Compagnon {
   protected String nom;

   public abstract void jouer();
   public abstract void parlerAvec();
   public abstract String demanderAvis(String question);
}

public class Chien extends Animal implements Compagnon {
   ...
}

Chien rufus = new Chien();
if (rufus.estVieux()) {
   String reponse = rufus.demanderAvis("Ca va ?");
}
```

3.6. Les classes : divers

- Méthodes et variables de classes avec le mot-clé static
  - on peut les appeler / les utiliser sans créer d'objet
  - les variables de classe ne sont pas initialisées à chaque instanciation
- Finalisation d'une méthode d'une classe avec le mot-clé final pour empêcher la surcharge

3.7. Les packages

- Les packages :
  - regroupent des classes ;
  - certains packages sont standards (java.lang, java.io, java.utils, …)
  - on peut également créer ses propres packages
- Commande import pour importer un package

```
package Zoo;
import Zoo.*;
import java.io.*;
```

3.8. Gestion des erreurs

- Mécanisme d'exceptions lorsqu'une erreur se produit

```
...
protected taille;

public int indice() {
     int i;
     try {
             i = (poids / taille);
     } catch (ArithmeticException e) {
             i = -1;
     } finally {
             return i;
     }
}
```

3.8. Gestion des erreurs

- Clause throws :
  - pour indiquer qu'une fonction ne traitera pas une exception
  - la fonction appelante doit gérer l'exception

```
...
protected taille;

public int indice() throws ArithmeticException {
    return (poids / taille);
}
```

```
...
int i;
try { i = rufus.indice(); } except { ... }
```

3.8. Gestion des erreurs

- Exceptions personnalisées :
  - Créer ses propres exceptions
  - On les invoque avec le mot-clé throw

```
class ExceptionTailleNulle extends Exception {
  public ExceptionTailleNulle() {}
}

...
throw new ExceptionTailleNulle();
```

4. Documentation

- La Javadoc (http://java.sun.com/j2se/1.5.0/docs/api/)
- www.google.fr
- www.commentcamarche.net
- Livres au CDI
- ...

5. Liste des mots-clés en Java

1. Structures de contrôle
2. Types de données
3. Classes et interfaces
4. Méthodes et portée
5. Divers

5.1. Structures de contrôle

- Elements génériques
  - break
  - continue
- Boucle « for »
  - for
- Boucle « while »
  - do
  - while
- Bloc « if »
  - if
  - then
  - else
- Bloc « switch »
  - switch
  - case
  - default

5.1. Structures de contrôle

- Traitement des exceptions
  - try
  - catch
  - finally
  - throw
  - throws

5.2. Types de données

- Nombres entiers
  - byte
  - short
  - int
  - long
- Nombres flottants
  - float
  - double
- Autres
  - char
  - void

5.3. Classes et interfaces

- Classes (déclaration)
  - class
  - abstract
  - extends
  - final
  - static
- Classes (utilisation)
  - super
  - instanceof
- Interfaces
  - interface
  - implements

5.4. Méthodes et portée

- Méthodes :
  - return
  - native
  - synchronised
- Portée des déclarations :
  - private
  - protected
  - public

5.5. Divers

- Objets
  - new
  - null
- Variables
  - transient
  - volatile
- Paquets
  - import
  - package

6. Les erreurs fréquentes en Java

1. La casse
2. La portée des déclarations
3. Les exceptions

6.1. La casse

- Deux variables portant le même nom, mais avec une casse différente, sont différentes (boolean <> Boolean !)
- Chaque classe doit se trouver dans un fichier qui porte le nom de la classe (attention à la casse !)
  → la classe nommée « truC » doit se trouver dans un fichier « truC.java », qui sera compilé en « truC.class ».

6.2. La portée des déclarations

- Les objets ne sont valables que dans le bloc (et les sous-blocs de ce bloc) dans lequel il a été créé
- Ainsi le code suivant ne peut pas marcher :

```
if (i != 0) { int j = 2 * i; }
System.out.println(j);
```

6.3. Les exceptions

- Une classe doit soit gérer elle-même les exceptions qu'elle peut générer, soit prévenir (dans son en-tête) qu'elle ne le fera pas.

```
public int indice() throws ArithmeticException {
    ...
}
```

7. Simplifications par rapport à C++

- Pas de pointeurs
- Pas de surcharge d'opérateurs
- Pas d'héritage multiple
- Chaînes et tableaux gérés en natif
- Gestion automatisée de la mémoire