Bases de la programmation orientée objet
Introduction

Denis Poitrenaud

IUT de l'Université Paris Descartes
Denis.Poitrenaud@ParisDescartes.fr

DUT Informatique – 2ème Semestre

Grandes lignes de ce cours

Programmation orientée objet

Le langage Java

Premiers éléments de langage

- Quelques bases aux travers d'exemples
- Types à référence
- Constantes et énumérations
- Traitements des erreurs
- Flux d'entrée/sortie fichiers en mode caractère
- Collections
- Documentation
- Développement dirigé par les tests

Organisation

- 14 semaines d'enseignement
- 4h30 heures hebdomadaires

- 2 projets – coefficient 1
- 2 DST – coefficient 3

- Pour me contacter : Denis.Poitrenaud@ParisDescartes.fr

PPN DUT Informatique – Module M2103

Contenus :

- Concepts fondamentaux de la programmation orientée objet (encapsulation, composition, polymorphisme, héritage, cycle de vie des objets)
- Lecture d'une conception orientée objet détaillée, par exemple diagramme de classes en UML (Unified Modeling Language)
- Mise en œuvre de tests unitaires
- Utilisation de briques logicielles, d'interfaces de programmation (API : Application Programming Interface), de bibliothèques
- Sensibilisation aux bonnes pratiques de la programmation, de la gestion de versions et de la documentation du code

Modalités de mise en œuvre :

- Collaboration avec le module M2104 "Bases de la conception objet"
- Apprentissage d'un langage de programmation orientée objet
- Utilisation d'un langage de modélisation objet (par exemple : UML)
- Utilisation d'un environnement de développement intégré (EDI, IDE Integrated Development Environment) d'un débogueur (debugger) et d'un environnement de test unitaire

Programmation orientée objet

Les objectifs :

- Produire vite en favorisant la réutilisation de code existant
- Produire juste en contraignant les accès directs aux données
- Produire ouvert en facilitant les extensions futures

Les moyens :

- Objet (donnée) et classe (type de données)
  - Encapsulation
  - Spécialisation/Généralisation (sous-typage)
  - Héritage

Des langages orientés objet :

- Ada, C++, C♯, Eiffel, `Java`, JavaScript, Objective-C, PHP, Python, Ruby, Simula, Smalltalk, etc

Des modèles de conception (UML) :

- Diagramme de classes, de séquences, de paquetages, etc

Le langage Java

Le langage Java est issu d'un projet de *Sun Microsystems* datant de 1990.

Généralement, on attribue sa paternité à trois de ses ingénieurs (James Gosling, Patrick Naughton, Mike Sheridan).

Java est devenu aujourd'hui l'un des langages de programmation les plus utilisés (voir `http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html`).

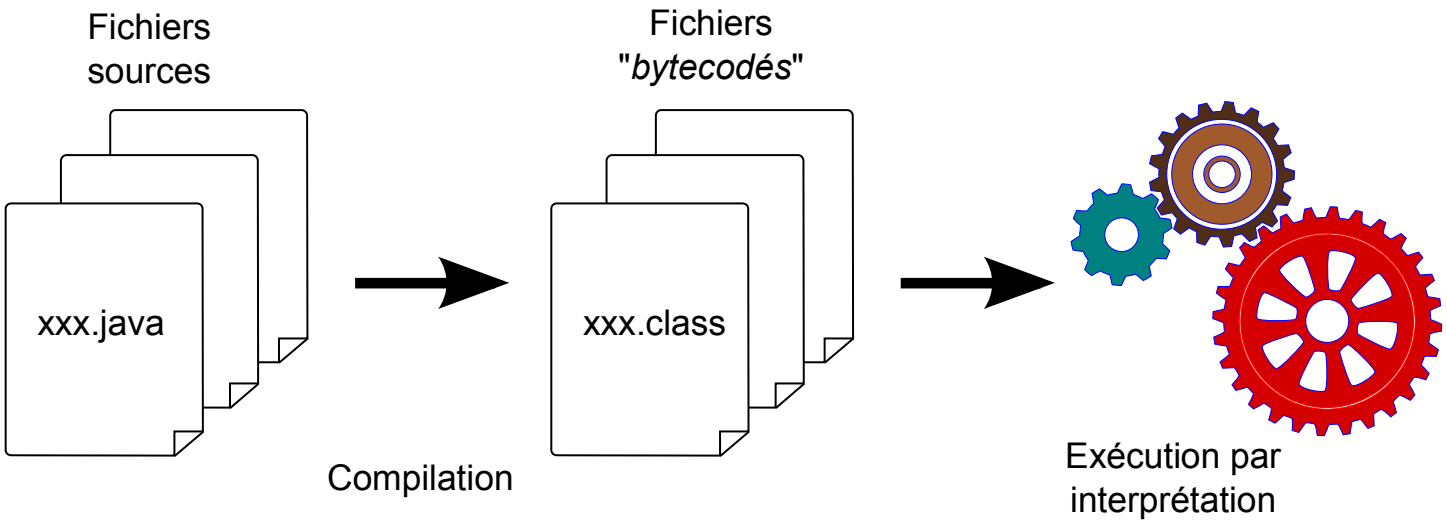
Il est incontournable dans plusieurs domaines :

- Systèmes dynamiques : Chargement dynamique de classes
- Internet : Les Applets java
- Systèmes communicants : RMI, Corba, EJB, etc.

Java versus C++

- Java est très proche du langage C++ (et donc du langage C)
- Toutefois, Java est plus simple que le langage C++, car les caractéristiques critiques de ce langage (celles qui sont à l'origine des principales erreurs) ont été supprimées
- Cela comprend
  - Les pointeurs
  - La surcharge d'opérateurs
  - L'héritage multiple

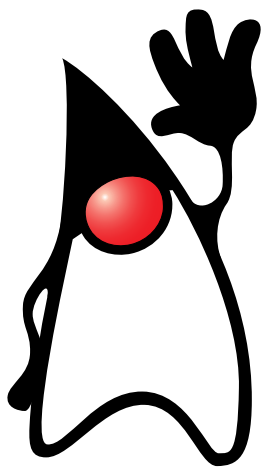
Un modèle d'exécution original



- L'interpréteur est dit être une "machine virtuelle"
  - Chaque système d'exploitation dispose d'une machine virtuelle
  - Ces machines intègrent des techniques modernes telle que la compilation à la volée pour accélérer l'interprétation du bytecode
- Ce schéma offre un bon compromis entre la portabilité, la dynamicité et la rapidité d'exécution

En résumé, qu'est-ce que Java ?

- Un langage : Orienté objet fortement typé avec classes
- Un environnement d'exécution : Une machine virtuelle et un ensemble de bibliothèques (*Java Runtime Environment – JRE*)
- Un environnement de développement : Une machine virtuelle et un ensemble d'outils (*Java Development Kit – JDK*)
- Un site de référence
  - `http://www.oracle.com/technetwork/java`
- Des IDE gratuits
  - NetBeans : `http://www.netbeans.org`
  - Eclipse : `http://www.eclipse.org`
  - IntelliJ IDEA : `http://www.jetbrains.com/idea`
-  Bruce Eckel
  Thinking in Java, 4th Edition
  Prentice Hall, 4ème édition, février 2006
  Trad. (2nd ed.) `http://penserenjava.free.fr/`
- Une mascotte : Duke



Premiers éléments de langage

Vue d'ensemble

- Types élémentaires, variables et constantes
- Structures de contrôle
- Flux d'entrée/sortie standards
- Fonctions et programmes
- Premiers types évolués
  - Chaînes de caractères, tableaux, structures
  - Allocation/désallocation mémoire
  - Passage de paramètres
  - Types énumérés
- Traitement des erreurs – les exceptions
- Flux d'entrée/sortie fichiers en mode caractère
- Automatisation des tests

Quelques bases aux travers d'exemples

Un premier programme Java

fichier `Hello.java`

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

- Tout élément de programme doit être déclaré dans une classe
- Chaque classe (publique) doit être déclarée dans son propre fichier
- Le nom de la classe et le nom du fichier doivent être cohérents
- Un programme est toujours une fonction **main** ayant le même prototype que ci-dessus
- **System.out** désigne le flux de sortie standard (par défaut l'écran) et **System.out.println** permet d'afficher sur ce flux

Compilation et exécution en ligne de commande

```
C:>cat Hello.java
public class Hello {
    public static void main(String[] args) {
        System.out.println("Hello world !!!");
    }
}

C:>javac Hello.java

C:>dir
25/11/2013  18:22               419 Hello.class
25/11/2013  18:19               126 Hello.java
               2 fichier(s)              545 octets

C:>java Hello
Hello world !!!
```

Un deuxième programme Java

```
import java.util.Scanner;

public class Age {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("saisissez votre âge : ");
        int age = sc.nextInt();
        if (age > 20)
            System.out.println("vous êtes âgé");
        else if (age > 0)
            System.out.println("vous êtes jeune");
        else
            System.out.println("vous êtes en devenir");
    }
}
```

Un deuxième programme Java

Exécution

```
saisissez votre âge :  16
vous êtes jeune
```

- Les types primitifs (**int**, **float**, **char**, etc) portent les mêmes noms qu'en C++ à l'exception des booléens (**boolean**)
- La lecture de données au clavier est basée sur le type **Scanner**
  - Il faut importer le type au début du fichier
  - Il faut "créer un objet" (**new**) avant de pouvoir s'en servir
  - **System.in** désigne le flux d'entrée standard (par défaut le clavier)
  - Des "méthodes" similaires à **nextInt** existent pour les autres types primitifs (**nextLong**, **nextFloat**, etc)
- Les variables peuvent être déclarées au milieu d'instructions (**int** age)
- Les structures de contrôle (**if**, **while**, **for**, **switch**, **break**, etc) sont les mêmes qu'en C++

Un troisième programme Java

```
public class Tableau {
  public static void f(double[] tab) {
    for (int i = 0; i < tab.length; ++i)
      for (int j = 0; j < tab.length - i - 1; ++j)
        if (tab[j] > tab[j + 1]) {
          double tmp = tab[j];
          tab[j] = tab[j + 1];
          tab[j + 1] = tmp;
        }
  }

  public static void main(String[] args) {
    double[] t = { 2.5, 7.1, 1.2, 5.7 };
    Tableau.f(t);
    for (double val : t)
      System.out.print(val + " ");
  }
}
```

Un troisième programme Java

Exécution

```
1.2  2.5  5.7  7.1
```

La boucle **for** suivante doit être lue comme “pour chaque valeur `val` présente dans le tableau `t` faire . . . ”

```
for (double val : t)
  System.out.print(val + " ");
```

Elle équivaut à la boucle suivante :

```
for (int i = 0; i < t.length; ++i) {
  double val = t[i];
  System.out.print(val + " ");
}
```

Types à référence

Les tableaux

Les tableaux sont tous dynamiques

```
String[] titres = new String[5];
```

Les dimensions multiples sont possibles

```
int[][] matrice = new int[9][9];
```

Leur taille est connue (mais non modifiable)

```
System.out.println(matrice.length + " lignes et "
                   + matrice[0].length + " colonnes");
```

Ils peuvent être le support des boucles *foreach*

```
for (String t : titres)
  System.out.println(t);

for (int[] ligne : matrice)
  for (int v : ligne)
    System.out.println(v);
```

Une première utilisation (très restrictive) des classes

Les classes peuvent être employée pour définir des types de données composites (i.e. **struct** du C/ C++)

```
public class Personne {
  String nom;
  int age;
}
```

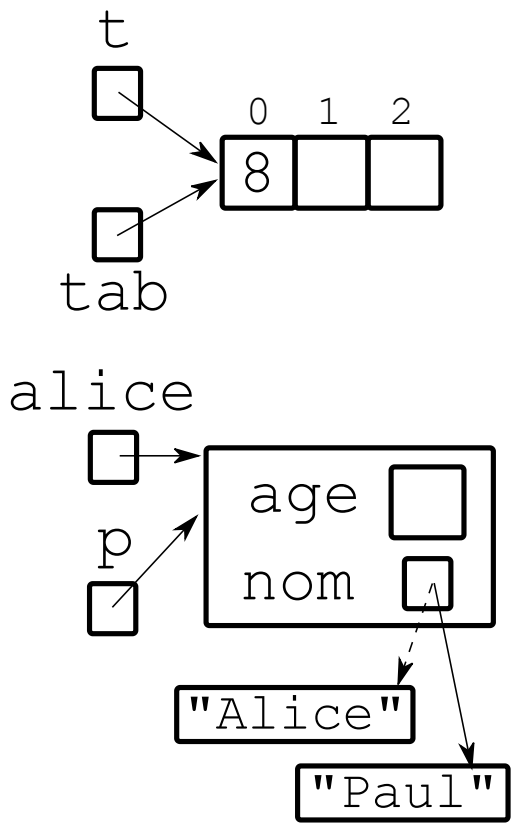
Les données d'un tel type doivent être créées dynamiquement

```
public class Appli {
  public static void main(String[] args) {
    Personne p = new Personne();
    p.nom = "Alice";
    p.age = 10;
    System.out.println(p.nom + " a "
                       + p.age + " ans");
  }
}
```

Références

Tout élément créé dynamiquement (tableau ou “structure”) est désigné par sa référence

```
1   int[] t = new int[3];
2   t[0] = 5;
3
4   int[] tab = t;
5   tab[0] = 8;
6
7   Personne alice = new Personne();
8   alice.nom = "Alice";
9
10  Personne p = alice;
11  p.nom = "Paul";
```



t[0] a été indirectement modifié ligne 5, ainsi que **alice.nom** ligne 11

Ramasse miette

À la différence du C/C++, les éléments créés dynamiquement n'ont pas à être désalloués explicitement par le programmeur

- La machine virtuelle explore régulièrement la mémoire occupée par le programme et libère automatiquement les zones mémoires qui ne sont plus référencées par des variables
- Le composant ayant cette charge dans la machine virtuelle est appelé un *ramasse miette* (garbage collector)
- Attention, une zone mémoire ne peut être libérée que si elle n'est plus référencée

```
int[] t = new int[1000000000];

// un long algorithme employant le tableau t

t = null; // le tableau peut être désalloué

// un long algorithme n'employant pas le tableau t
```

Références et passage de paramètres

- Pour un type primitif, la valeur du paramètre effectif est transmise
  $\Longrightarrow$ effet de bord impossible
- Pour tous les autres types, la référence vers la donnée est transmise
  $\Longrightarrow$ effet de bord possible

```
public class Appli {
  public static void vieilli(Personne p) {
    p.age += 1;
  }

  public static void main(String[] args) {
    Personne a = new Personne();
    a.nom = "Alice";
    a.age = 10;
    vieilli(a);
    assert (a.age == 11);
  }
}
```

Constantes et énumérations

Les constantes

Globale

```
public class Exemple {
    public static final int MAX = 100; // Exemple.MAX
}
```

Locale à une classe

```
public class Exemple {
    private static final int MAX = 100;
}
```

Locale à une fonction

```
public class Exemple {
    public static void fonction() {
        final int MAX = 100;
    }
}
```

Attention avec les références, **final** rend constant la référence mais pas la donnée référencée

Les énumérations

fichier CouleurFeu.java

```java
public enum CouleurFeu {
  VERT, ORANGE, ROUGE;
}
```

```java
public class Appli {
  public static void main(String[] args) {
    CouleurFeu rouge = CouleurFeu.ROUGE;
    System.out.println(rouge);

    CouleurFeu vert = CouleurFeu.valueOf("VERT");
    if (rouge.compareTo(vert) < 0)
      assert(rouge.ordinal() < vert.ordinal());

    CouleurFeu[] tab = CouleurFeu.values();
    for (CouleurFeu c : tab)
      System.out.println(c);
  }
}
```

Traitements des erreurs

Nature des erreurs

Au sein d'une fonction, deux situations d'erreur peuvent être détectées :

1. Un programmeur a invoqué la fonction de façon erronée (généralement en transmettant des paramètres ayant des valeurs incorrectes)
2. Le contexte du programme ne permet pas de réaliser ce qui devrait l'être (erreur d'entrée/sortie, problème matériel, etc)

- Il est raisonnable d'arrêter brutalement le programme (**assert**, **System.exit**, etc) pour les erreurs de la première famille
- Tout programme doit se prémunir (i.e. détecter et réagir) des erreurs de la seconde catégorie

Détection et réaction locales aux erreurs

Il est parfois possible de réagir localement aux erreurs

```java
/**
 * Saisie robuste au clavier d'une note (entre 0 et 20)
 * @return la note saisie au clavier
 */
public static int saisieNote() {
  Scanner sc = new Scanner(System.in);
  do {
    if (sc.hasNextInt()) { // saisie d'un entier ?
      int n = sc.nextInt(); // lit l'entier
      if (0 <= n && n <= 20) { // valeur correcte ?
        sc.close();
        return n; // seul point de sortie possible
      }
    }
    else
      sc.next(); // lit un mot (et l'ignore)
    System.out.println("raté, recommencez");
  } while (true);
}
```

Signalement par code d'erreur

Lorsque l'erreur ne peut être traitée sur place, il est parfois possible de renvoyer une valeur particulière la caractérisant

```
public static final int FORMAT_INVALIDE = -1;
public static final int VALEUR_INVALIDE = -2;

public static int conversionNote(String s) {
  Scanner sc = new Scanner(s);
  if (!sc.hasNextInt()) {
    sc.close();
    return FORMAT_INVALIDE;
  }
  int n = sc.nextInt();
  sc.close();
  if (0 > n || n > 20)
    return VALEUR_INVALIDE;
  return n;
}
```

Signalement par code d'erreur

Retourner un code d'erreur n'est pas une solution satisfaisante :

- Lorsque la valeur retournée ne permet pas d'encoder des valeurs particulières pour les erreurs, le prototype de la fonction doit être artificiellement complexifié
- Lors d'un appel de la fonction, il est nécessaire de tester qu'aucune erreur n'a été détectée
- Si l'erreur ne doit pas être prise en compte au niveau de l'appel mais à un niveau supérieur, la fonction appelante doit-elle même retourner un code d'erreur.

Tout ces inconvénients sont réglés par l'usage du mécanisme d'exception

Signalement par levée d'exception

```
public static int conversionNoteEx(String s)
    throws Exception {
  Scanner sc = new Scanner(s);
  if (!sc.hasNextInt()) {
    sc.close();
    // levée d'une exception
    throw new Exception("format invalide");
  }
  int n = sc.nextInt();
  sc.close();
  if (0 > n || n > 20)
    // levée d'une exception
    throw new Exception("valeur invalide");
  return n;
}
```

Invocation avec traitement des exceptions

```
public static void usageAvecControle() {
  Scanner sc = new Scanner(System.in);
  String s = sc.next();
  sc.close();
  try {
    // traitement sans erreur
    int n = conversionNoteEx(s);
    System.out.println(n);
  }
  catch (Exception e) {
    // traitement des erreurs
    // le contrôle est automatiquement dérouté ici dès
    // qu'une exception est levée dans le bloc try
    System.err.println("cause de l'erreur : " +
                                        e.getMessage());
  }
  // instruction toujours exécutée
  System.out.println("c'est fini");
}
```

Invocation avec remontée automatique des exceptions

```
public static void usageAvecTransmission()
        throws Exception {
    Scanner sc = new Scanner(System.in);
    String s = sc.next();
    sc.close();

    // Si une exception est levée, elle sera auto-
    // matiquement signifiée à l'appelant
    int n = conversionNoteEx(s);

    // Si une exception a été levée, le code suivant
    // ne sera pas exécuté
    System.out.println(n);
}
```

Invocation mixant traitement et remontée

```
public static void usageMixe() throws Exception {
  Scanner sc = new Scanner(System.in);
  String s = sc.next();
  sc.close();
  try {
    int n = conversionNoteEx(s);
    System.out.println(n);
  }
  catch (Exception e) {
    if (e.getMessage().startsWith("valeur"))
      System.err.println("ce n'est pas trop grave");
    else
      // retransmission explicite de l'exception
      throw e;
  }
}
```

Précondition et exception

Les divisions par zéro, les accès à un tableau avec un indice incorrect (ou à une référence nulle), etc provoquent une levée d'exception

```
1  public class Erreur {
2    public static void f() {
3      int a = 1, b = 4;
4      int[] t = {1, 2};
5      t[b] = a; // indice hors borne (4)
6      b = a / (a / b); // division par zéro
7    }
8    public static void main(String[] args) {
9      f();
10   }
11 }
```

Exécution

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 4
at cours1.Erreur.f(Erreur.java:5)
at cours1.Erreur.main(Erreur.java:9)
```

Bien que le code de `f` soit dangereux, il n'est pas obligatoire de l'encadrer par un bloc **try**/**catch** (ou de préciser sa dangerosité par une clause **throws**)

Précondition et exception

Les exceptions peuvent être employées pour signaler une violation d'une précondition

Il est usuel de lever une exception de type **RuntimeException** car elles n'imposent pas la mise en place d'un bloc try/catch

```
public static int conversionNotePré(String s) {
  Scanner sc = new Scanner(s);
  if (!sc.hasNextInt()) {
    sc.close();
    // levée d'une exception
    throw new RuntimeException("format invalide");
  }
  int n = sc.nextInt();
  sc.close();
  if (0 > n || n > 20)
    // levée d'une exception
    throw new RuntimeException("valeur invalide");
  return n;
}
```

Lors d'un appel, un paramètre incorrect provoque un arrêt brutal du programme

Bibliothèque standard et exception

Les erreurs d'emploi de la bibliothèque standard sont presque toujours signalées par des levées d'exception ayant un type spécifique à l'erreur (mais similaire à `RuntimeException`)

Class `Integer`

```
public static int parseInt(String s)
                    throws NumberFormatException
```

Parses the string argument as a signed decimal integer. The characters in the string must all be decimal digits, except that the first character may be an ASCII minus sign '-' to indicate a negative value or an ASCII plus sign '+' to indicate a positive value.

Parameters:

- `s` - a String containing the `int` representation to be parsed

Returns:

- the integer value represented by the argument in decimal.

Throws:

- `NumberFormatException` - if the string does not contain a parsable integer.

Bibliothèque standard et exceptions

Même si ce n'est pas obligatoire, les exceptions de type `RuntimeException` (et donc `NumberFormatException`) peuvent être attrapées

```
public static int conversionNote(String s)
    throws Exception {

  try {
    int n = Integer.parseInt(s);

    if (0 > n || n > 20)
      throw new Exception("valeur invalide");

    return n;

  } catch (NumberFormatException e) {
    throw new Exception("format invalide");
  }
}
```

Flux d'entrée/sortie fichiers en mode caractère

Écriture simple d'un fichier texte

```
public class FichierTexte {
  public static void ecriture(String nom) { // absolu ou relatif
    try {
      // lève une exception si le fichier ne peut être
      // ouvert en écriture (chemin incorrect, droit, etc)
      // si le fichier existe, il est écrasé
      // 'new FileOutputStream(nom, true)' pour compléter
      // un fichier existant (append)
      PrintWriter out = new PrintWriter(
                        new FileOutputStream(nom, false));

      out.println("voila des données (une ligne)");
      out.println("et encore d'autres");
      // print et println peuvent écrire des données
      // de tout type (int, float, etc).

      out.close();

    } catch (FileNotFoundException e) {
      System.out.println("impossible d'ouvrir le fichier");
    }
  }
  ...
}
```

Lecture simple d'un fichier texte

```
import java.io.*;

public class FichierTexte {
  ...
  public static void lecture(String nom) {
    try {
      // lève une exception si le fichier ne peut être
      // ouvert en lecture (inexistant, droit, etc)
      Scanner in = new Scanner(new FileInputStream(nom));

      // lecture ligne à ligne
      while (in.hasNextLine())
        System.out.println(in.nextLine());
      // hasNext() et next() peuvent être employées pour
      // une lecture mot à mot
      // (hasNextInt et nextInt pour les entiers, etc)

      in.close();

    } catch (FileNotFoundException e) {
      System.out.println("Impossible d'ouvrir le fichier");
    }
  }
}
```

Collections

Containers de la bibliothèque standard

Collections :

- `ArrayDeque`
- **`ArrayList`**
- `EnumSet`
- `HashSet`
- `LinkedHashSet`
- **`LinkedList`**
- `PriorityQueue`
- `TreeSet`
- `etc`

Tableaux associatifs :

- `EnumMap`
- **`HashMap`**
- `IdentityHashMap`
- `LinkedHashMap`
- **`TreeMap`**
- `WeakHashMap`
- `etc`

Collections – LinkedList

```
import java.util.*;

public class Containers {
  public static void collections() {
    LinkedList<String> liste = new LinkedList<String>();
    liste.addLast("C++"); liste.addLast("Ada");
    liste.addFirst("Java");

    for (String s : liste) System.out.println(s);

    if (liste.contains("PHP")) System.out.println("bizarre");
    System.out.println(liste.get(2));
    System.out.println(liste.getFirst());

    liste.remove(2); // par position
    liste.remove("C++"); // par valeur

    System.out.println(liste.size());
    System.out.println(liste);
    ...
  }
  ...
}
```

Collections – ArrayList

```
public class Containers {
  public static void collections() {
    ...
    ArrayList<Integer> tab = new ArrayList<Integer>();
    tab.addAll(Arrays.asList(1, 12, 7, 9));
    tab.add(11); // auto-boxing

    System.out.println(tab);
    Collections.sort(tab);
    System.out.println(tab);

    for (Integer i : tab) System.out.println(i);

    if (tab.contains(10)) System.out.println("bizarre");
    int n = tab.get(2); // auto-unboxing

    tab.remove(2); // par position
    tab.remove(new Integer(12)); // par valeur
    tab.set(0, 34);

    System.out.println(tab.size());
    System.out.println(tab);
  }
  ...
}
```

Documentation

Documentation

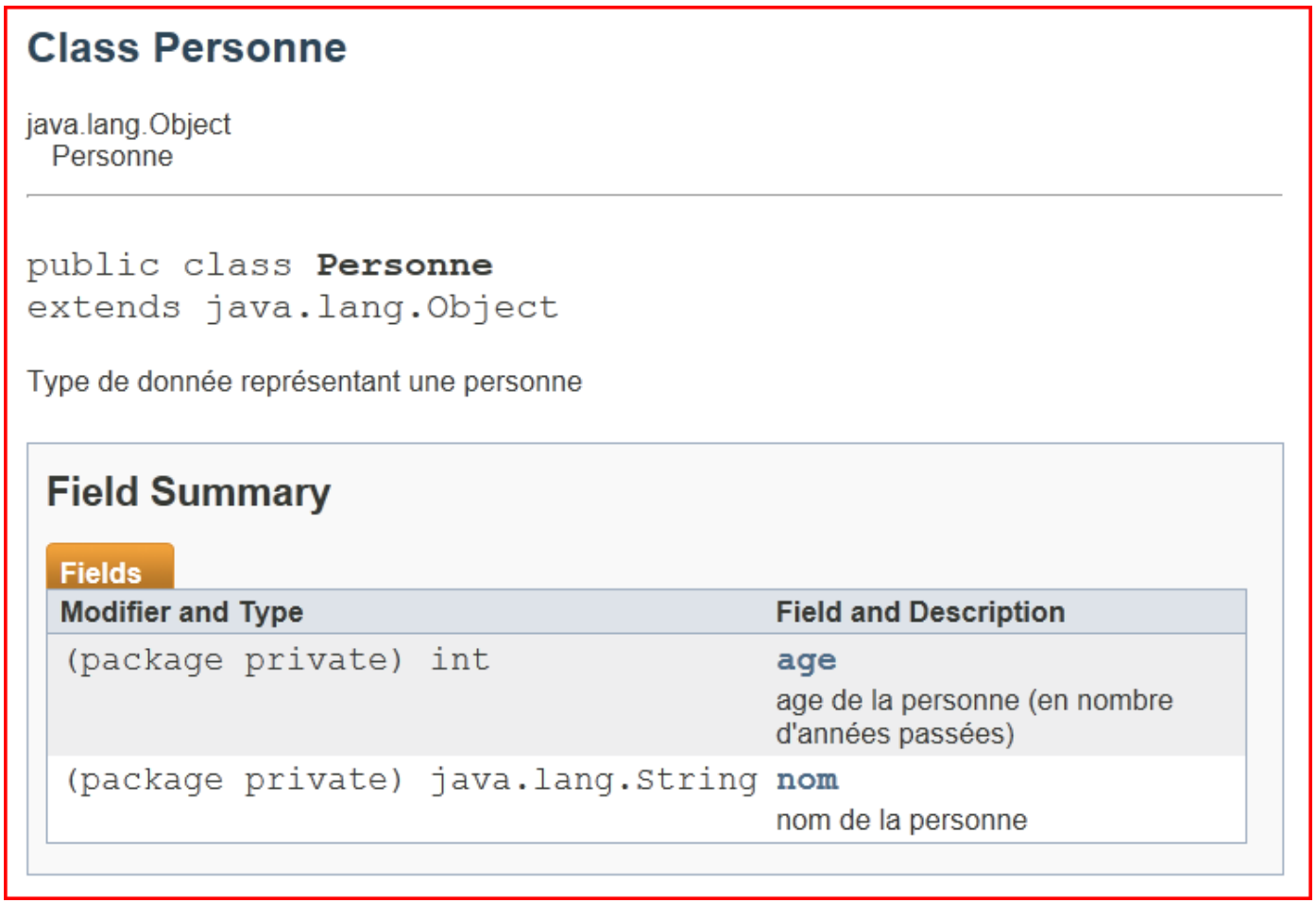
Commentaires Javadoc $\Longrightarrow$ intégration de la documentation

```java
/** Type de donnée représentant une personne */
public class Personne {
  /** nom de la personne */
  String nom;
  /** age de la personne (en années) */
  int age;
}
```

```java
public class Appli {
  /**
   * Indique la majorité d'une personne
   * @param p la personne
   * @return vrai si la personne est majeure
   */
  public static boolean estMajeure(Personne p) {
    return p.age >= 18;
  }
  ...
}
```

Avec Eclipse

Production de la documentation au format HTML en quelques clics (via l'utilitaire `javadoc`)



Class Personne

java.lang.Object
  Personne

```
public class Personne
extends java.lang.Object
```

Type de donnée représentant une personne

Field Summary

Fields

Modifier and Type	Field and Description
(package private) int	**age** age de la personne (en nombre d'années passées)
(package private) java.lang.String	**nom** nom de la personne

Développement dirigé par les tests

Développement dirigé par les tests

Un test doit

- pouvoir s'exécuter sans intervention humaine (pas de saisie)
- indiquer en fin d'exécution s'il a échoué (détecté au moins une erreur) ou non

Principe général de la méthode de développement :

1. Ajouter un test
2. Le test échoue $\Longrightarrow$ ajouter une fonctionnalité
3. Écrire le code le plus simple pour qu'il passe
4. Vérifier que le test passe
5. Remanier le code
6. Recommencer en ajoutant un autre test

Test au plus tôt

Écrire les tests avant le programme !

- Impose de découpler le programme de son environnement
- Documentation vivante
  - Spécification via les tests de recette
  - Conception via les tests unitaires
- Non régression
  - Exécution des tests à chaque nouvelle version
  - Modifications (ajouts, remaniements) en confiance
  - Langage de script (automatisation, répétition)

Type de tests

- Tests unitaires (unit tests)
  - White-box
  - Vérifient des éléments individuels du programme
  - Test de non régression après chaque modification
- Tests de recette (acceptance tests)
  - Black-box
  - Vérifient que le client a été entendu
  - Définis par le client
  - Définissent les détails des histoires !
  - Test de non régression à chaque nouvelle version

Test unitaire en Java avec JUnit

Java facilite l'écriture des tests unitaires

```
import static org.junit.Assert.*;
import org.junit.Test;
import java.util.Random;
public class TableauTest {
  @Test
  public void unTest() {
    final int TAILLE = 1000; final long GRAINE = 0;
    Random rd = new Random(GRAINE);
    double[] tab;
    tab = new double[TAILLE];
    for (int i = 0; i < TAILLE; ++i)
      tab[i] = rd.nextDouble();

    Tableau.f(tab);

    for (int i = 0; i < TAILLE - 1; ++i)
      assertTrue(tab[i] <= tab[i+1]);
  }
}
```

Avec Eclipse

Génération automatique du squelette du code de test

```
import static org.junit.Assert.*;
import org.junit.Test;

public class TableauTest {
   @Test
   public void unTest() {
      fail("Not yet implemented");
   }
}
```

Exécution et obtention d'un rapport en quelques clics