




Chapitre 2

# Bases de la POO en Java





POO en Java

## Sommaire

- **Concepts Objets de base**
  - Encapsulation, Héritage, Polymorphisme
- **Les packages Java**
- **Statics**
- **Mécanismes d'abstraction**
  - Classes abstraites et interfaces Java
- **Gestion des exceptions**
- **Travailler avec les collections**
- **Nouveautés syntaxiques des version Java 5/6/7/8**

M.Romdhani, Mars 2019

2

## Concepts Objets de base

### Les classes Java

POO en Java

#### ■ Syntaxe

```
class Nom_de_classe
{ //déclaration des membres privées
  //déclaration des membres publics
} // Pas de point virgule
```

- Il n'y a pas de bloc pour la partie privée et un bloc pour la partie publique

#### ■ L'encapsulation de données :

- Les membres privés ne sont utilisables que dans la classe
- Les membres publics sont utilisables partout

- Toutes les classes de Java héritent de `java.lang.Object`. Il héritent ainsi des méthodes telles que `getClass()`, `clone()`, `toString()`

## Classes Java : Constructeurs

- La création d'un nouvel objet est réalisée grâce à un **constructeur**
  - Ce n'est pas une véritable méthode puisque l'objet sur lequel il s'applique n'existe qu'à l'issue de son exécution
  - Un constructeur peut admettre des paramètres et possède donc une signature
- Un constructeur peut être surchargé et peut appeler le constructeur de la classe parente

**Classe\_name objclass = new Classe\_name(param);**

La destruction d'objets est faite par la machine virtuelle grâce au processus **Garbage Collector**

## Le principe d'encapsulation de données

- L'encapsulation stipule que attributs d'une classe doivent être privés

- Garantir une utilisation cohérente de données

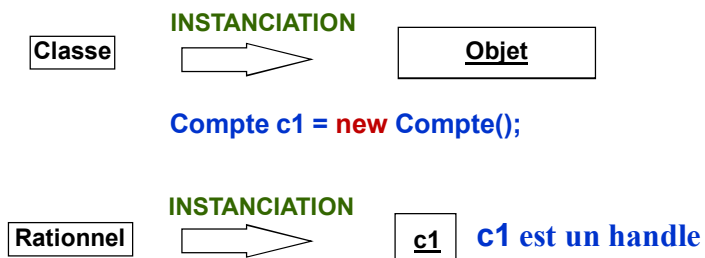
```
public class Compte {
    private String numero;    // Le numérateur du compte
    private double solde;     // Le solde courant du copte
}

public void déposer(double quantite) {
    solde += quantite;
}

public void retirer(double quantite) {
    solde -= quantite;
}
} //Fin de la déclaration de la classe
```

## Objets Java : les handles

- Les objets sont créés par instantiation d'une classe



- Le nouvel objet possède l'ensemble des propriétés (attributs) déclarés dans la classe dont il est une instance
- Toutes les méthodes déclarées dans cette même classe peuvent lui être appliquées

## Exemple de définition d'une classe

```
public class Point {
    private double x;
    private double y;

    public Point(double ax, double ay) {
        x = ax;
        y = ay;
    }

    @Override
    public String toString() {
        return "(" + x + ", " + y + ")";
    }
}
```

CLASSE

## Résumé de la syntaxe d'une Classe

### ■ Attributs/Variables d'états

### ■ Constructeurs...

### ■ Méthodes

```
public class Point {
    private double x;
    private double y;

    public Point(double x,double y) {
        this.x = x;
        this.y = y;
    }
    public String toString() {
        return "("+x+","+y+")";
    }
}
```

CLASSE

## Constructeurs (1/5)

■ **Rappel** : toute variable doit être initialisée

■ **Question** : comment initialiser les variables d'une instance?

■ **Réponse** : le (ou les) constructeurs!

```
public class Point {
    private double x;
    private double y;

    public Point(double x,double y) {
        this.x = x;
        this.y = y;
    }
}
```

CLASSE

Point p = new Point(1,2);

INSTANCES

## Constructeurs (2/5)

### ■ Plusieurs constructeurs sont possibles

```
public class Point {
    // ...
    public Point(double x,double y) {
        this.x = x;
        this.y = y;
    }
    public Point() {
        x = 0;
        y = 0;
    }
}
```

CLASSE

```
Point p = new Point(1,2);
`
```

INSTANCES

```
Point q = new Point();
```

## Constructeurs (3/5)

### ■ Constructeurs par défaut

```
public class Point {
    public double x;
    public double y;
}
```

CLASSE

```
Point p = new Point();
```

INSTANCES

```
Point q = new Point(1,2);
```

refusé à la compilation

```
public class Point {
    private double x;
    private double y;
    public Point(double x,double y) {
        this.x = x;
        this.y = y;
    }
}
```

CLASSE

```
Point p = new Point();
```

INSTANCES

```
Point q = new Point(1,2);
```

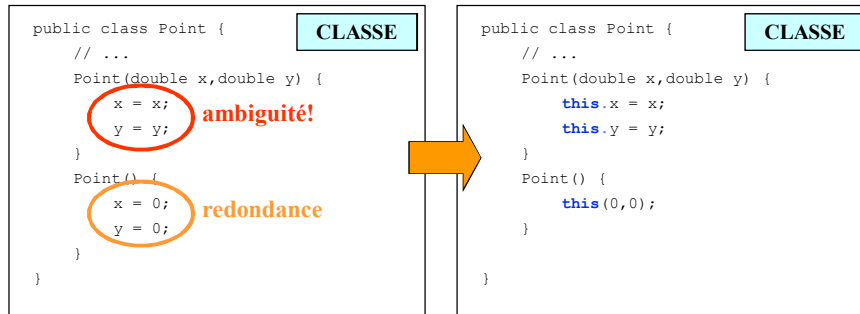
refusé à la compilation

## Constructeurs (4/5)

POO en Java

### ■ Le mot clé **this** pour:

- lever les ambiguïtés
- réutiliser un constructeur



M. Romdhani, Mars 2019

13

## L'héritage en Java

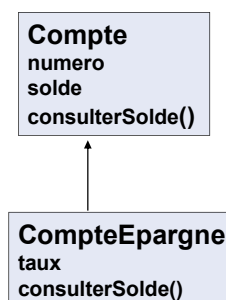
POO en Java

### ■ Syntaxe

```

class Compte{
    // déclarations
}
class CompteEpargne extends Compte{
    //Déclarations
}

```



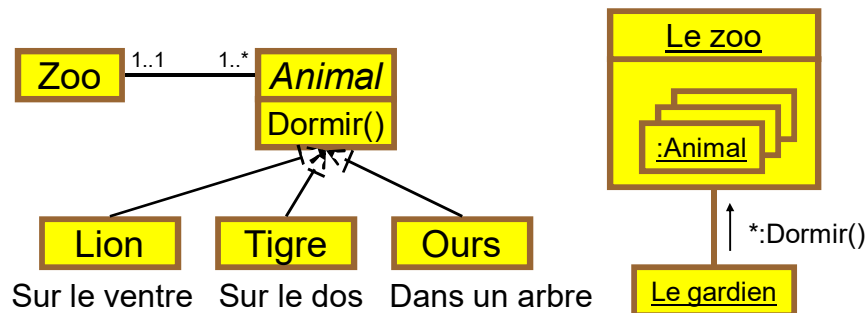
- Que l'héritage simple en Java.
- Les classes dérivent, par défaut, de `java.lang.Object`.
- Une référence d'une classe C peut contenir des instances de C ou des classes dérivées de C.
- L'opérateur **instanceOf** permet de déterminer la classe d'une instance.
- Les classes **final** ne peuvent pas être redéfinies dans les sous-classes.

M. Romdhani, Mars 2019

14

## Polymorphisme

- Un nom d'objet peut désigner des instances de classes différentes issues d'une même arborescence
- L'objet peut réagir différemment à des opérations communes



## Polymorphisme : illustration

```

Abstract class Animal {
    void dormir();
}
class Lion extends Animal {
    public void dormir() {
        System.out.println(" Je suis le lion et je dors sur le ventre");
    }
}
class Tigre extends Animal{
    public void dormir(){
        System.out.println(" Je suis le tigre et je dors sur le dos");
    }
}
class Ours extends Animal{
    public void dormir(){
        System.out.println(" Je suis l'ours et je dors sur un arbre");
    }
}

class IllustrationPolymorphisme {
    public static void main(String[] args) {
        Animal[] tab = { new Tigre(), new Lion(), new Tigre(),new Ours()};

        for (int i=0; i<tab.length; i++)
            tab[i].dormir();
    }
}

```



# Les packages Java

## Les packages

POO en Java

- **Dans le cas d'une application réelle, on regroupe les classes liées au même domaine dans une structure de package.**
  - Mieux organiser les classes, et maîtriser la complexité d'une application
  - L'API de java est elle aussi organisée en packages: `java.lang`, `java.util`, `java.io`, ...
- **Un package regroupe un ensemble de classes sous un même espace de nommage.**
- **Les noms des packages suivent le schéma :** `name.subname ...`
- **Une classe `Watch` appartenant au package `time.clock` doit se trouver dans le fichier `time/clock/Watch.class`**  
**Le nom du package doit concorder avec le nom du répertoire correspondant**
- **Les packages permettent au compilateur et à la JVM de localiser les fichiers contenant les classes à charger. Elle utilise pour cela la variable d'environnement `CLASSPATH`**
- **L'instruction `package` indique à quel package appartient la ou les classe(s) de l'unité de compilation (le fichier).**

## Déclaration d'un package

- L'instruction `package` indique à quel package appartient la ou les classe(s) de l'unité de compilation (le fichier).

Exemple

```
./be/businesstraining/2d/Circle.java
package be.businesstraining.2d;
public class Circle
{ ... }
```

```
./be/businesstraining/3d/Sphere.java
package be.businesstraining.3d;
public class Sphere
{ ... }
```

## Utilisation d'un package

- L'instruction `import packageName` permet d'utiliser des classes sans les préfixer par leur nom de package.

■ Exemple

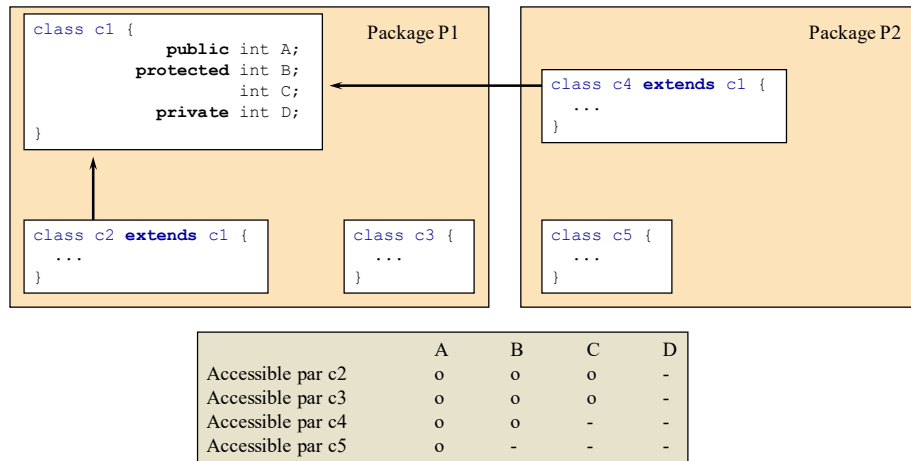
```
./be/businesstraining/paintShop/MainClass.java
package be.businesstraining.paintShop;

import be.businesstraining.2d.*;
import be.businesstraining.3d.*;

public class MainClass()
{
    public static void main(String[] args) {
        Circle c1 = new Circle(50);
        Circle c2 = new Circle(70);
        Sphere s1 = new Sphere(100);
        Sphere s2 = new Sphere(40);
    }
}
```

## Les modificateurs private, public et protected en Java

POO en Java



M.Romdhani, Mars 2019

21

## Statics

## Static Variables and Methods

- The following code declares and uses a static counter variable:

```
class Frog {
    static int frogCount = 0; // Declare and initialize static variable
    public Frog() {
        frogCount += 1; // Modify the value in the constructor
    }
    public static void main (String [] args) {
        new Frog(); new Frog(); new Frog();
        System.out.println ("Frog count is now " + frogCount);
    }
}
```

- When this code executes, three Frog instances are created in main(), and the result is  
Frog count is now 3
- The following code is an example of illegal access of a nonstatic variable from a static method:

```
class Foo {
    int x = 3;
    public static void main (String [] args) {
        System.out.println("x is " + x);
    }
}
```

**Understand that this code will never compile, because you can't access a nonstatic (instance) variable from a static method. Just think of the compiler saying**

## Accessing Static Methods and Variables

```
class Foo
{
    int size = 42;
    static void doMore( ){
        int x = size;
    }
}
```

static method cannot  
access an instance  
(non-static) variable

```
class Bar
{
    void go ( );
    static void doMore( ){
        go( );
    }
}
```

static method cannot  
access a non-static  
method

```
class Baz
{
    static int count;
    static void woo( ){ }
    static void doMore( ){
        woo( );
        int x = count;
    }
}
```

static method  
can access a static  
method or variable

## Sun's Java Code Conventions

### ■ Classes and interfaces :

- **The first letter should be capitalized**, and if several words are linked together to form the name, the first letter of the inner words should be uppercase (a format that's sometimes called "PascalCase"). For classes, the names should typically be nouns.
  - For example: Dog , Account , PrintWriter
- For interfaces, the names should typically be adjectives like :Runnable Serializable

### ■ Methods

- **The first letter should be lowercase**, and then normal camelCase rules should be used. In addition, the names should typically be verb-noun pairs.
  - For example: getBalance() doCalculation(), setCustomerName()

### ■ Variables

- Like methods, the camelCase format should be used, **starting with a lowercase letter**. Sun recommends short, meaningful names, which sounds good to us.
  - Some examples: buttonWidth accountBalance myString

### ■ Constants

- Java constants are created by marking variables **static and final**. They should be named using **uppercase letters** with underscore characters as separators:
  - MIN\_HEIGHT

## Mécanismes d'abstraction

## Les classes abstraites

- Une classe abstraite est une classe ayant au moins une méthode abstraite.
- Une méthode abstraite ne possède pas de définition.
- Une classe abstraite ne peut pas être instanciée (`new`).
- Une classe dérivée d'une classe abstraite ne redéfinissant pas toutes les méthodes abstraites est elle-même abstraite.

## Exemple de classe abstraite

```
class abstract Shape {
    public abstract double perimeter();
}

class Circle extends Shape {
    ...
    public double perimeter() { return 2 * Math.PI * r ; }
}

class Rectangle extends Shape {
    ...
    public double perimeter() { return 2 * (height + width); }
}

...
Shape[] shapes = {new Circle(2), new Rectangle(2,3), new Circle(5)};
double sum_of_perimeters = 0;
for(int i=0; i<shapes.length; i++)
    sum_of_perimeters = shapes[i].perimeter();
```

## Les interfaces

- Une interface correspond à une classe où toutes les méthodes sont abstraites.
- Une classe peut implémenter (**implements**) une ou plusieurs interfaces tout en héritant (**extends**) d'une classe.
- Une interface peut hériter (**extends**) de plusieurs interfaces.

## Exemple d'interface

```
public interface Shape {
    double perimeter();
}

public class Circle implements Shape {
    ...
    public double perimeter() { return 2 * Math.PI * r ; }
}

public class Rectangle implements Shape {
    ...
    public double perimeter() { return 2 * (height + width); }
}

...
Shape[] shapes = {new Circle(2), new Rectangle(2,3), new Circle(5)};
double sum_of_perimeters = 0;
for(int i=0; i<shapes.length; i++)
    sum_of_perimeters = shapes[i].perimeter();
```

# Gestion des exceptions

## Traitement des exceptions Principes

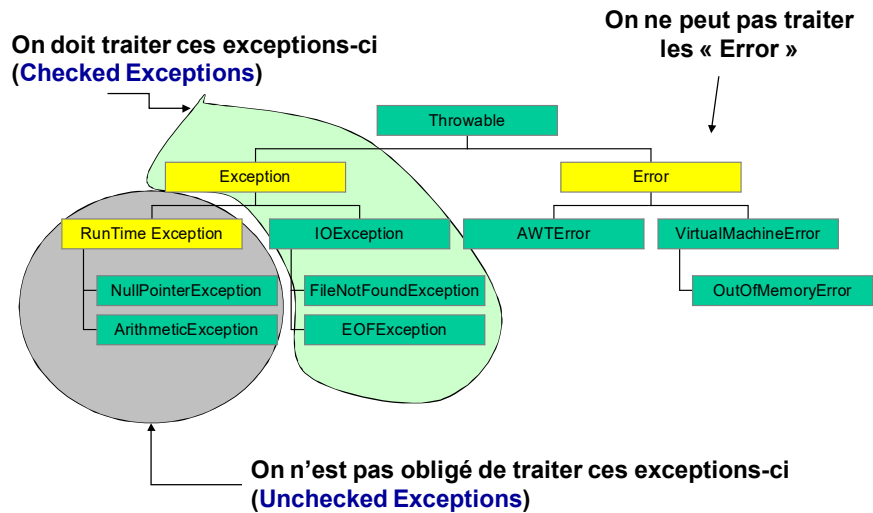
POO en Java

### ■ Le traitement des exceptions contient deux aspects:

- L'interception des exceptions
  - Utilisation du bloc *try – catch – finally* pour récupérer les exceptions
  - Et réaliser les actions nécessaires
- Le lancement (la génération) d'exceptions
  - Automatiquement par l'environnement run-time ou la machine virtuelle pour certaines exceptions prédéfinies par Java
  - Explicitement par le développeur dans une méthode avec « throws » et « throw » (en tout cas pour les exceptions créées par le développeur)



## Hiérarchie des exceptions



## Exceptions Les exceptions (1)

- Elles permettent de séparer un bloc d'instructions de la gestion des erreurs pouvant survenir dans ce bloc.

```

try {
    // Code pouvant lever des IOException ou des SecurityException
}
catch (IOException e) {
    // Gestion des IOException et des sous-classes de IOException
}
catch (Exception e){
    // Gestion de toutes les autres exceptions
}
  
```

## Exceptions

POO en Java

## Les exceptions (2)

- Ce sont des instances de classes dérivant de `java.lang.Exception`
- La levée d'une exception provoque une remontée dans l'appel des méthodes jusqu'à ce qu'un bloc `catch` acceptant cette exception soit trouvé. Si aucun bloc `catch` n'est trouvé, l'exception est capturée par l'interpréteur et le programme s'arrête.
- L'appel à une méthode pouvant lever une exception doit :
  - soit être contenu dans un bloc `try/catch`
  - soit être situé dans une méthode propageant (`throws`) cette classe d'exception
- Un bloc (optionnel) `finally` peut-être posé à la suite des `catch`. Son contenu est exécuté après un `catch` ou après un `break`, un `continue` ou un `return` dans le bloc `try`

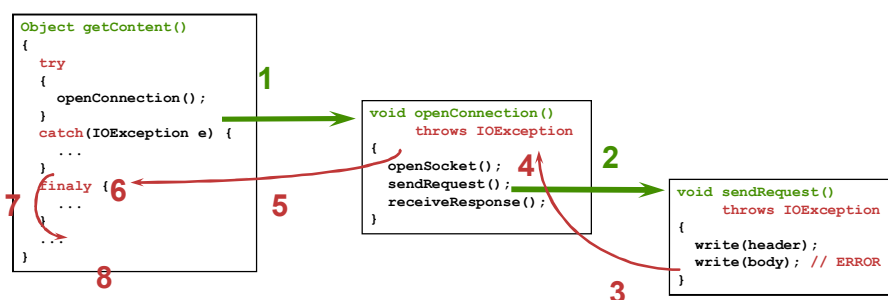
M.Romdhani, Mars 2019

35

## Exceptions

POO en Java

## Les exceptions (3)



M.Romdhani, Mars 2019

36

## Instruction "try-with-resource"

- L'instruction "try-with-resource" de Java 7 permet de gérer automatiquement la fermeture des ressources (readers, writers, sockets, connexions ...).
- Adieu le bloc "finally" et surtout adieu les bugs liés aux oublis d'appels de la méthode "close()".

```
public static void main(String args[]) {
    String filepath = "src/demo/trywithresource/mytext.txt";
    try (BufferedReader reader = new BufferedReader(new FileReader(filepath))) {
        String line = reader.readLine();
        while (line != null) {
            System.out.println(line);
            line = reader.readLine();
        }
    } catch (IOException e) {
        System.err.println("Erreur : " + e.getMessage());
    }
}
```

## Multiple Catch handling

- L'intérêt majeur est - comme le montre cet exemple - d'éviter de dupliquer inutilement du code en factorisant les gestions d'erreurs similaires.

```
try {
    Class.forName("org.hsqldb.jdbcDriver");
    Connection conn = DriverManager.getConnection("jdbc:hsqldb:mem:testdb", "sa", "");
    Statement stmt = conn.createStatement();
    stmt.execute("CREATE TABLE TMP (DATA VARCHAR(255))");
} catch (ClassNotFoundException | SQLException e) {
    System.err.println("Error during database access : " + e.getMessage());
} catch (Exception e) {
    System.err.println("Unexpected error ! Please contact the technical support");
} finally {
    // close resources OR use try-with-statement !
}
```

## Le framework de collections

### Les collections d'objet

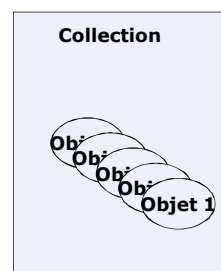
POO en Java

#### ■ Opérations de base

- Ajout d'un objet
- Suppression d'un objet
- Récupération d'un objet

#### ■ Types de collections courantes

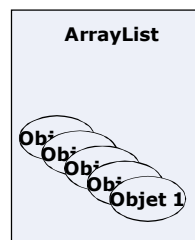
- Les vecteurs
- Les listes
- Les ensembles
- Les tables de hashage
- Les piles



## Les collections d'objet

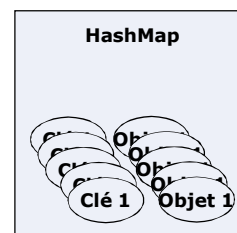
### ■ Classe ArrayList

- Les éléments sont des objets
- Doublons autorisés
- Récupération par numéro d'index ou valeur de l'objet



### ■ Classe HashMap

- Paires clé/valeur
- Les clés sont des objets
- Récupération par la valeur de la clé ou par la valeur de l'objet



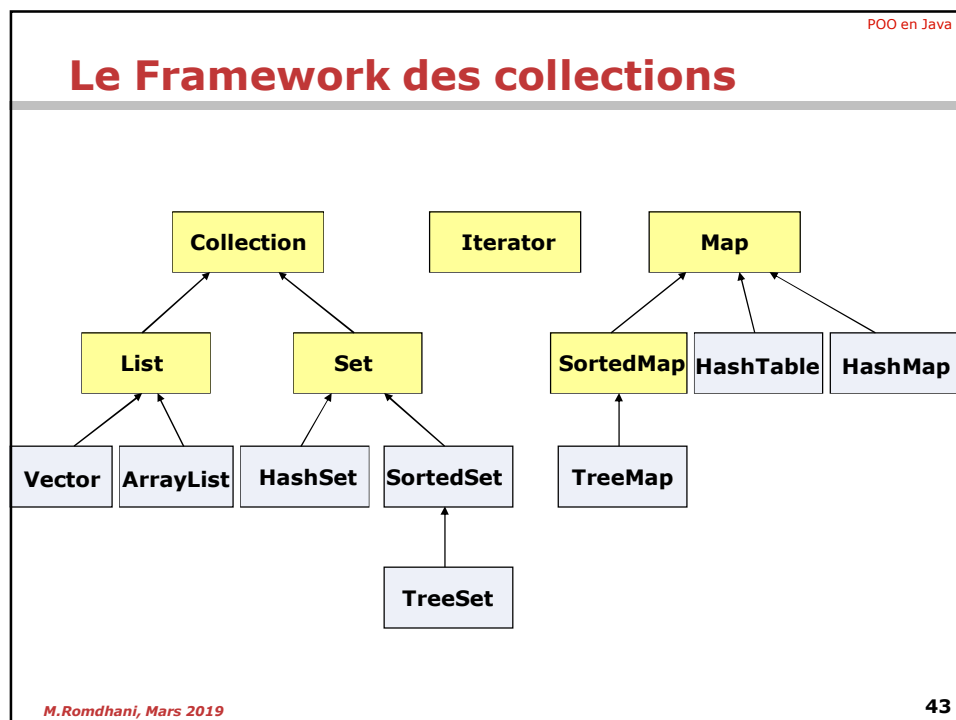
## Java Collections Framework

### ■ Collection

- Object that groups multiple **elements** into one unit
- Also called container

### ■ Collection framework consists of

- Interfaces
  - Abstract data type
- Implementations
  - Reusable data structures
- Algorithms
  - Reusable functionality



POO en Java

## Collections génériques

- **Les collections sont génériques à partir du J2SDK 1.5**
  - Améliorer la qualité en évitant les problèmes d'Inboxing/Outboxing
  - Exemple
    - List doit être appelée avec un paramètre de type : **List<String>**
    - L'itérateur doit aussi être générique et instancié avec le même type des objets de la collection **Iterator <String>**
- **Il est possible que l'utilisateur puisse déclarer des Collections génériques**

```

public class Box <T> {
    private List <T> contents;    ...
}

```
- **Une structure algorithmique for a été rajoutée pour faciliter l'exploitation des collections. Elle est souvent appelée "enhanced for" !**
  - **for (type var : collection) {...}**
    - En remplacement à : **for (Iterator iter=c.iterator();iter.hasNext();){...}**
    - Ceci est similaire au foreach du C#.Net !!!

M.Romdhani, Mars 2019 44

## Nouveautés syntaxiques des versions Java 5/6/7/8

### Résumé des nouveautés Java SE 5

POO en Java

#### ■ Les nouveautés marquantes de Java SE 5 (Tiger)



##### ■ Les annotations

- Technique d'injection de déclarations pouvant influencer sur la compilation et/ou l'exécution des programmes

##### ■ La généricité (Generics)

- Sécurité du typage à la compilation

##### ■ Autoboxing/unboxing

- "Wrapping" automatique des primitives

#### ■ Les nouveautés moins importantes

- Les types énumérés
- Le nouveau énoncé for
- Les imports statiques
- Les arguments variables
- Le Scanner et le Formatter

## Résumé des nouveautés Java SE 6

### ■ Aucune nouveautés syntaxiques n'a été introduite dans Java SE 6 (Mustang)

- Toutefois l'API a connu les améliorations et extensions suivantes :
  - Integrated Web Services (JAX-WS).
  - Scripting Language Support
  - JDBC 4.0 API
  - Java Compiler API
  - Pluggable Annotations
  - Native PKI, Java GSS, Kerberos and LDAP support.



## Résumé des nouveautés Java SE 7

### ■ Nouveautés de Java SE 7 (Dolphin)

- **Strings in switch Statement**
- **Type Inference for Generic Instance Creation (Diamond operator <>)**
- **Try with Resources**
- **Multiple Catch Handling**
- Binary Literals, underscore in literals
- Support for Dynamic Languages
- Java nio Package





## Résumé des nouveautés Java SE 8

- **Nouveautés de Java SE 8**
  - **Lambda Expressions**
  - **Pipelines and Streams**
  - **Date and Time API**
  - **Default Methods**
  - **Type Annotations**
  - **Nashhorn JavaScript Engine**
  - **Concurrent Accumulators**
  - **Parallel operations**