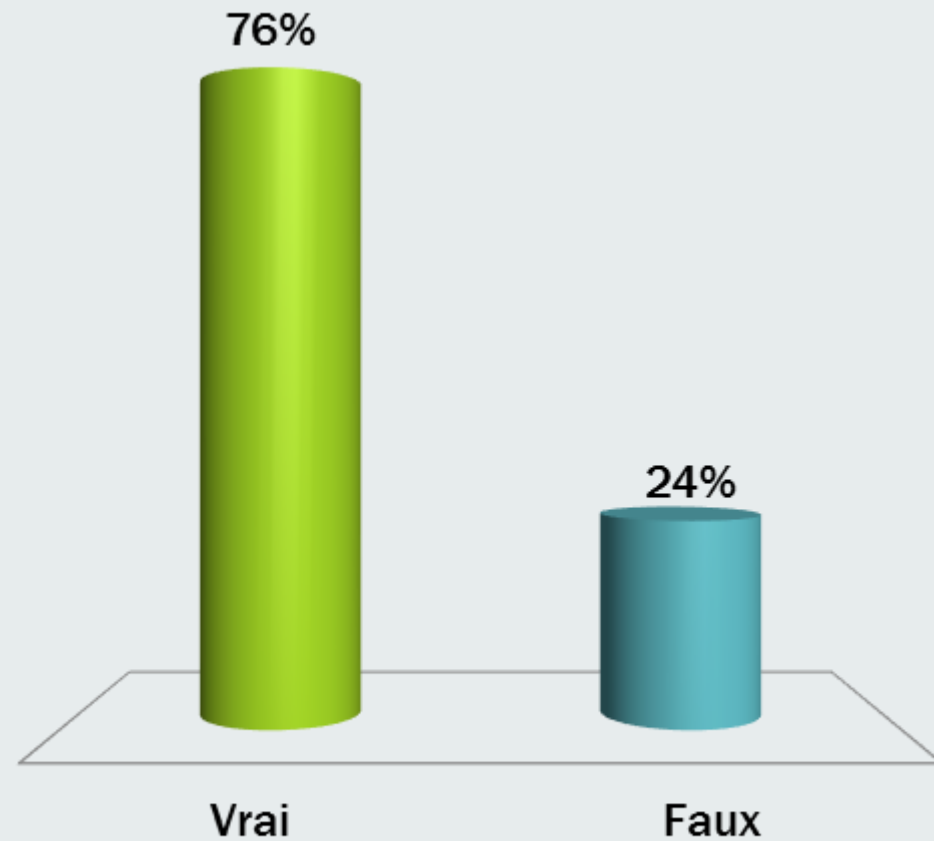


# JAVA / UML

# BOITIER SUR LE CANAL 41

- A. Vrai
- B. Faux

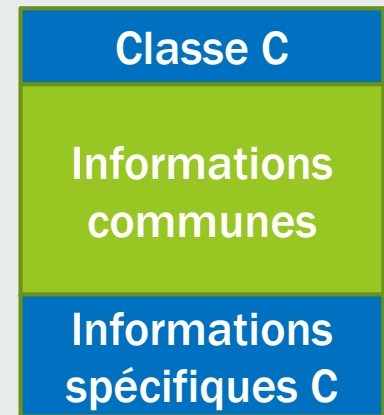


# DIAGRAMME DE CLASSE

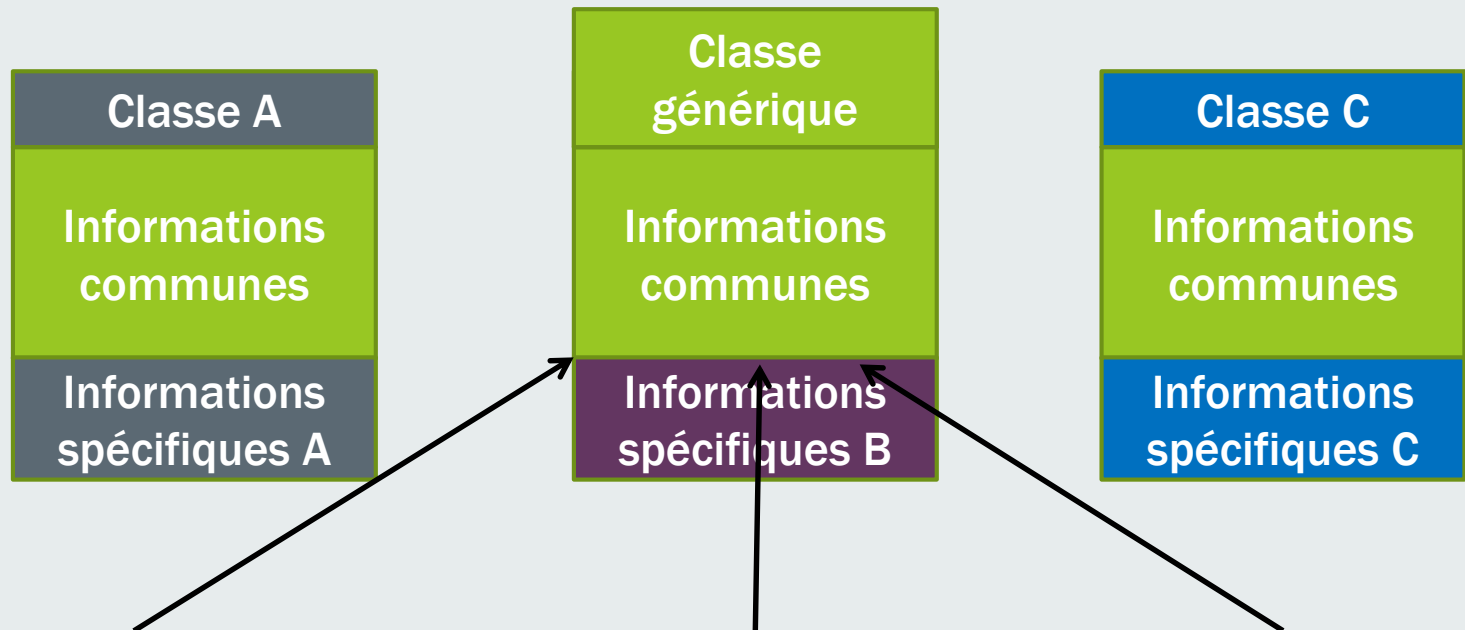
# RELATION D'HÉRITAGE

- Utilisée lorsque plusieurs classes ont des propriétés communes
- Permet de rationaliser le code

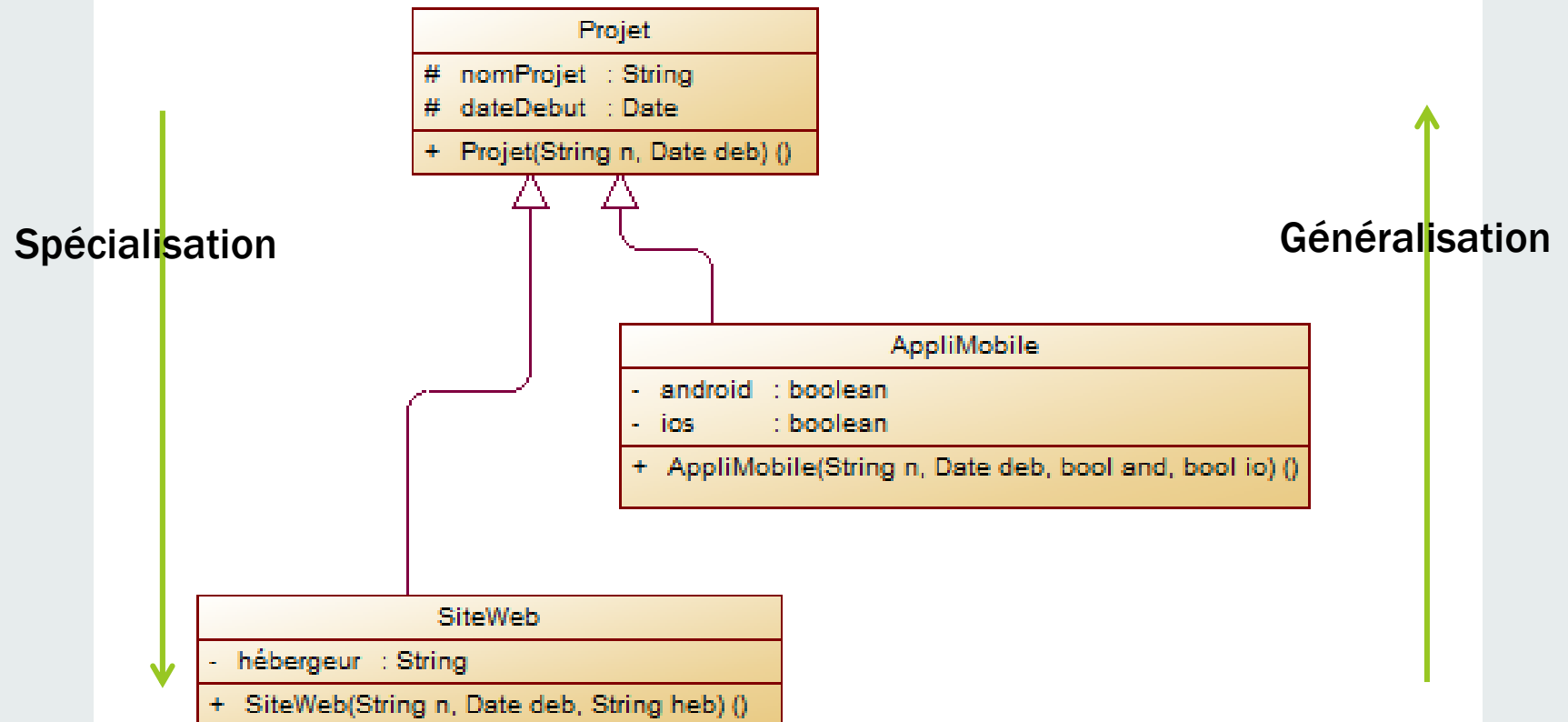
# ARCHITECTURE SANS HÉRITAGE



# ARCHITECTURE AVEC HÉRITAGE



# RELATION D'HÉRITAGE



# RELATION D'HÉRITAGE

## Spécialisation

- Une ou plusieurs classes sont des classes spécialisées
  - Hérite de la structure de la classe générique
  - Attributs et méthode spécifiques

## Généralisation

- Une classe contient les informations génériques ou informations communes



# RELATION D'HÉRITAGE

Classe mère

Classe générique

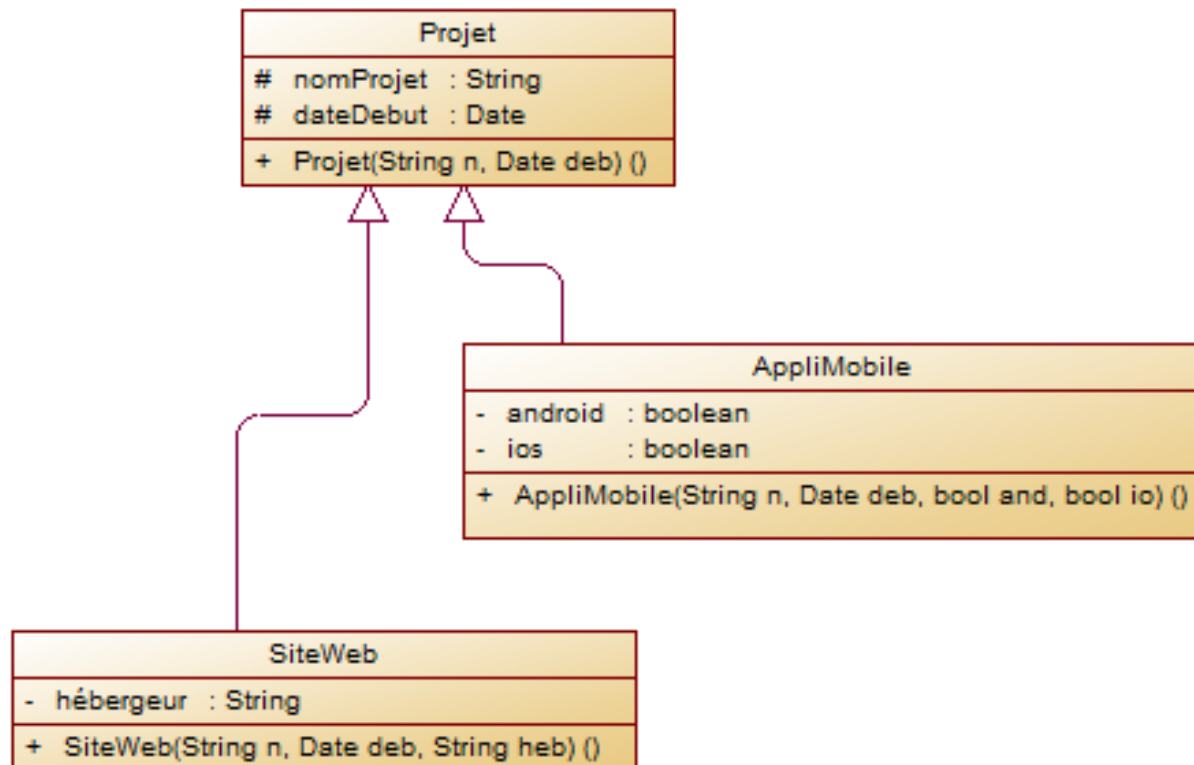
Super classe/  
Classe de base

Classe fille

Classe spécialisée

Classe dérivée

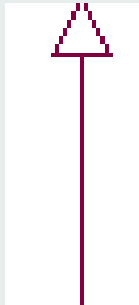
Sous classe



# RELATION D'HÉRITAGE

## UML

- Représenté par

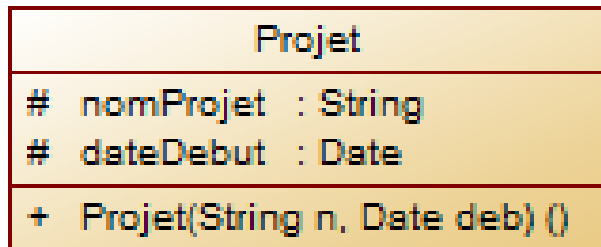


## JAVA

- Définition de la relation d'héritage lors de création des sous-classes : `extends`

# RELATION D'HÉRITAGE/CLASSE MÈRE

## UML



## JAVA

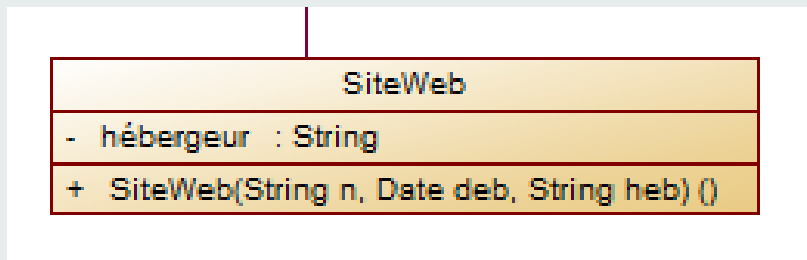
### ■ Attributs communs

### ■ Méthodes

- Définition dans la classe mère
- Peuvent être redéfinies dans les classes filles

# RELATION D'HÉRITAGE /CLASSE FILLE

## UML



## JAVA

- Extends pour définir la relation d'héritage avec la classe mère
- Hérite automatiquement des attributs protégés et des méthodes de la classe mère

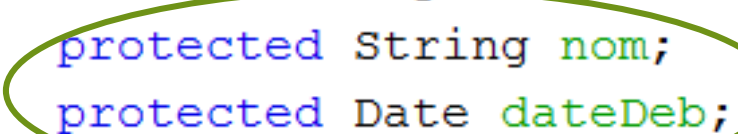
# DANS LA CLASSE FILLE

- Doit faire appel au constructeur de la classe mère pour initialiser les attributs hérités (mot clé super en Java, base en C#)
- A accès aux attributs hérités comme à ses attributs locaux dans ses méthodes
- Peut appeler les méthodes de la classe mère (mot clé super en Java, base en C#)
- Peut redéfinir les méthodes de la classe mère
- Peut avoir ses propres attributs et méthodes (non accessibles par la classe mère)

# CRÉATION DES CLASSES

## Projet

```
public class Projet {  
    protected String nom;  
    protected Date dateDeb;  
}
```



## Site web

```
public class SiteWeb extends Projet{  
    private String hébergeur;  
}
```

# LES CONSTRUCTEURS

## Projet

```
public Projet(String n, Date deb)
{
    nom=n;
    dateDeb=deb;
}
```

## Site Web

- Doit initialiser tous ses attributs (hérités et locaux)

```
public SiteWeb(String n, Date deb, String heb)
{
    //appel du constructeur de la classe mère
    //pour initialiser les attributs hérités
    super(n, deb);
    //initialisation des attributs de la classe fille
   hebergeur = heb;
}
```

# LES MÉTHODES

## Projet

```
public String toString()  
{  
    return nom + " début "+dateDeb.toString();  
}
```

## SiteWeb

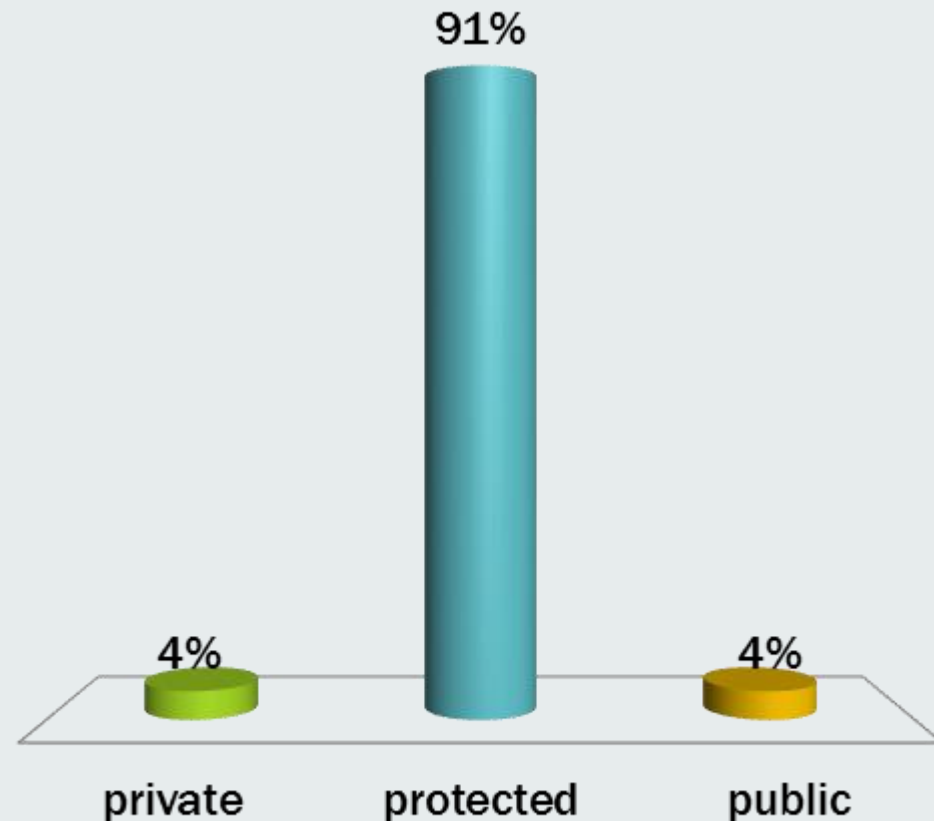
```
public String toString()  
{  
    return nom + " début "+dateDeb.toString()+  
        "hébergeur "+hebergeur;  
}
```

```
public String toString()  
{  
    //appel de la méthode toString de la classe mère  
    String info = super.toString();  
    //ajout des infos de la classe fille  
    return info + " hébergeur "+hebergeur;  
}
```



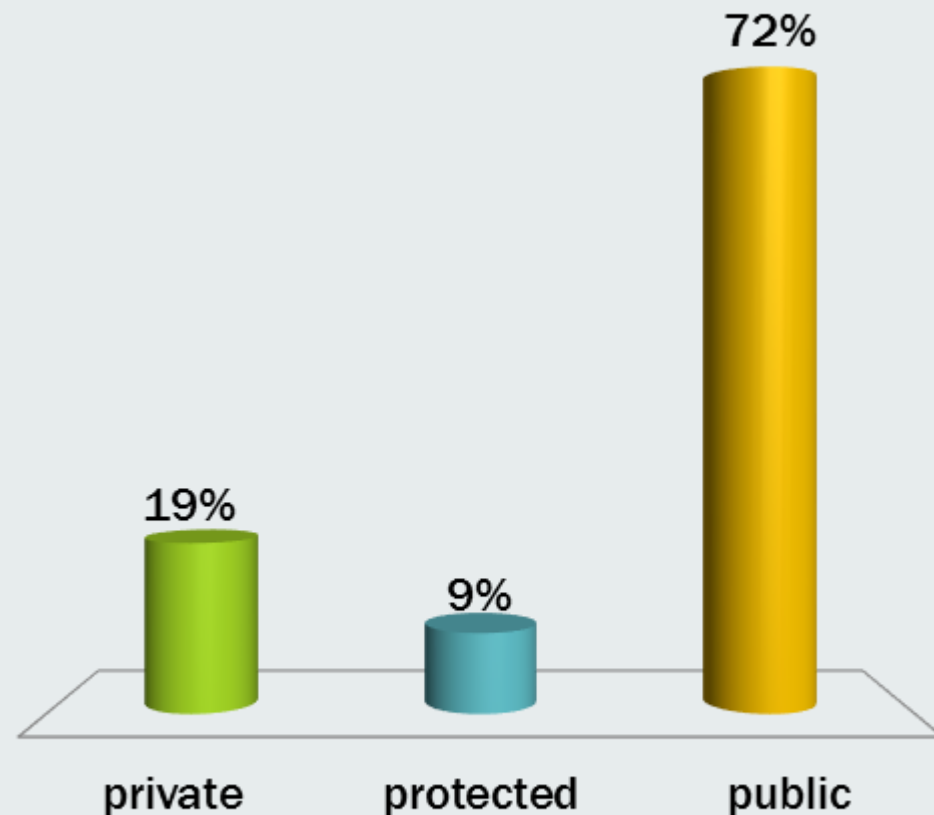
# QUELLE DOIT-ÊTRE LA PORTÉE DES ATTRIBUTS DANS LA CLASSE MÈRE ?

- A. private
- B. protected
- C. public



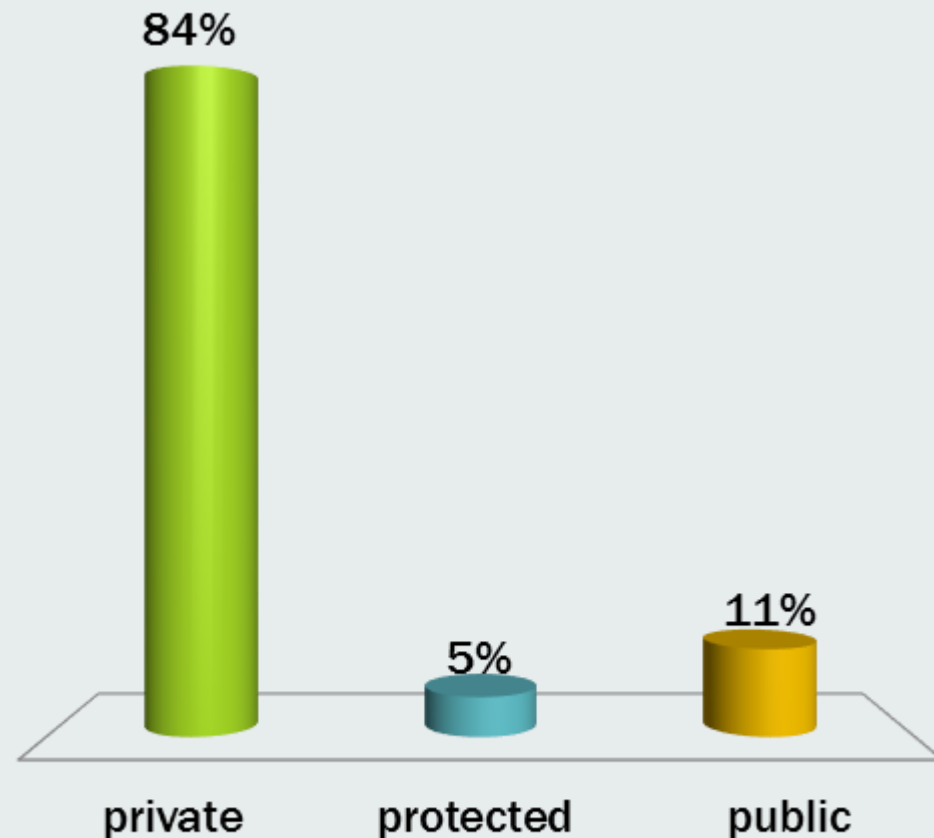
# QUELLE DOIT-ÊTRE LA PORTÉE DES MÉTHODES DANS LA CLASSE MÈRE ?

- A. private
- B. protected
- C. public



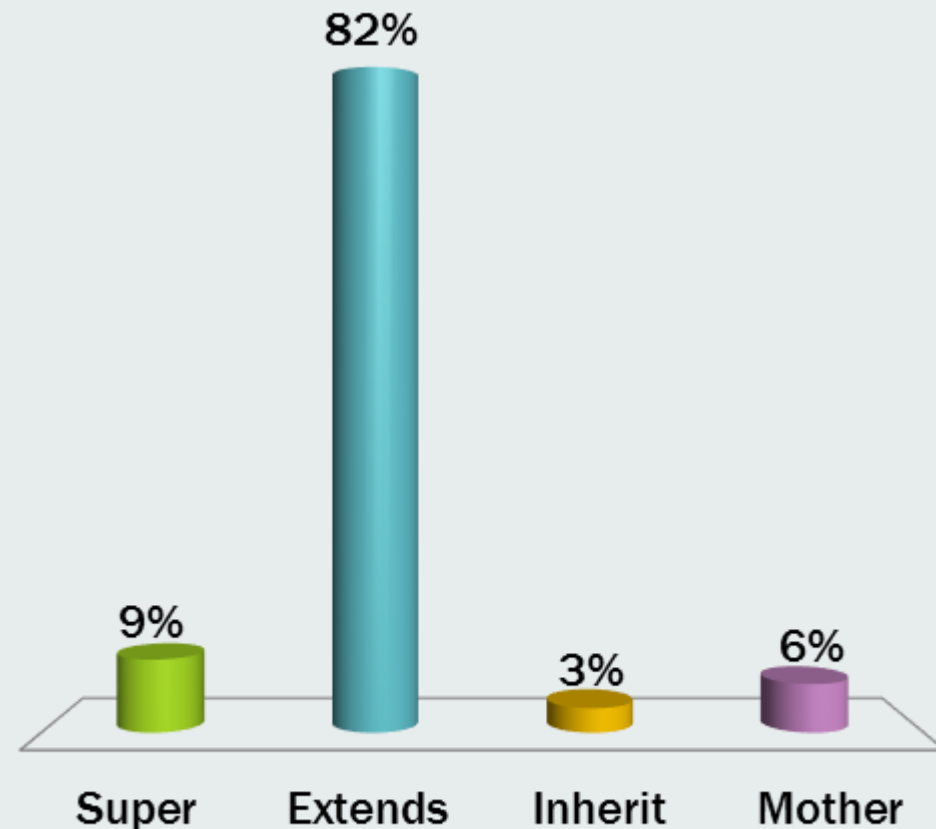
# QUELLE DOIT-ÊTRE LA PORTÉE DES ATTRIBUTS DANS LA CLASSE FILLE ?

- A. private
- B. protected
- C. public



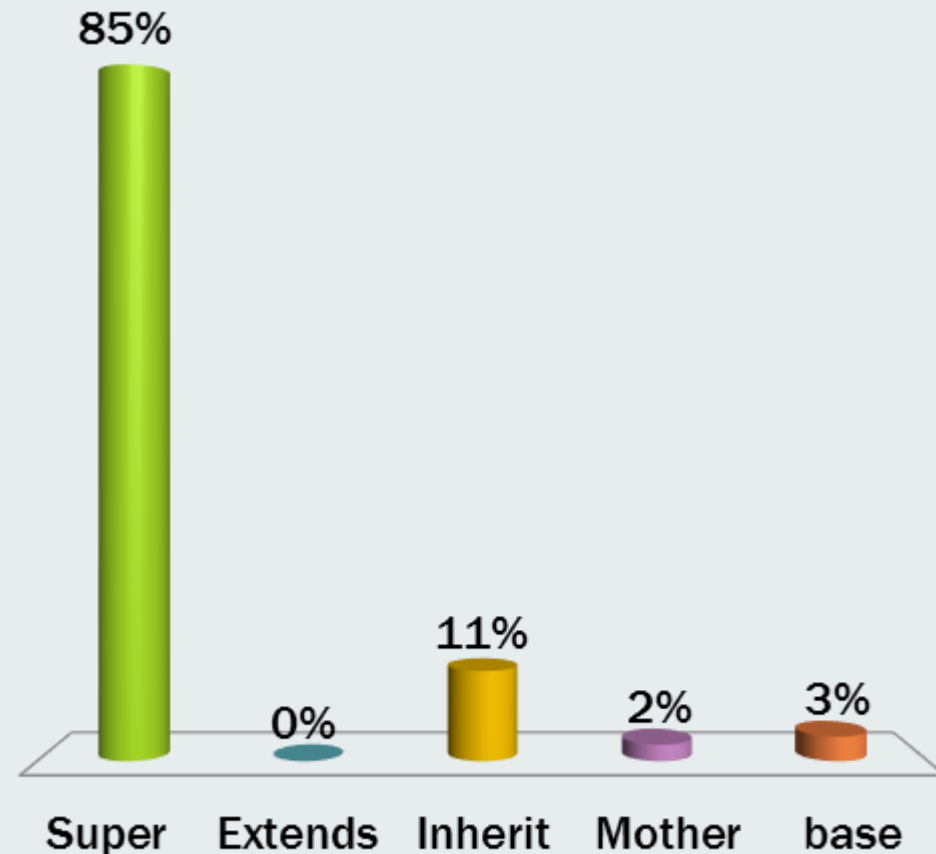
# QUEL MOT CLÉ EST UTILISÉ POUR DÉFINIR LA RELATION D'HÉRITAGE ?

- A. Super
- B. Extends
- C. Inherit
- D. Mother



# QUEL MOT CLÉ EST UTILISÉ EN JAVA POUR ACCÉDER À UNE MÉTHODE DE LA CLASSE MÈRE (DEPUIS UNE CLASSE FILLE)

- A. Super
- B. Extends
- C. Inherit
- D. Mother
- E. base

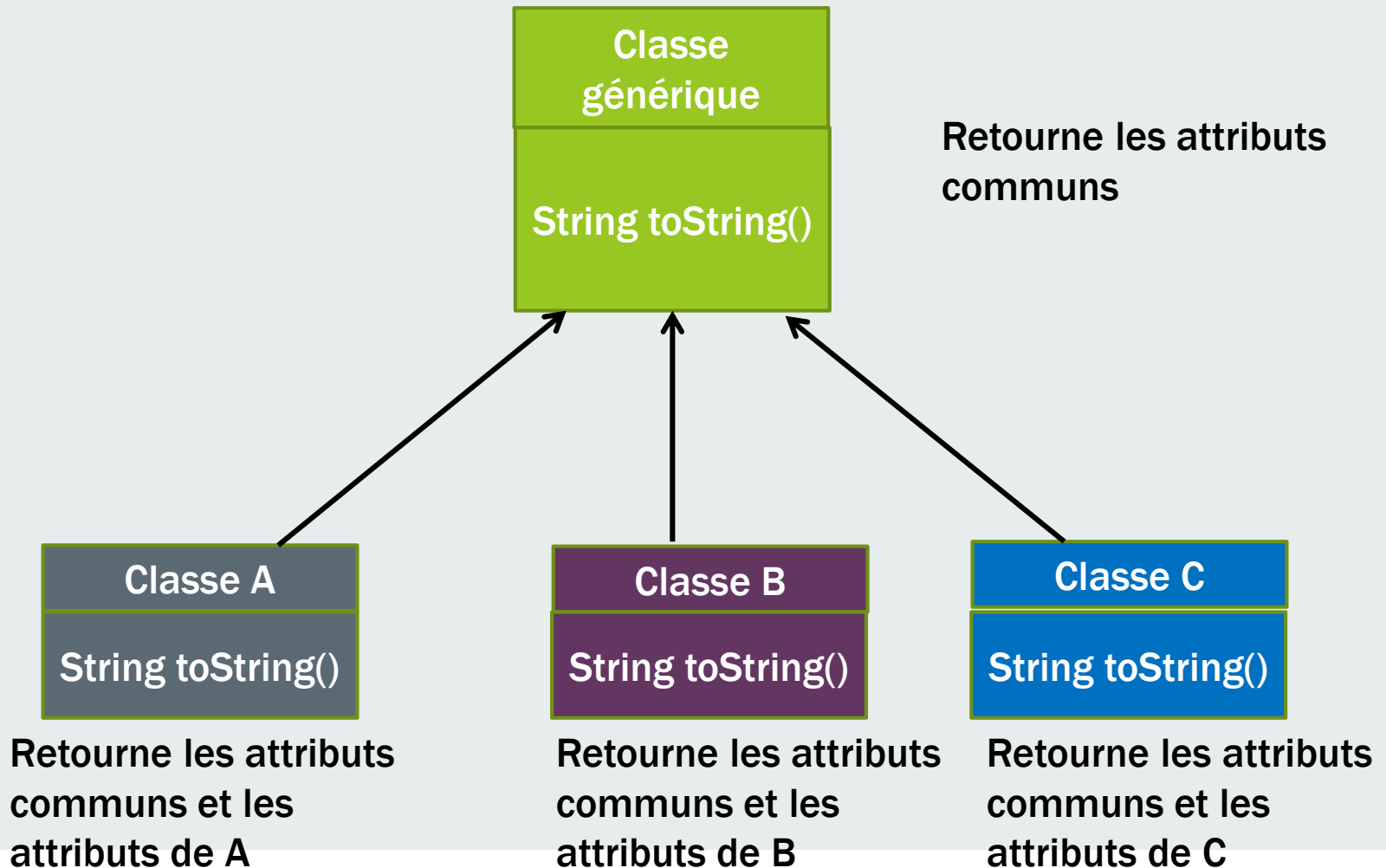


# LE POLYMORPHISME

# PRINCIPE DU POLYMORPHISME

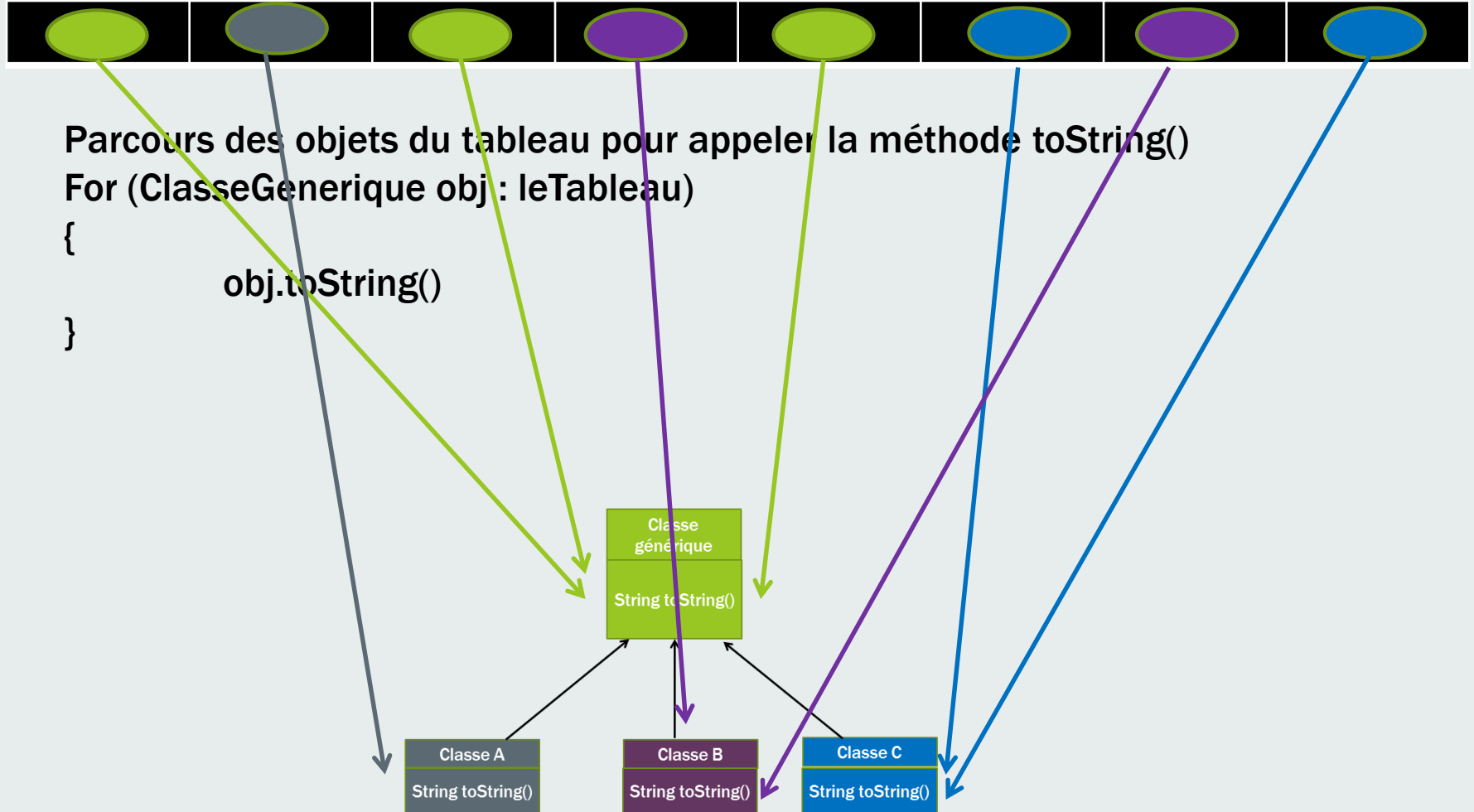
- Polymorphisme = Plusieurs formes
- Une méthode (même nom, même type de retour, même paramètres) peut exécuter du code différents dans des classes différentes

# ARCHITECTURE AVEC HÉRITAGE





# MISE EN ŒUVRE DU POLYMORPHISME



# CAS PARTICULIER

- Pour interdire la redéfinition d'une méthode par une classe fille, on utilise le mot clé `final` lors de la création de la méthode dans la classe mère

- Exemple dans Classe Générique

```
Public final void NePeutPasEtreRedefinie()  
{...}
```

# SURCHARGE / REDEFINITION

## Surcharge

- Création d'une méthode avec le même nom mais des paramètres différents

## Redéfinition

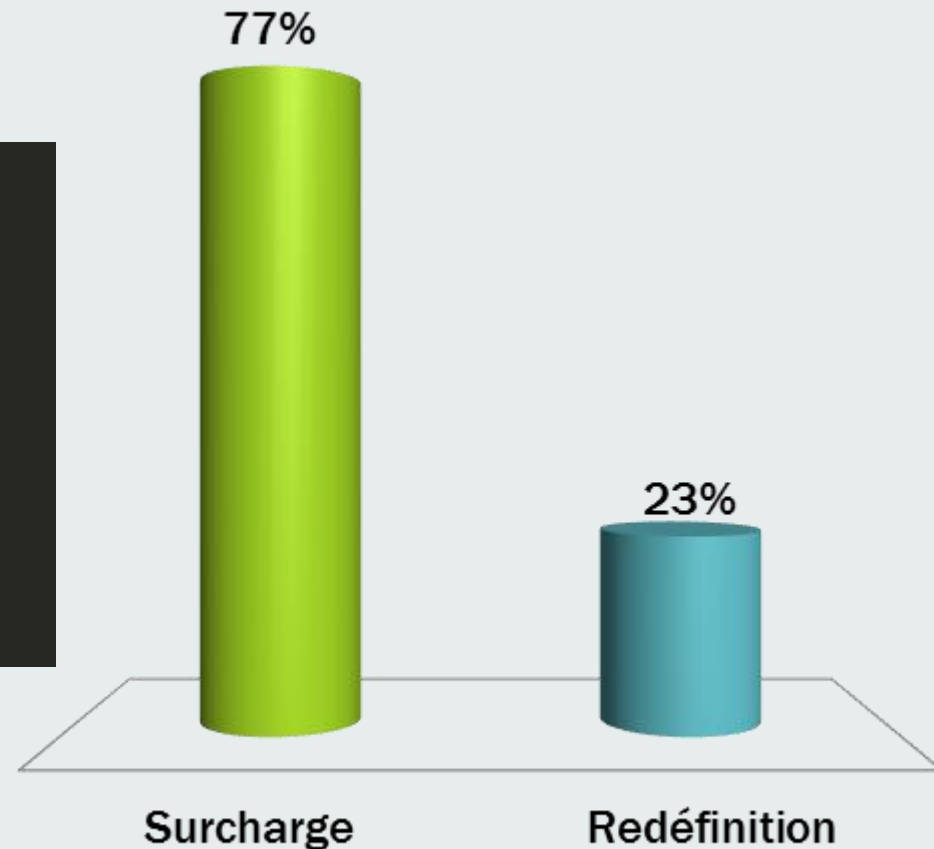
- Création dans une classe fille d'une méthode avec la même signature que la classe mère (même nom, même paramètres) mais un code différent

# EXEMPLE

A. Surcharge

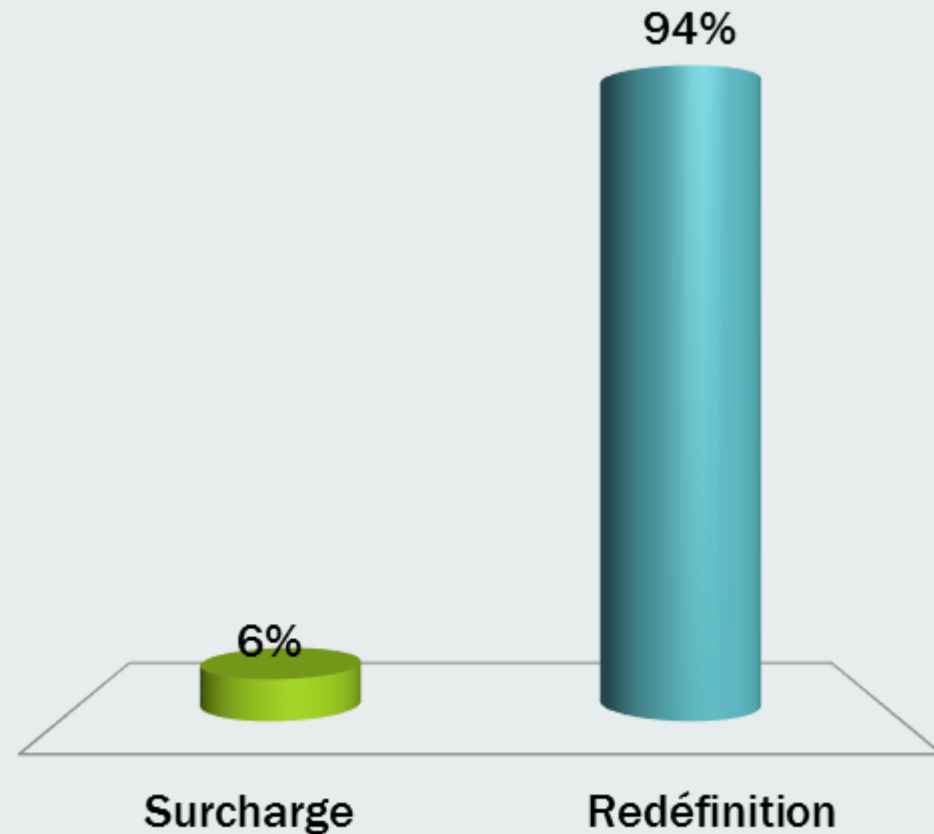
B. Redéfinition

```
1 static void parcourirTableau(String[] tab)
2 {
3     for(String str : tab)
4         System.out.println(str);
5 }
6
7 static void parcourirTableau(int[] tab)
8 {
9     for(int str : tab)
10         System.out.println(str);
11 }
```



# EXEMPLE : MÉTHODE TOSTRING()

- A. Surcharge
- B. Redéfinition



# LE FRAMEWORK JAVA

# FRAMEWORK

## Définition

- Ensemble de composants logiciels de base
  - Bibliothèque de classes/fonctions
  - Principes d'architecture, d'organisation

## Exemple

- .NET (Microsoft) : C#, VB.NET...
- J2SE (Oracle) : Java
- Symfony (Sensio Labs) : php

# AVANTAGES

- Gain de temps de développement
- Robustesse de l'application
- Homogénéité du code



# LA MACHINE VIRTUELLE JAVA

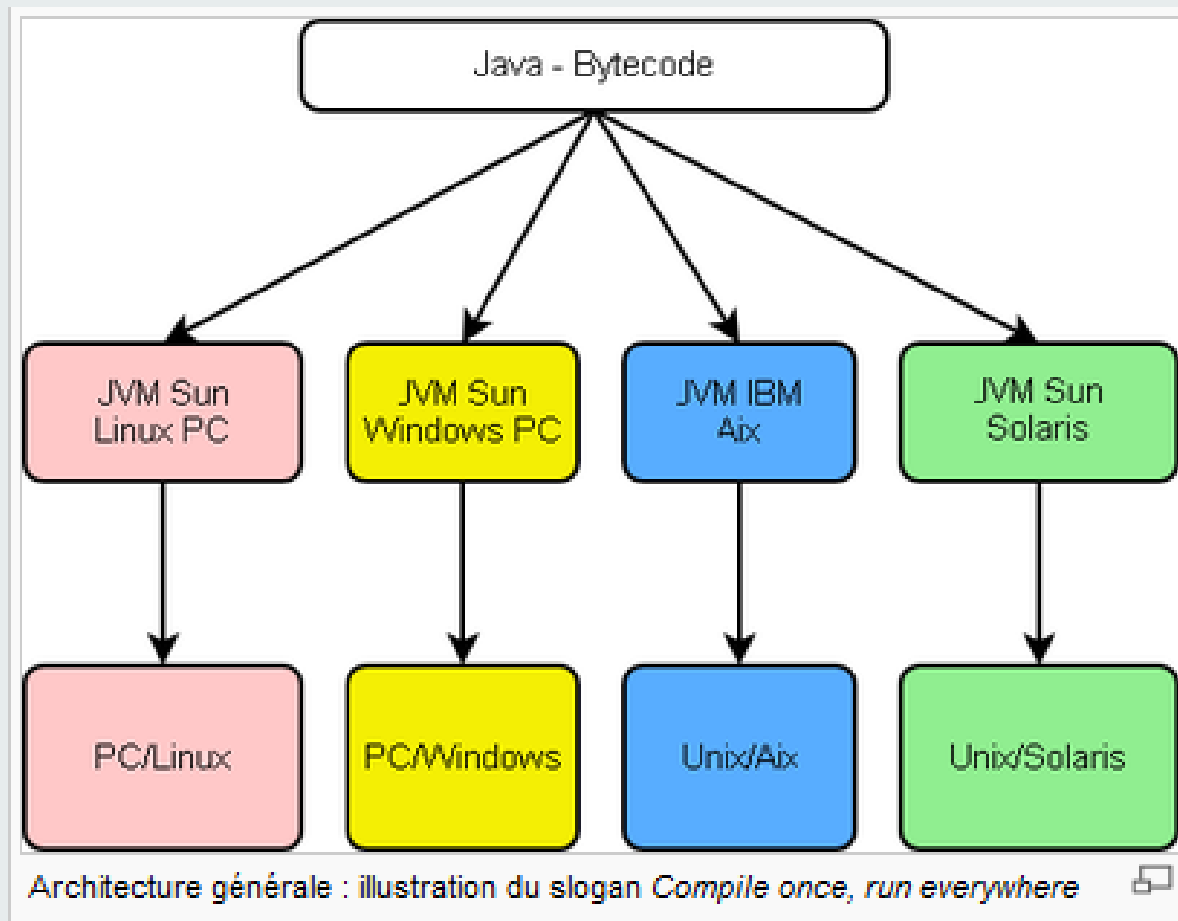
## Bytecode

- Programmes java sont compilés dans un langage intermédiaire : le bytecode
- Le bytecode (.jar) est exécuté par la machine virtuelle
- Permet une portabilité sur les différents OS

## Exécution

- Doit être installée sur les postes qui exécutent l'application
- La version installée doit correspondre à la version utilisée pour le développement

# LA MACHINE VIRTUELLE

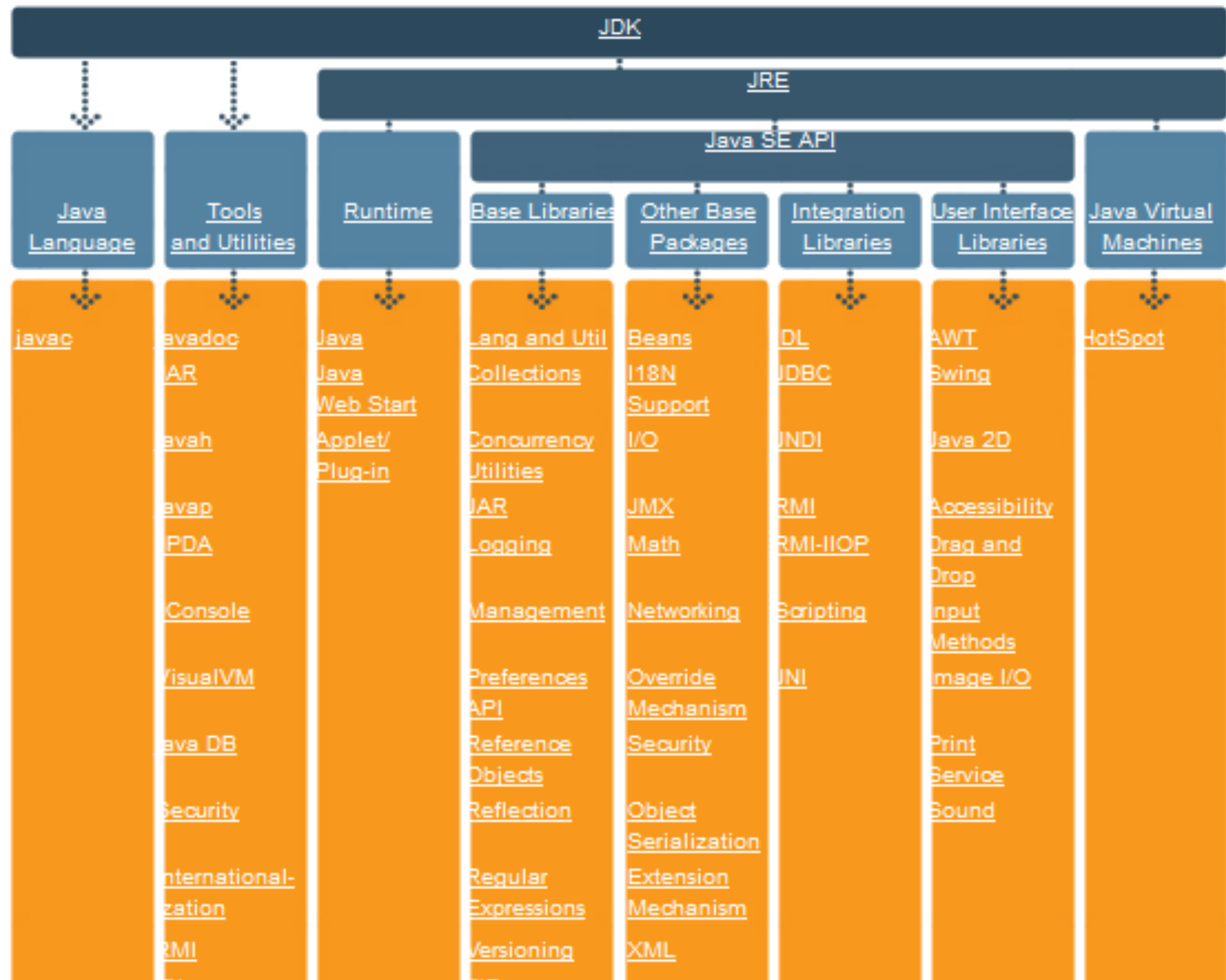


**JVM : Java Virtual Machine**

# GARBAGE COLLECTOR

- Permet de gérer automatiquement la mémoire
- Un objet créé alloue de la mémoire pour stocker ses propriétés
  - Difficultés pour gérer la destruction des objets
- Libération de la mémoire gérée par le garbage collector
  - Mémoire libérée dès qu'un objet n'a plus aucune référence

## Java SE Platform at a Glance



# EXEMPLE

- On souhaite créer une liste d'objets
  - Package `java.util`
    - Interface `List`
    - Interface `Collection`
    - Class `ArrayList`
    - Class `Vector`
    - Class `Collections...`

# FRAMEWORK JAVA

- **Classe Object**

- Classe Racine
- Héritée par défaut par toutes les classes
- Définit des méthodes standards que chaque classe peut implémenter

# CLASSE OBJECT

protected Object	<code>clone()</code> Creates and returns a copy of this object.
boolean	<code>equals(Object obj)</code> Indicates whether some other object is "equal to" this one.
protected void	<code>finalize()</code> Called by the garbage collector on an object when garbage collection determines that there are no more references to the object.
<code>Class&lt;?&gt;</code>	<code>getClass()</code> Returns the runtime class of this Object.
int	<code>hashCode()</code> Returns a hash code value for the object.
void	<code>notify()</code> Wakes up a single thread that is waiting on this object's monitor.
void	<code>notifyAll()</code> Wakes up all threads that are waiting on this object's monitor.
String	<code>toString()</code> Returns a string representation of the object.
void	<code>wait()</code> Causes the current thread to wait until another thread invokes the <code>notify()</code> method or the <code>notifyAll()</code> method.
void	<code>wait(long timeout)</code> Causes the current thread to wait until either another thread invokes the <code>notify()</code> method or the <code>notifyAll()</code> method, or a certain amount of real time has elapsed.
void	<code>wait(long timeout, int nanos)</code> Causes the current thread to wait until another thread invokes the <code>notify()</code> method or the <code>notifyAll()</code> method, or a certain amount of real time has elapsed.

# EQUALS

- Test l'égalité de deux objets (même classe)
  - This
  - Objet passé en paramètre



# PRINCIPE DU CLONAGE

## Attributs de classes immuables

- Copie par valeur faite par l'appel de `super.clone()`
- Copie de surface (shallow copy)

## Attributs de classes non immuables

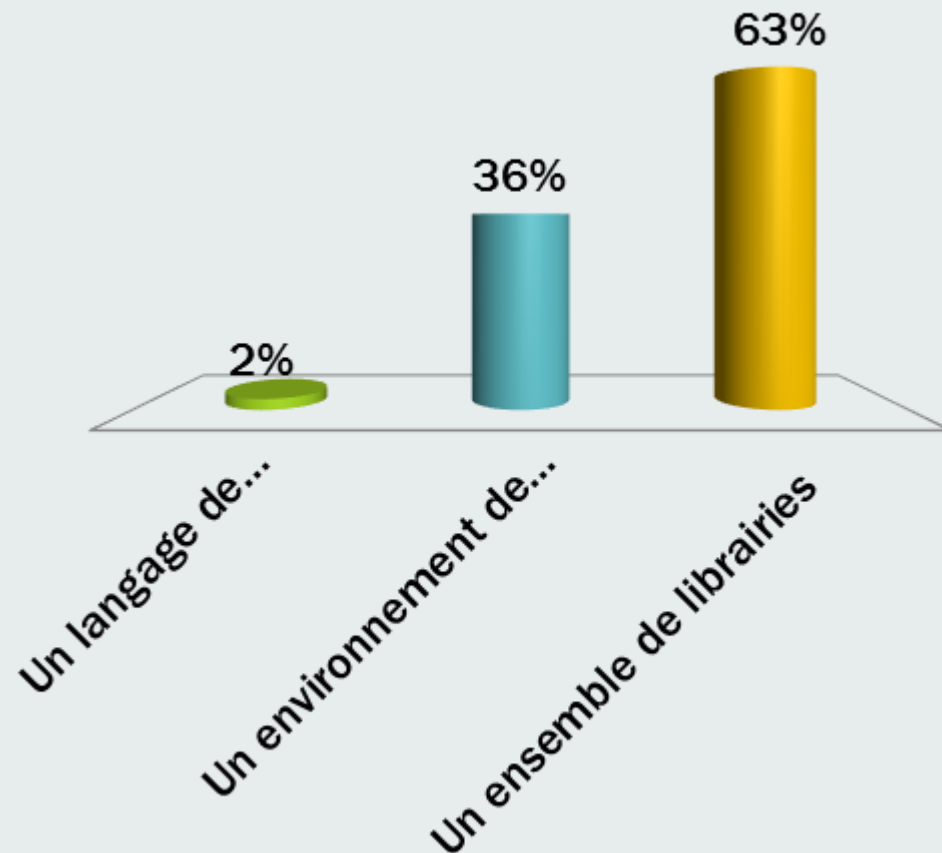
- Copie par valeur à ajouter dans la méthode `clone`
- Copie en profondeur (deep copy)

# PRINCIPES DU CLONAGE

- 2 références d'objet différentes
  - `x.clone() != x` doit renvoyer `true`
- Classes identiques
  - `x.clone().getClass() == x.getClass()` renvoie `true` par convention
- 2 objets identiques
  - `x.clone().equals(x)` renvoie `true` par convention

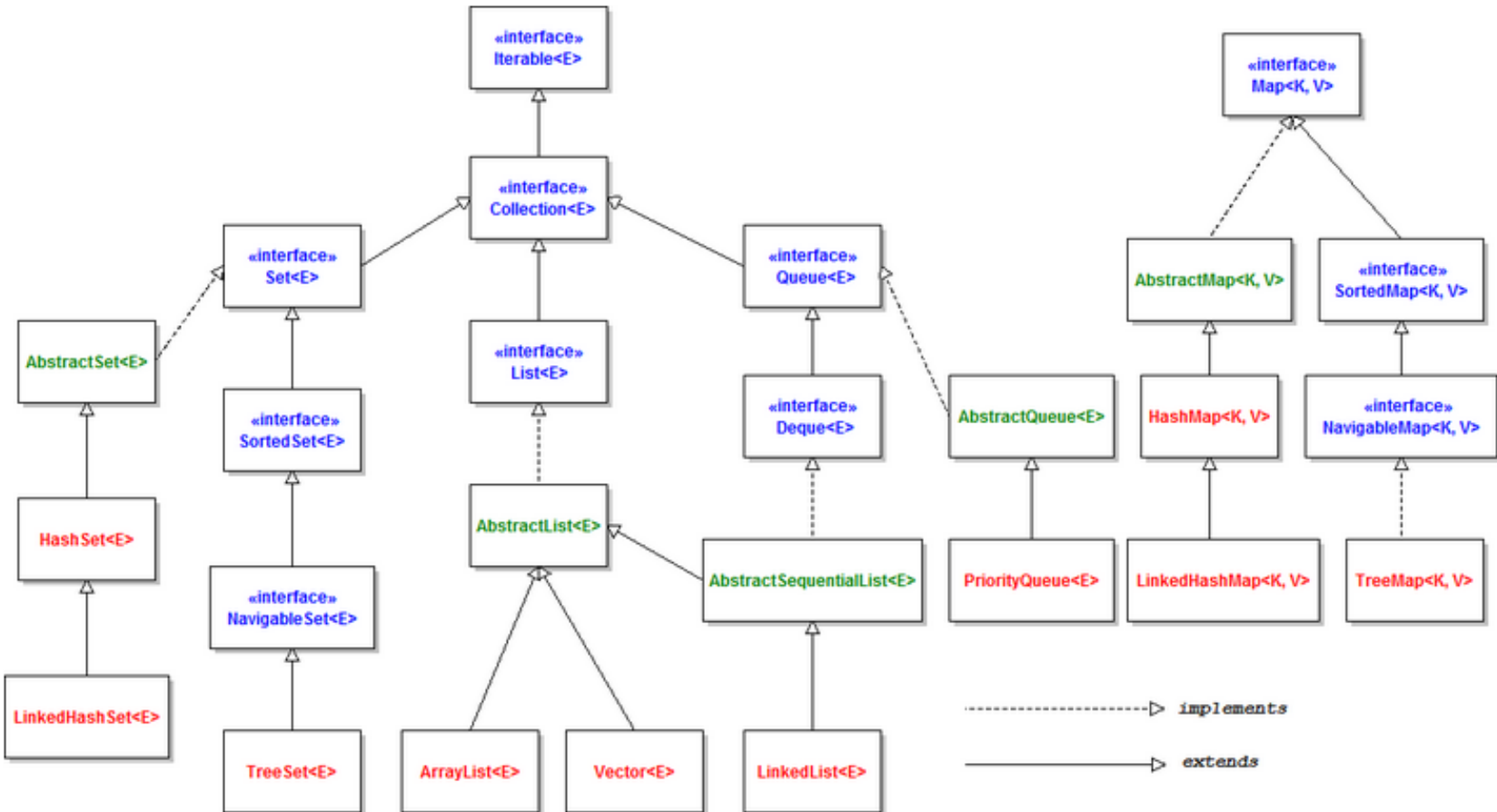
# QU'EST-CE QU'UN FRAMEWORK

- A. Un langage de développement
- B. Un environnement de développement
- C. Un ensemble de librairies



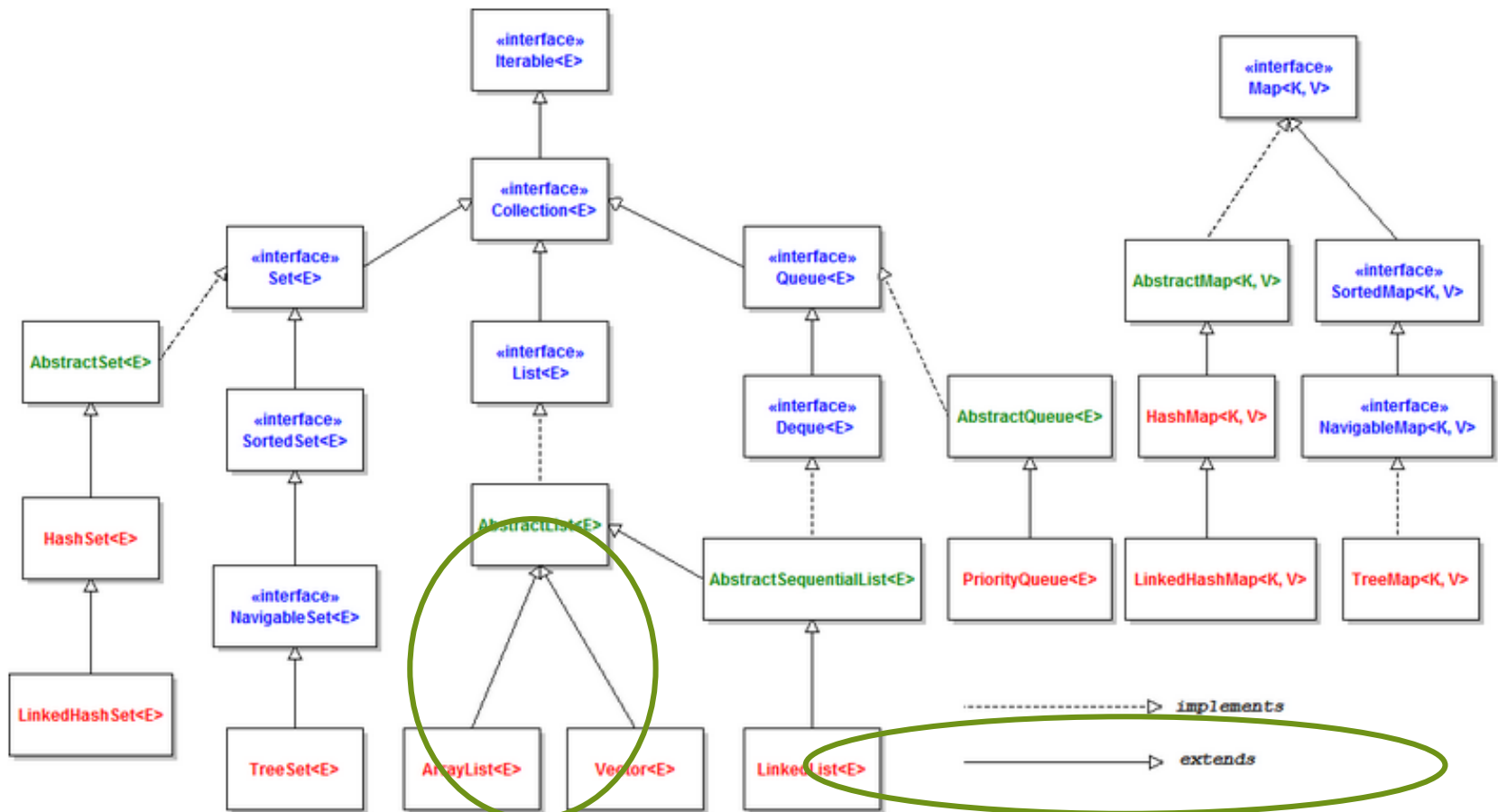
# STRUCTURATION DE CLASSES

# STRUCTURATION JAVA



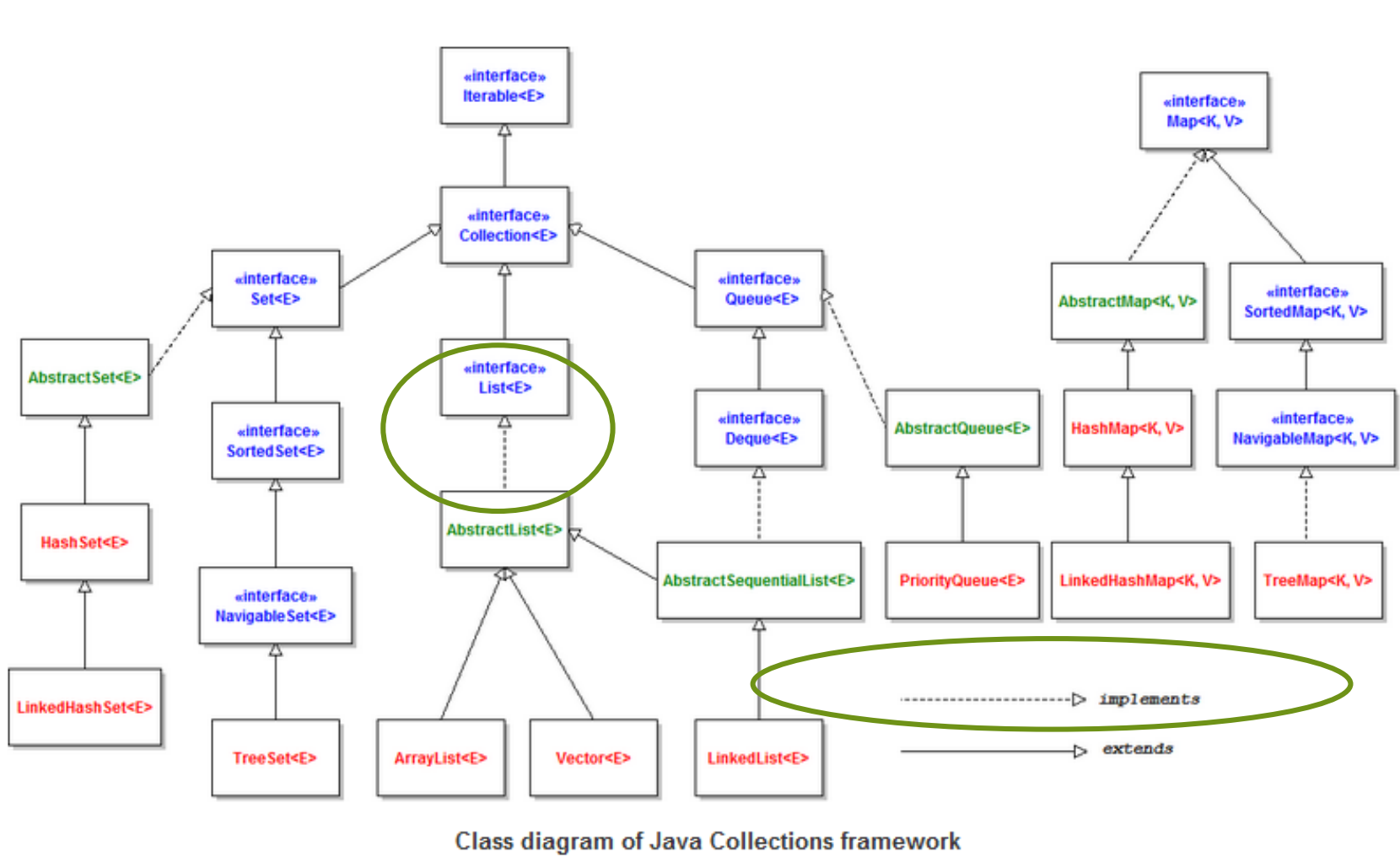
## Class diagram of Java Collections framework

# HÉRITAGE



Class diagram of Java Collections framework

## IMPLEMENTATION



## Class diagram of Java Collections framework

# HÉRITAGE/IMPLÉMENTATION

## Héritage

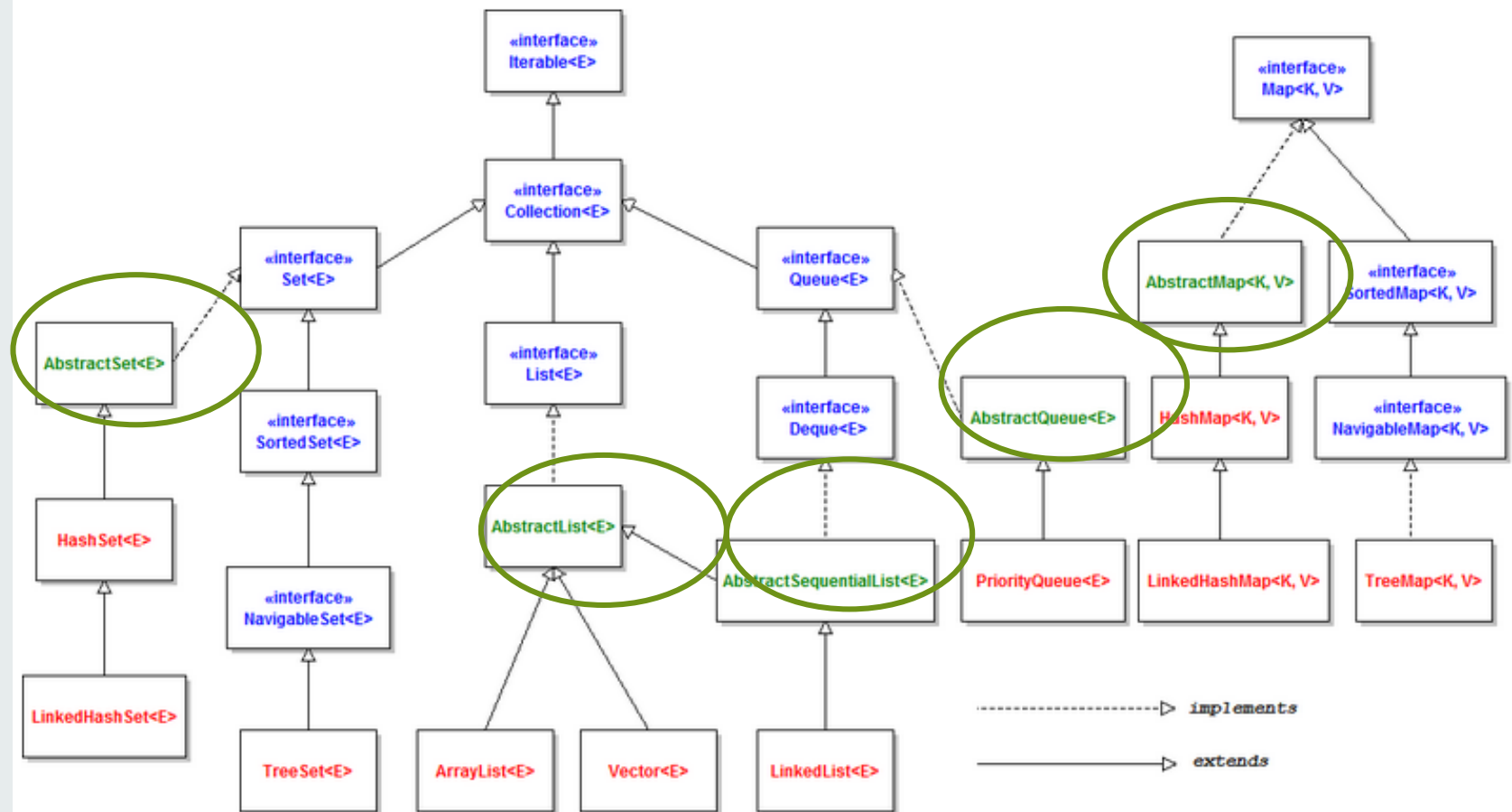
- La classe fille hérite des attributs `protected` et des méthodes de la classe mère
- Une classe ne peut héritée que d'une classe

## Implémentation

- La classe dérivée à l'obligation d'implémenter les méthodes de la classe mère
- Une classe peut implémenter plusieurs classes

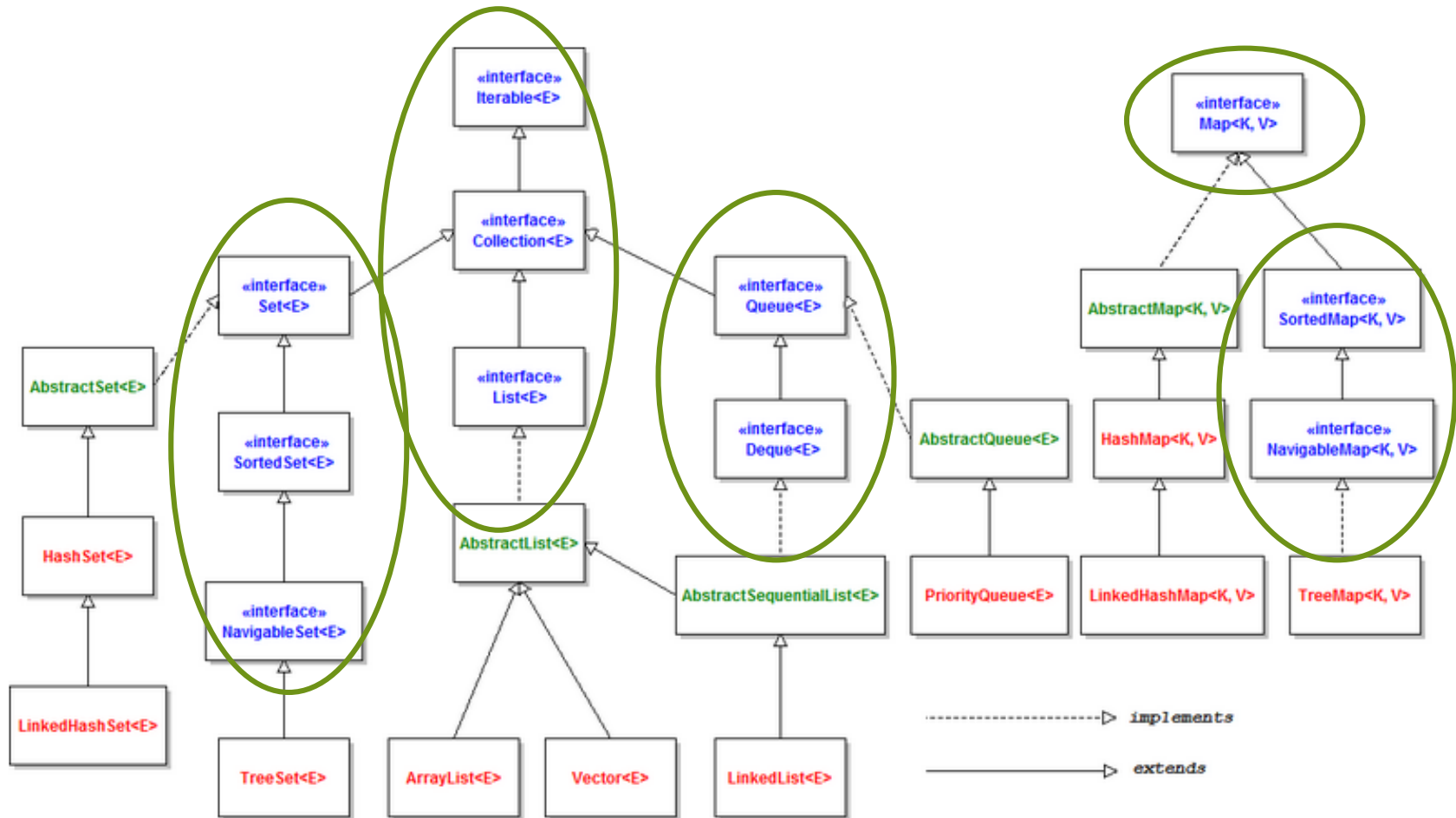


# CLASSE ABSTRAITE



Class diagram of Java Collections framework

# INTERFACE



Class diagram of Java Collections framework

**CLASSE ABSTRAITE**

# CLASSE ABSTRAITE

## Classe abstraite

- Contient au moins une méthode abstraite

## Méthode abstraite

- Mot clé `abstract`
- N'a pas de corps dans la classe mère
- Doit être redéfinie dans toutes les classes filles
  - Si une classe fille ne le fait pas, les méthodes concernées et la classe seront elles-mêmes abstraites

# PROGRAMMATION JAVA

## Classe générique

```
public abstract
ClasseGenerique{
//attributs
//constructeur
//méthodes
//méthode abstraite
public abstract void
calcule();
}
```

## Classe fille

```
public ClasseA extends
ClasseGenerique
{
//attributs
//constructeur
//méthodes
//méthode obligatoire
public void calcule ()
{
    //code obligatoire
}
}
```

# EXEMPLE CLASSE ABSTRAITE

- Les équipements sont des rouleaux, pinceaux, fusils
- On ne peut pas créer d'objet Equipement
- La classe Equipement est abstraite et définit la méthode colorier



**INTERFACE**

# INTERFACE

- Une interface ne contient que des méthodes abstraites
- Permet de définir une structure obligatoire des classes filles
- Les méthodes de l'interface sont par essence public et abstract (mot clés facultatifs dans la définition de la méthode)



# PROGRAMMATION JAVA

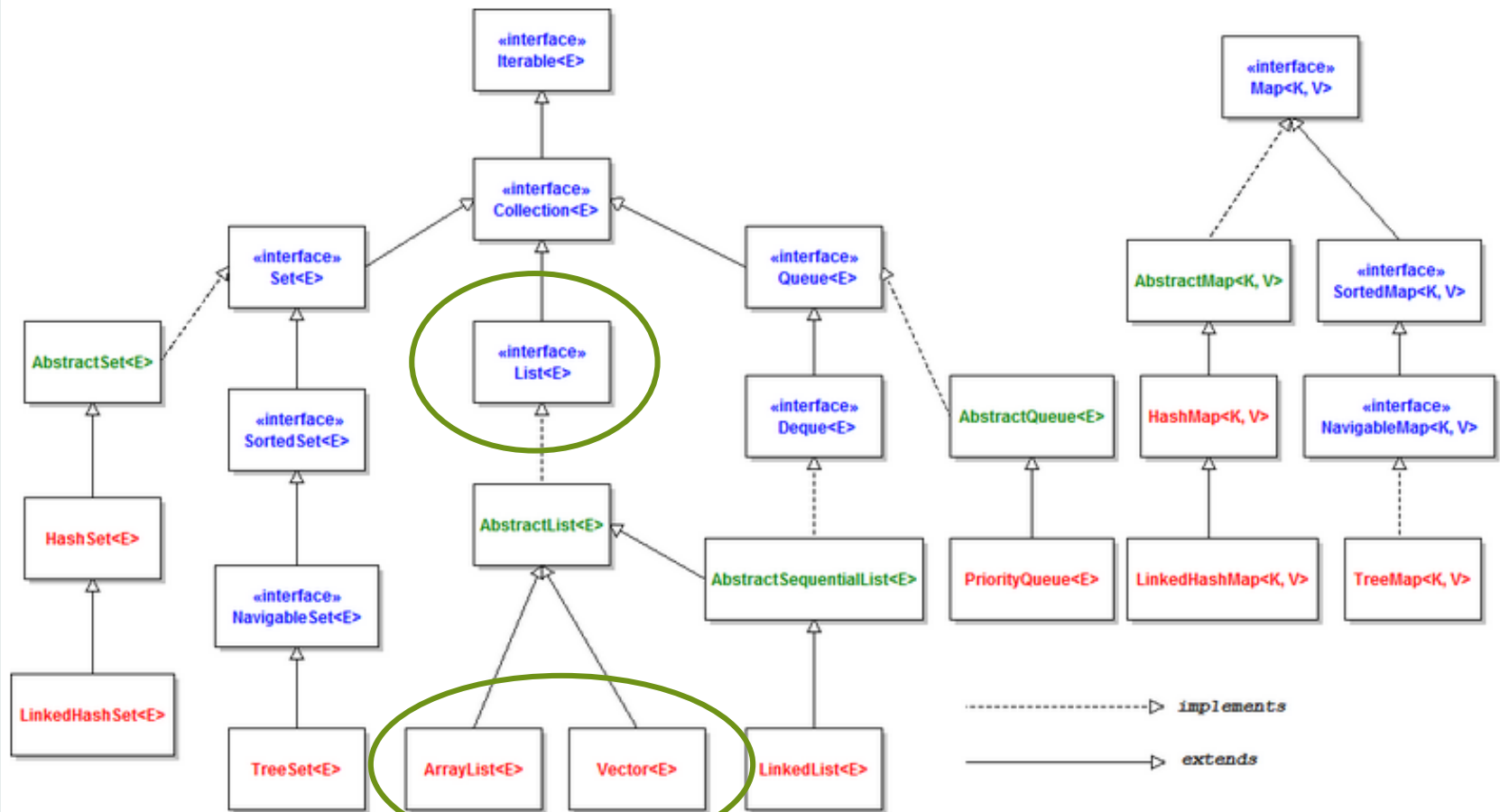
## Interface

```
Public Interface NomInterface
{
//pas de constructeur
//méthodes abstraites
public void calcule();
}
```

## Classe fille

```
Public Classe ClasseFille
implements NomInterface
{
//attributs
//constructeurs
//méthodes
//méthode obligatoire
public void calcule()
{
//code obligatoire
}
}
```

# EXAMPLE : INTERFACE LIST



Class diagram of Java Collections framework

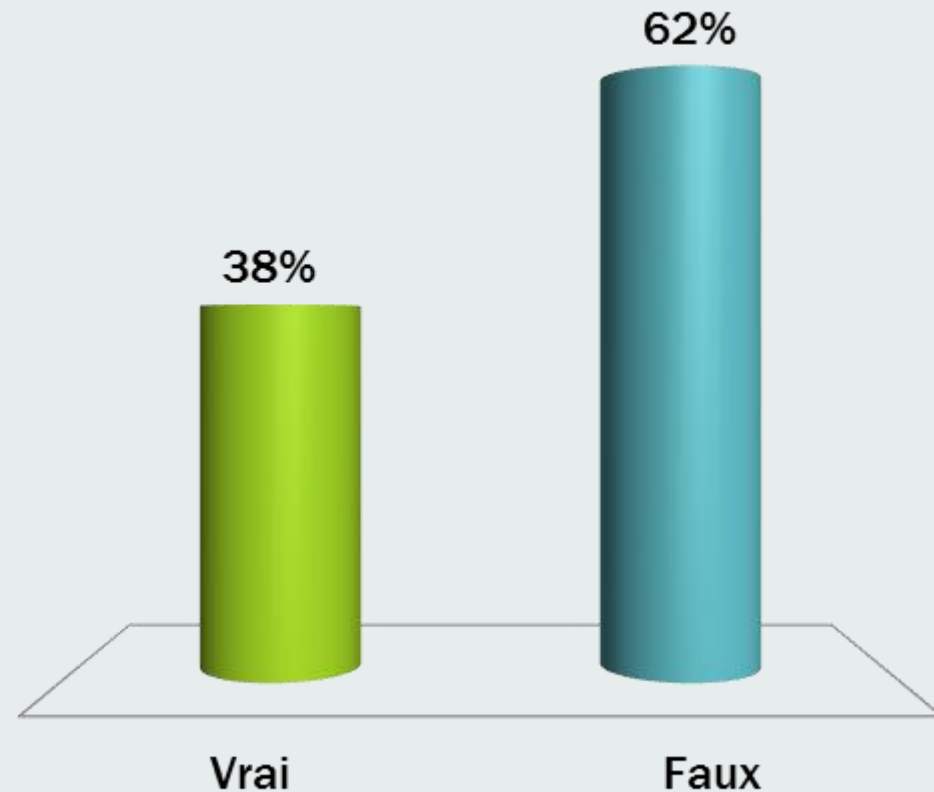
# EXAMPLE INTERFACE LIST

## Methods

Modifier and Type	Method and Description
boolean	<code>add(E e)</code> Appends the specified element to the end of this list (optional operation).
void	<code>add(int index, E element)</code> Inserts the specified element at the specified position in this list (optional operation).
boolean	<code>addAll(Collection&lt;? extends E&gt; c)</code> Appends all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's iterator (optional operation).
boolean	<code>addAll(int index, Collection&lt;? extends E&gt; c)</code> Inserts all of the elements in the specified collection into this list at the specified position (optional operation).
void	<code>clear()</code> Removes all of the elements from this list (optional operation).
boolean	<code>contains(Object o)</code> Returns true if this list contains the specified element.
boolean	<code>containsAll(Collection&lt;?&gt; c)</code> Returns true if this list contains all of the elements of the specified collection.
boolean	<code>equals(Object o)</code> Compares the specified object with this list for equality.
E	<code>get(int index)</code> Returns the element at the specified position in this list.
int	<code>hashCode()</code> Returns the hash code value for this list.
int	<code>indexOf(Object o)</code> Returns the index of the first occurrence of the specified element in this list, or -1 if this list does not contain the element.

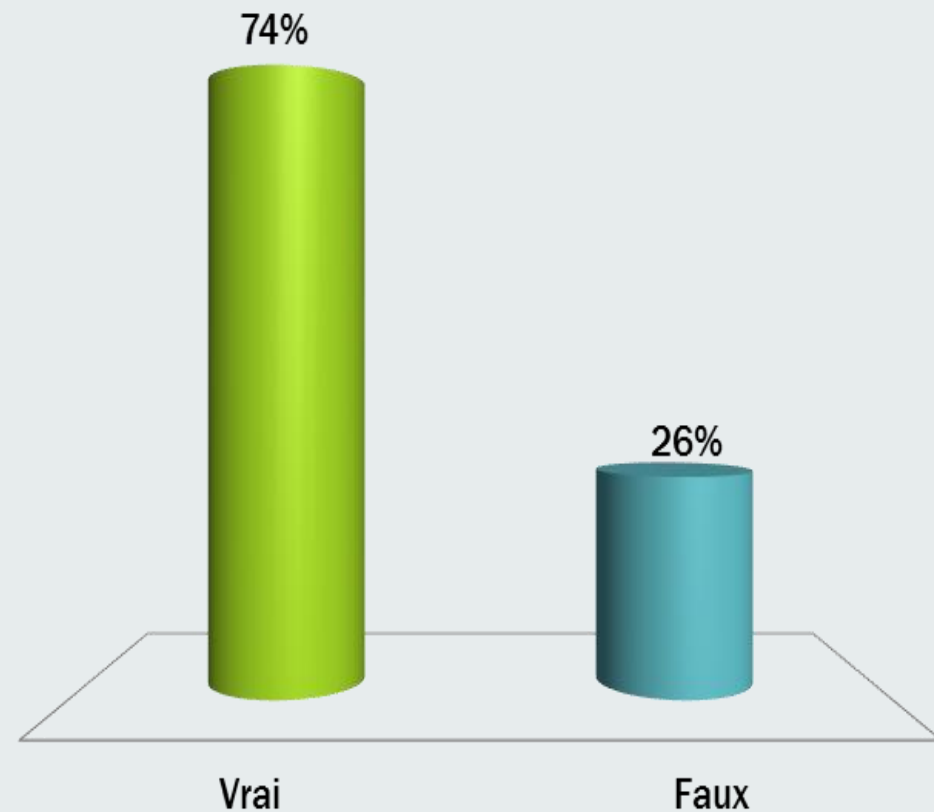
# UNE CLASSE PEUT HÉRITER DE PLUSIEURS CLASSES

- A. Vrai
- B. Faux



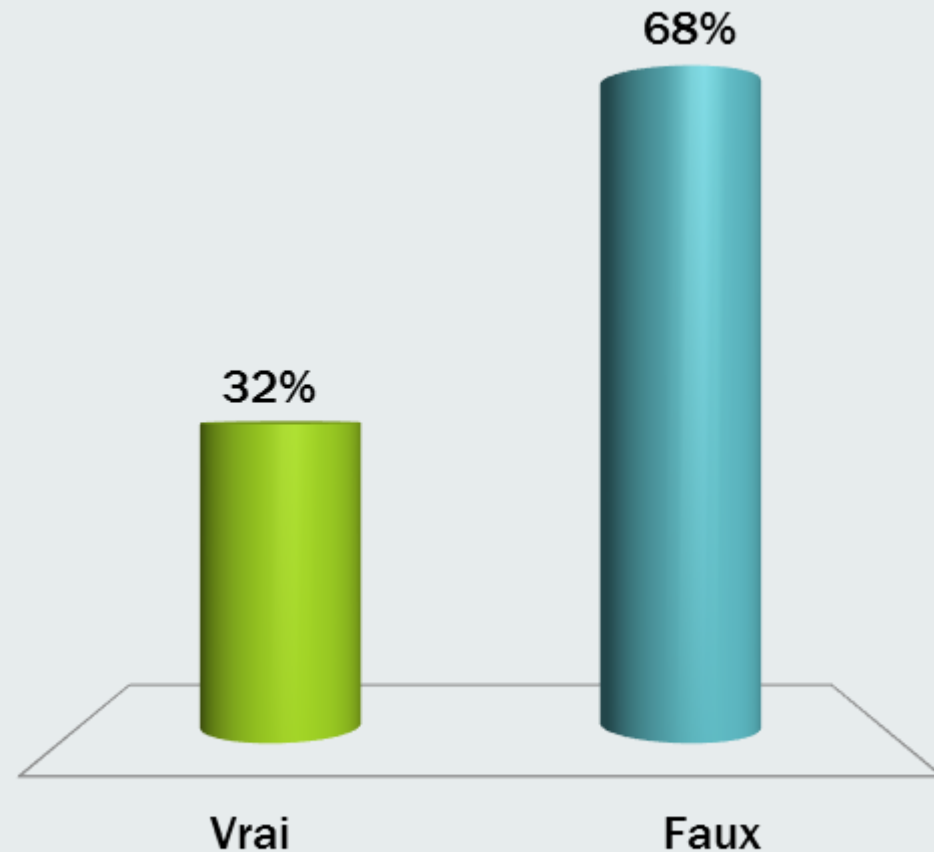
# UNE CLASSE PEUT IMPLÉMENTER PLUSIEURS INTERFACES

- A. Vrai
- B. Faux



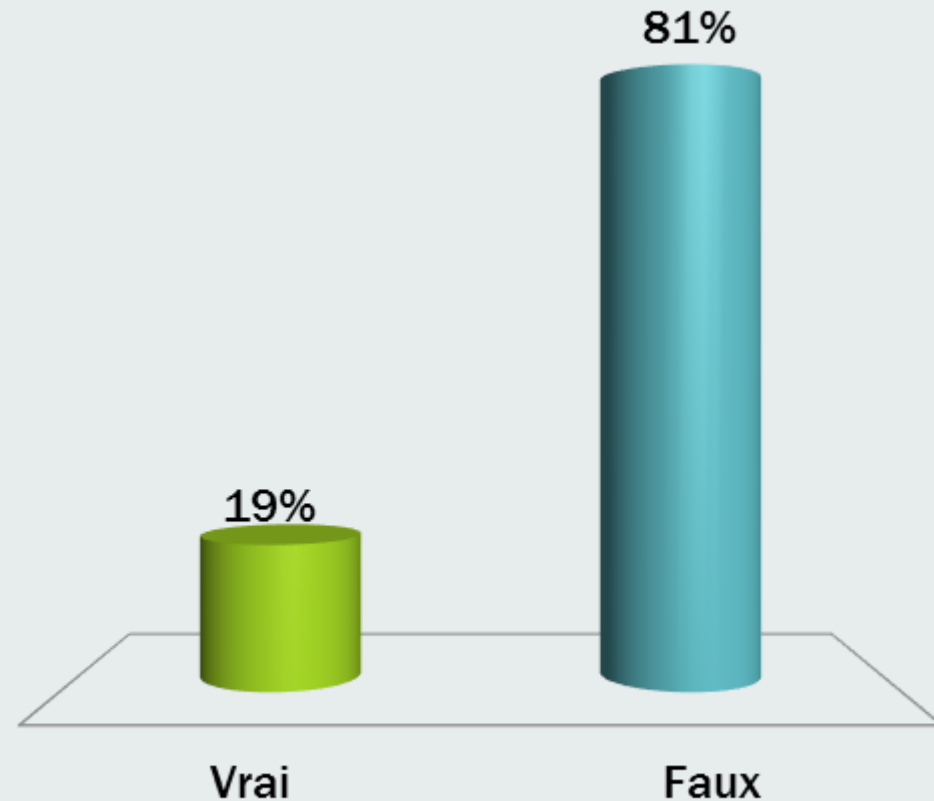
# UNE CLASSE ABSTRAITE IMPOSE DES ATTRIBUTS À SES CLASSES FILLES

- A. Vrai
- B. Faux



# UN OBJET D'UNE CLASSE ABSTRAITE NE PEUT APPELER QUE DES MÉTHODES ABSTRAITES

- A. Vrai
- B. Faux



# NOUVEAUTÉS JAVA 8



# VERSION DU FRAMEWORK JAVA

## JAVA 8

- 2014-2017
- Interfaces fonctionnelles
- Expressions lambdas
- Méthodes par défaut
- Stream : flux de données
- API java.time

## JAVA 9

- Sortie le 21/09/2017
- Modularisation de la JDK
- Mise à jour du garbage collector
- REPL jShell: interpréteur de commande JAVA
- Évolution du format de fichier jar
- Nouvelles API pour les Collections
- ...

# JAVA 8 -INTERFACES FONCTIONNELLES

- Interface avec une seule méthode abstraite
- Erreur de compilation si une interface fonctionnelle a plus d'une méthode

```
@FunctionalInterface  
public interface Runnable {  
    void run();  
}
```

# JAVA 8 – EXPRESSION LAMBDA

## Avant

```
Arrays.sort(testStrings, new Comparator<String>() {  
    @Override  
    public int compare(String s1, String s2) {  
        return(s1.length() - s2.length());  
    }  
});
```

## Après

### ■ Simplification de l'écriture du code

```
// Forme longue :  
Arrays.sort(testStrings, (String s1, String s2) -> { return s1.length() - s2.length(); });  
  
// Forme courte (possible uniquement s'il n'y a qu'une instruction) :  
Arrays.sort(testStrings, (String s1, String s2) -> s1.length() - s2.length());
```

# JAVA 8 – MÉTHODES PAR DÉFAUT

- Possibilité de définir le comportement par défaut d'une méthode abstraite dans une interface
- Exemple :

```
interface Person {  
    void sayGoodBye();  
    default void sayHello() {  
        System.out.println("Hello there!");  
    }  
}
```

Pas d'obligation pour une classe fille de définir SayHello

# JAVA 8 - STREAM

- Possibilité de créer un flux à partir d'une source de données (tableau, collection, fichier...)
- Possibilité de transformation d'un flux
- Opérations sur le flux (tri, filtre...)

# JAVA 8 – API JAVA.TIME

- Classe pour gérer le temps machine
- Classe pour gérer le temps humain :
  - LocalDate
  - LocalTime
  - Méthode static now()
  - Conversion...

# JAVA 9 - MODULARISATION

- **Projet Jigsaw**
- **Jdk structurée pour permettre de charger uniquement les modules nécessaires**
- **Gains de performance**
- **Gains de sécurité**
- **Dans le fichier module-info.java, on définit les packages nécessaires à l'application**

```
module fr.developpez.com {  
    requires java.sql;  
    opens fr.developpez.com.services;  
}
```

# JAVA 9- EVOLUTION DU FORMAT DE FICHER JAR

- Permet de définir des bibliothèques différentes à utiliser en fonction de la version de la jdk installée



# JAVA 9 – EVOLUTION API DES COLLECTIONS

- Méthodes statiques ajoutées pour faciliter l'initialisation des collections et rendre les attributs non modifiables

- Avant

```
List<String> myList = new ArrayList<String>();  
myList.add("Developpez.com");  
myList.add("Aime");  
myList.add("Java");  
myList = Collections.unmodifiableList(myList);
```

- Après

```
List<String> newList = List.of("Developpez.com", "Aime", "Java")
```

# JAVA 9 – REPL JSHELL

- Read Evaluate Print Loop
- Equivalent au Shell de Python
- Interpréteur de commandes java (sans définition de classe ou méthode)

**PERSISTENCE**

# PERSISTANCE DES DONNÉES

- La majorité des applications utilisent des données
- Ces données doivent être
  - Conservées
  - Sauvegardées
  - A jour
  - Sans erreur...

# SÉRIALISATION

- La sérialisation permet de rendre un objet persistant :
  - Pour stocker l'objet après la fermeture de l'application
  - Pour échanger des informations entre applications
  - ...
- La sérialisation peut être réalisée
  - Dans un fichier
  - Dans une base de données
- Le framework Java gère la persistance par le biais de l'interface `Serializable`

# SÉRIALISATION D'UN OBJET

- Nécessaire d'implémenter l'interface `Serializable` et de créer un identifiant dans la classe
- Les types standards sont sérialisables
- Les types complexes ne sont pas sérialisables et doivent être définis comme `transient` (ignoré lors de la sérialisation)

# SERIALISATION D'UN OBJET

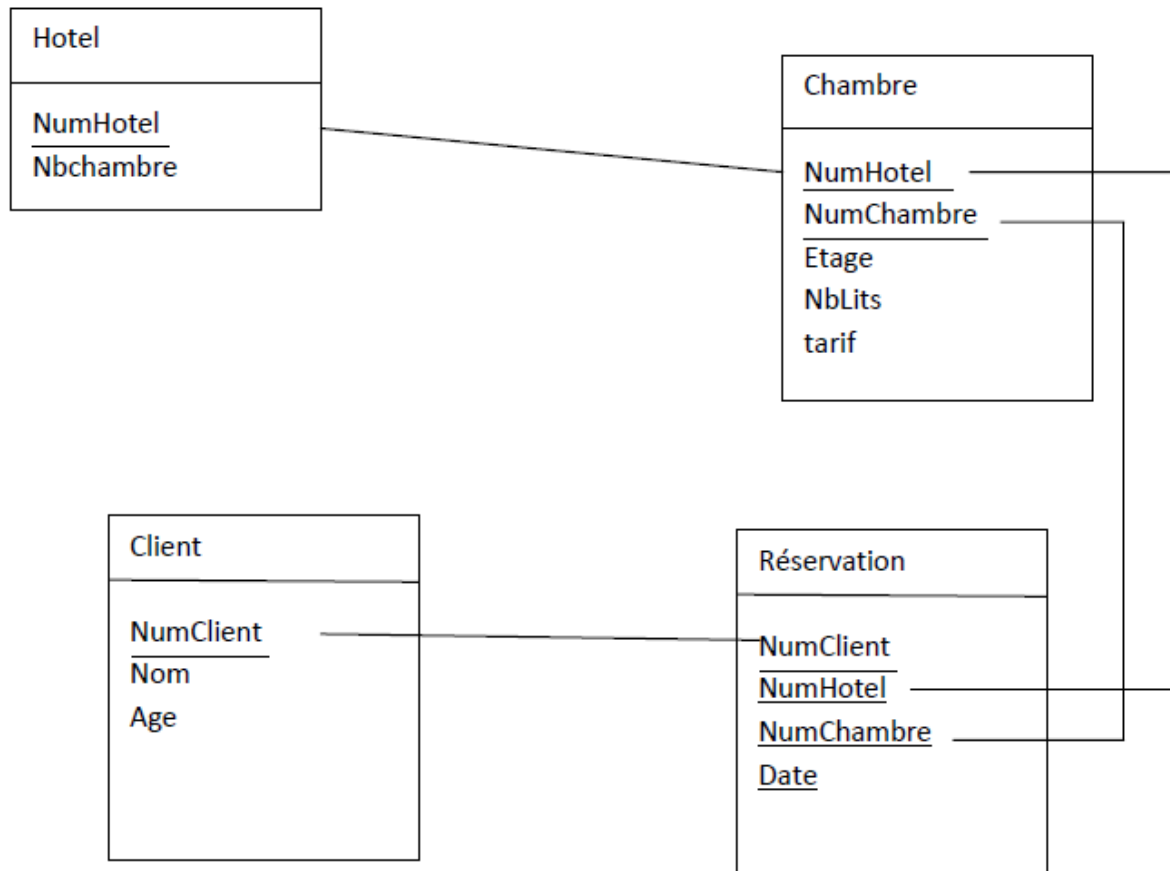
- Le programme doit créer un flux de sérialisation associé à un fichier
  - Utilisation de la méthode `writeObject` pour écrire l'objet dans le fichier
  - Utilisation de la méthode `readObject` pour lire les informations d'un objet depuis un fichier

# PERSISTANCE DES DONNÉES

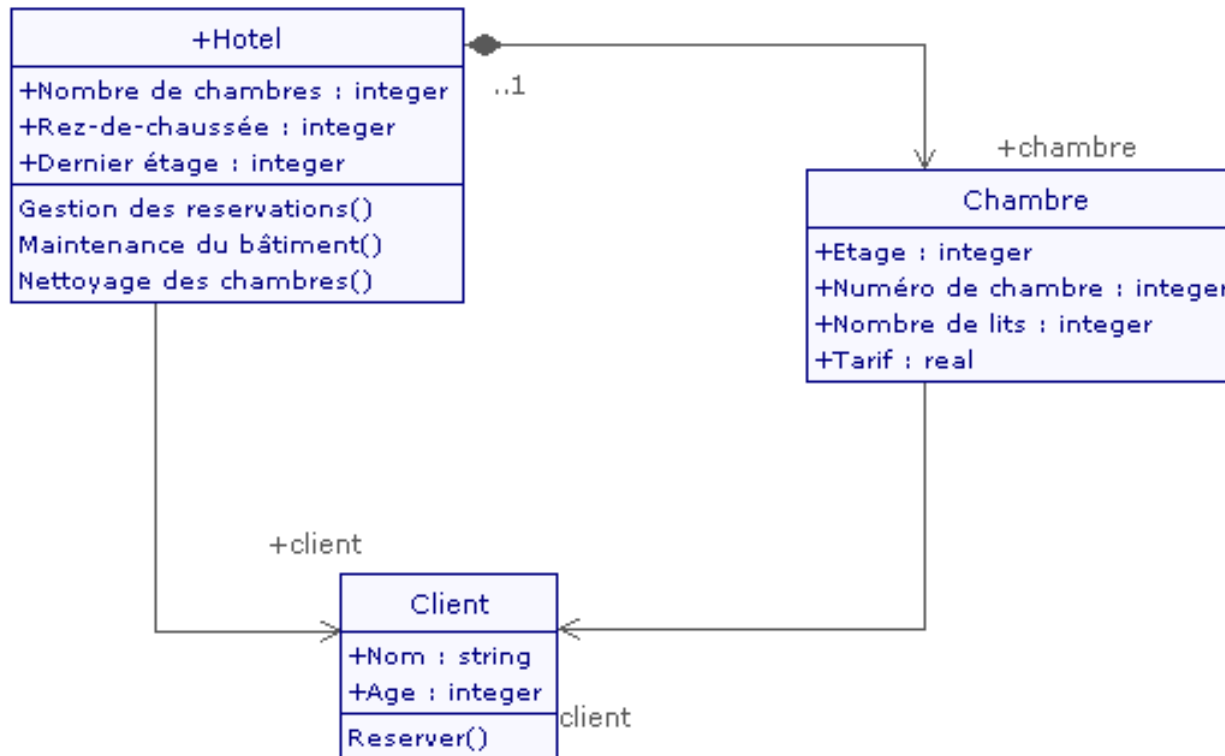
- Outil de stockage des données :
  - Bases de données relationnelle
- Langages de programmation :
  - Procédural
  - Objet



# MODÈLE RELATIONNEL



# DIAGRAMME DE CLASSE



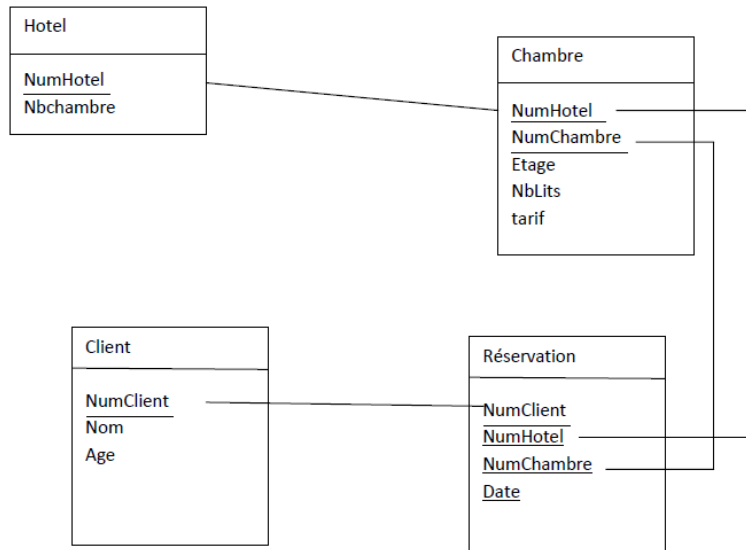
# PERSISTANCE DES DONNÉES

## ■ Particularités des représentations

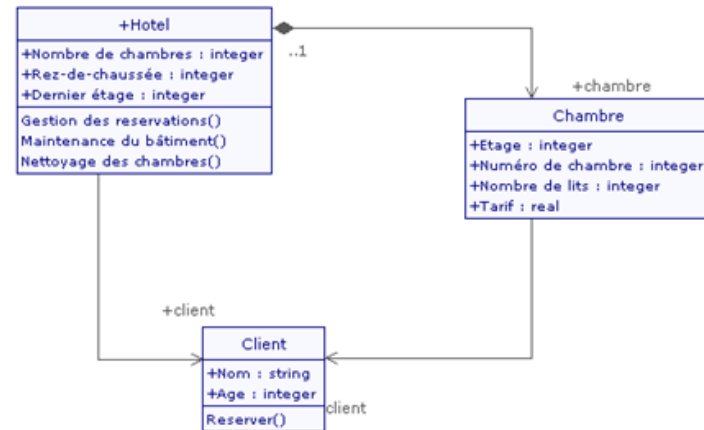
Relationnel	Objet
Identifiant pour chaque enregistrement	Héritage
	Polymorphisme

# MAPPING OBJET-RELATIONNEL

## Base de données



## Classes



Description de la correspondance entre les classes des programmes et les tables de la base de données

# ACCÈS CLASSIQUE AUX DONNÉES DE LA BD

- Utilisation d'un driver JDBC ou ODBC

```
Class.forName("com.mysql.jdbc.Driver").newInstance();
this.dbConnect = DriverManager.getConnection("jdbc:mysql:" +
    this.dbURL, this.user, this.password);
this.dbStatement = this.dbConnect.createStatement();

String requete = "delete from Formation where idForm = " + id;
ResultSet res = bd.mySQLEXec(requete);
if (res != null) {
    return true;
}
```

# LA PERSISTENCE

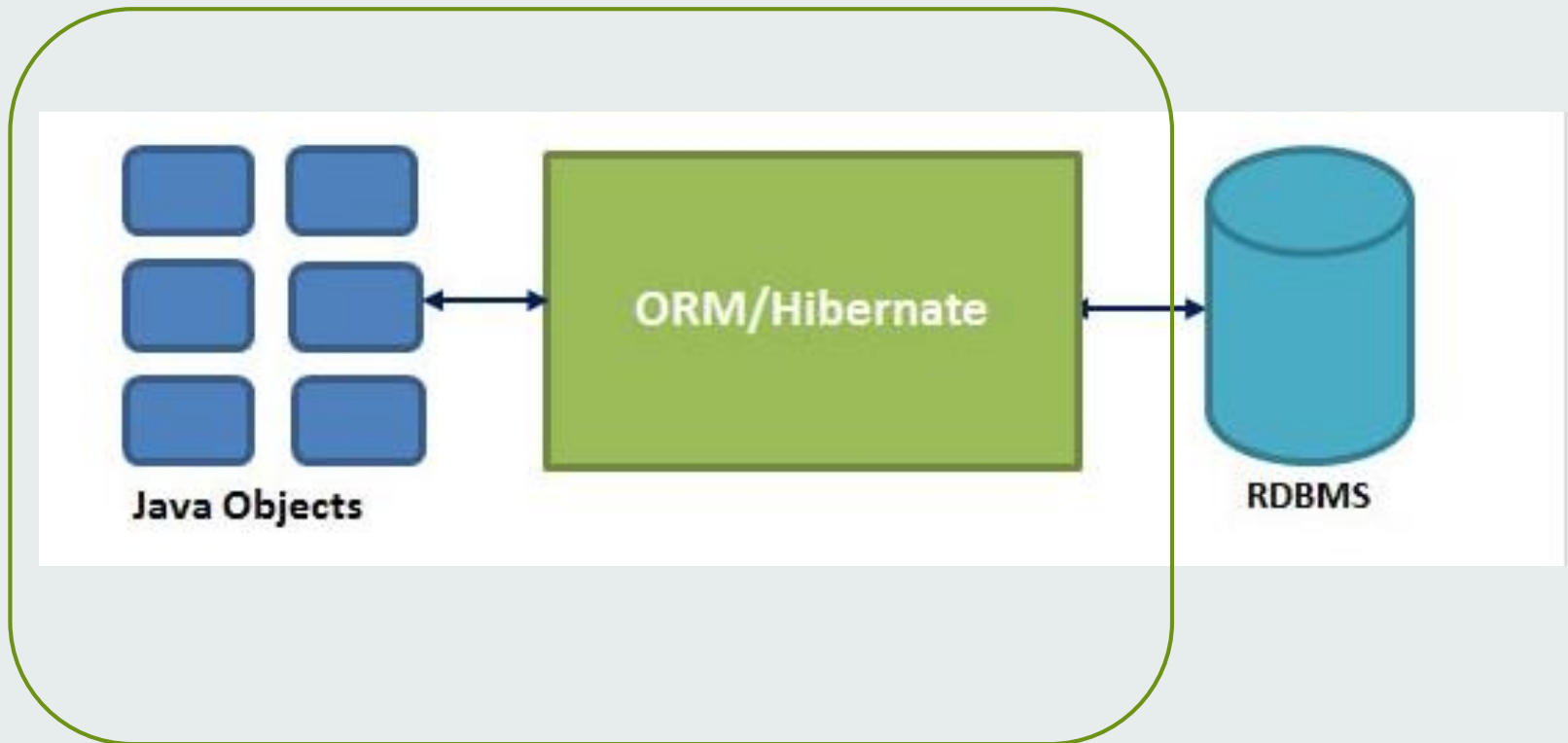
- Gestion des données automatique (requêtes générées à partir du mapping objet-relationnel)
  - Ajout d'un objet = insert des enregistrements
  - Modification d'un objet = update
  - Suppression d'un objet = delete

# NOTIONS DE LA PERSISTANCE

- Sauvegarde des données en temps réel
- Reprise après panne
- Gestion de la mémoire
- Assure l'intégrité des données
- Gestion des accès concurrents

# EXEMPLE : HIBERNATE

Couche métier





# CONFIGURER LA CONNEXION A LA BD

```
<hibernate-configuration>
  <session-factory>
    <property name="connection.driver_class">com.mysql.jdbc.Driver</property>
    <property name="connection.url">jdbc:mysql://localhost:3306/test_db</property>
    <property name="connection.username">root</property>
    <property name="connection.password">xxx</property>

    <property name="connection.pool_size">1</property>
    <property name="dialect">org.hibernate.dialect.MySQLDialect</property>
    <property name="current_session_context_class">thread</property>
    <property name="cache.provider_class">org.hibernate.cache.NoCacheProvider</property>
    <property name="show_sql">true</property>
    <property name="hbm2ddl.auto">validate</property>

    <mapping class ="models.Category" />

  </session-factory>
</hibernate-configuration>
```


# MAPPING

## Correspondance Classe - Table

```
01. <?xml version="1.0"?><!DOCTYPE hibernate-mapping
02. PUBLIC "-//Hibernate/Hibernate Mapping DTD 2.0//EN"
03. "http://hibernate.sourceforge.net/hibernate-mapping-2.0.dtd"><hibernate-mapping>
04. <class name="Personnes" table="personnes">
05.   <id name="idPersonne" type="int" column="idpersonne">
06.     <generator class="native"/>
07.   </id>
08.   <property name="nomPersonne" type="string" not-null="true" />
09.   <property name="prenomPersonne" type="string" not-null="true" />
10.   <property name="datenaissPersonne" type="date">
11.     <meta attribute="field-description">date de naissance</meta>
12.   </property>
13. </class>
14. </hibernate-mapping>
```

# EXEMPLE DE CRÉATION D'OBJET PERSISTANT

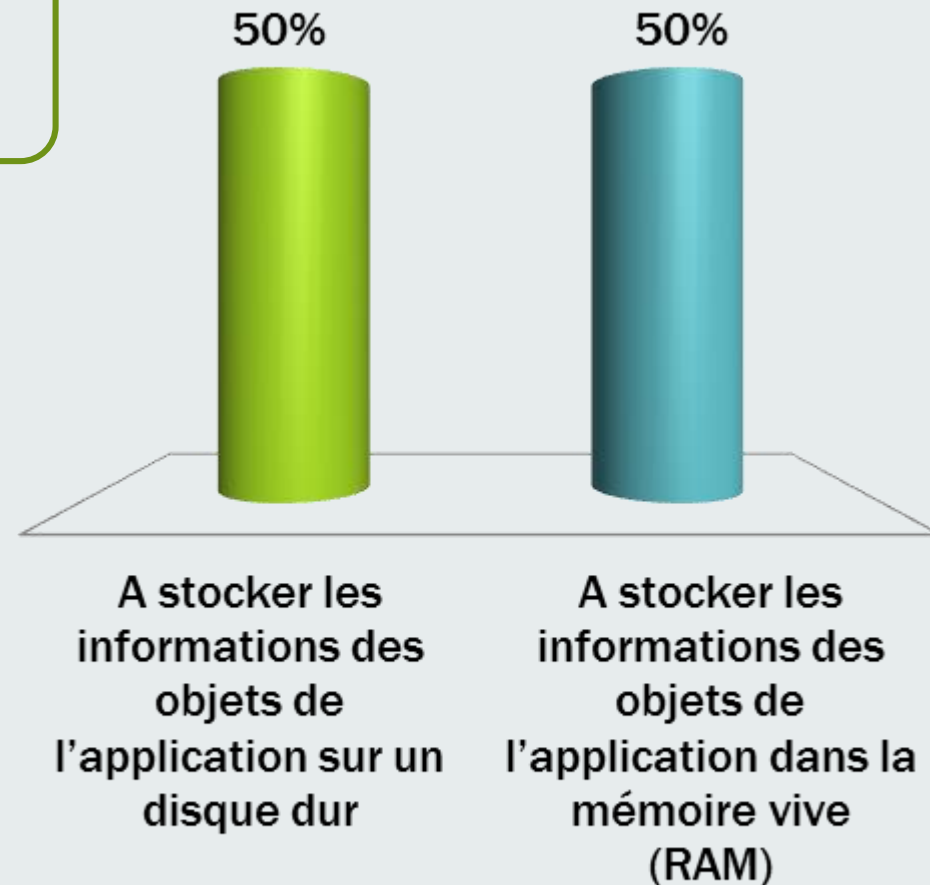
```
Transaction tx = null;
try {
    tx = session.beginTransaction();
    Personnes personne = new Personnes("nom3", "prenom3", new Date());
    session.save(personne);
    session.flush() ;
    tx.commit();
} catch (Exception e) {
    if (tx != null) {
        tx.rollback();
    }
    throw e;
}
```



Enregistre les données dans la BD

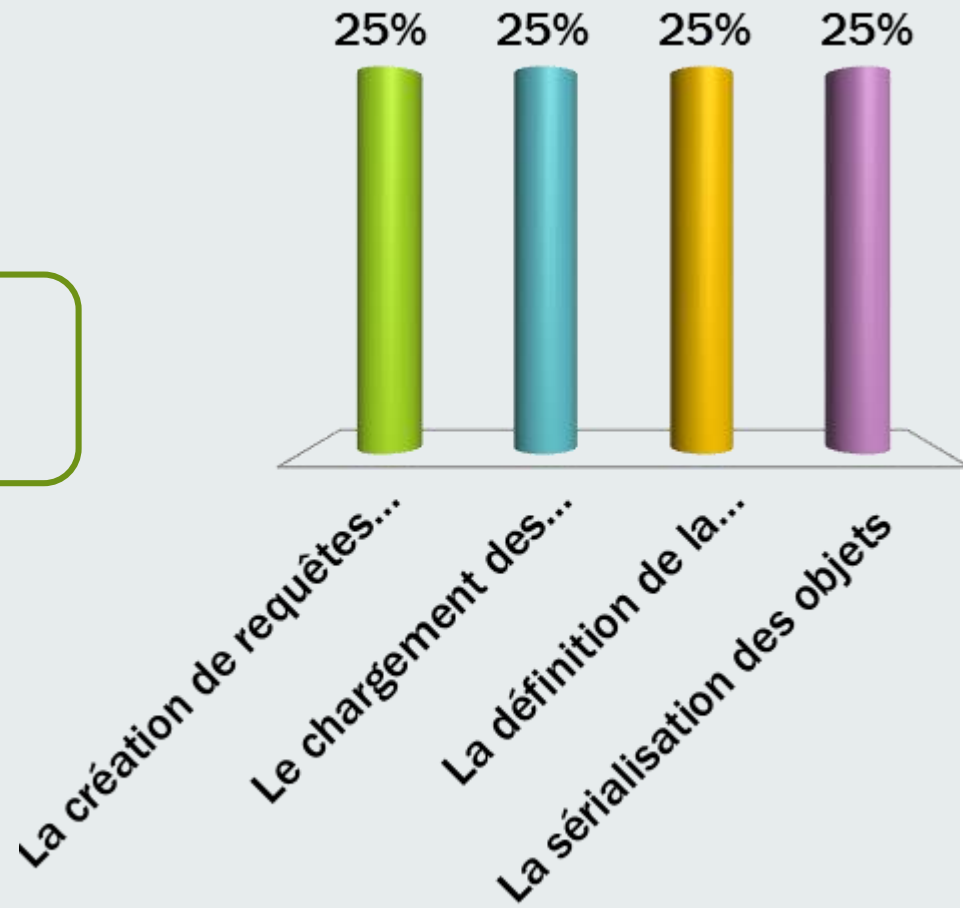
# A QUOI SERT LA PERSISTANCE ?

- A. A stocker les informations des objets de l'application sur un disque dur
- B. A stocker les informations des objets de l'application dans la mémoire vive (RAM)



# QU'EST-CE QUE LE MAPPING OBJET-RELATIONNEL ?

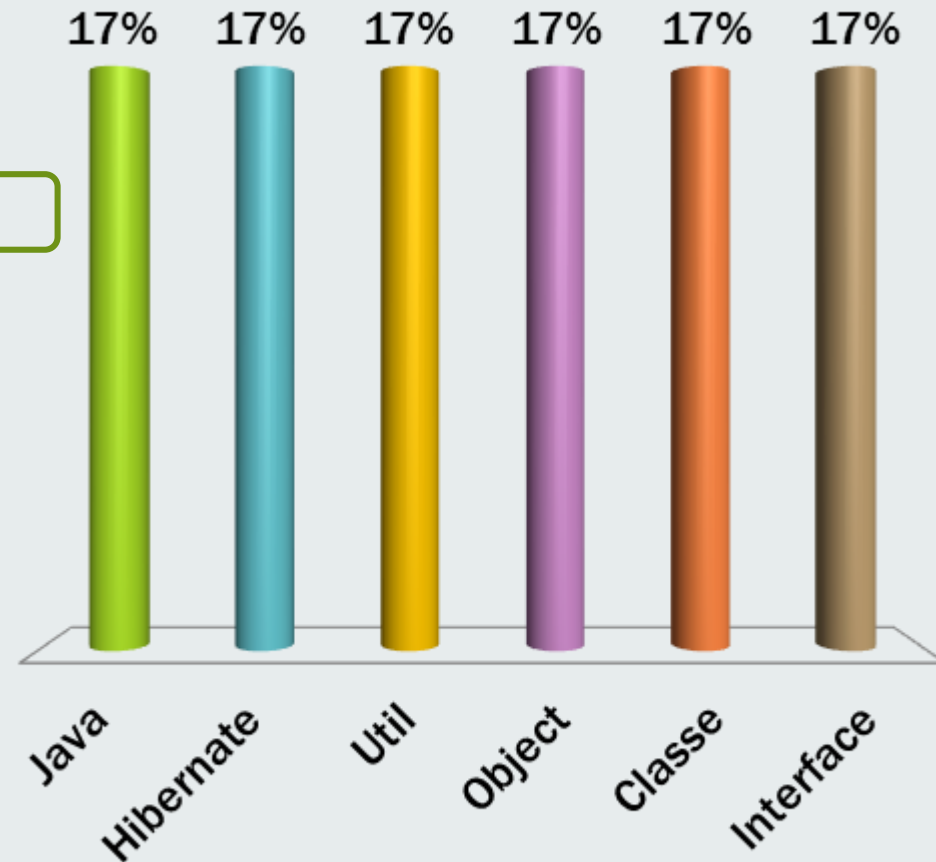
- A. La création de requêtes SQL dans une classe
- B. Le chargement des données des tables
- C. La définition de la correspondance entre les classes et les tables
- D. La sérialisation des objets



QUIZ

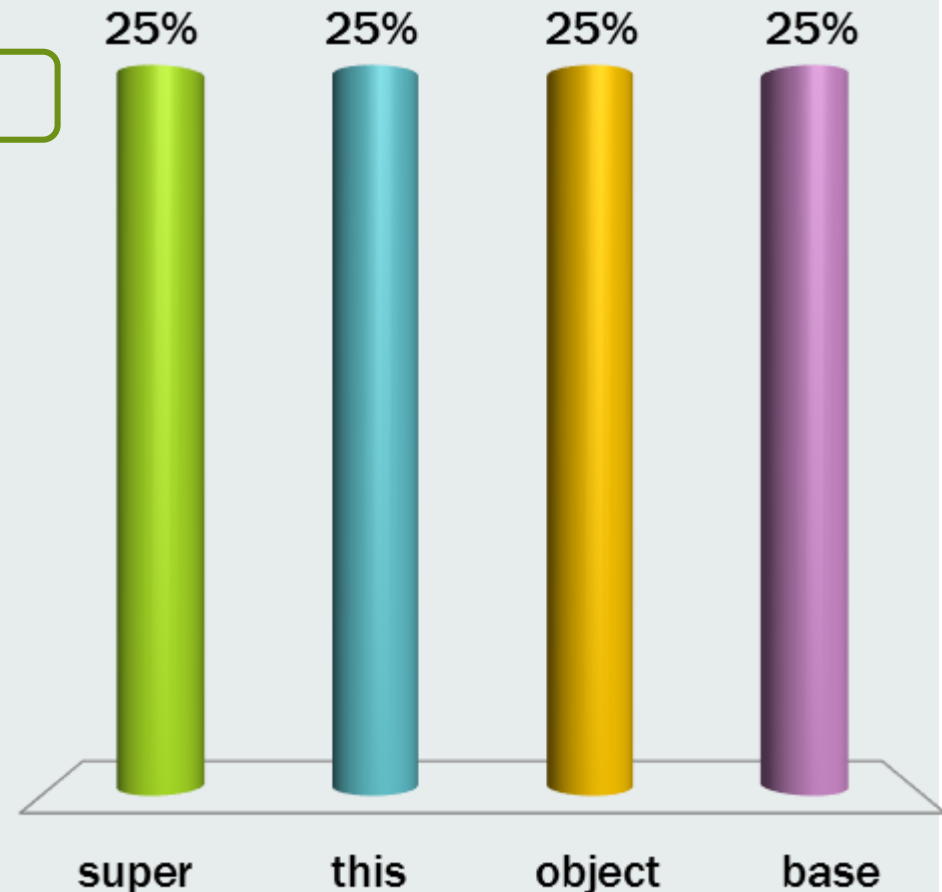
# QUELLE EST LA CLASSE DE BASE EN JAVA ?

- A. Java
- B. Hibernate
- C. Util
- D. Object**
- E. Classe
- F. Interface



# DANS UNE MÉTHODE, COMMENT ACCÈDE-T-ON À L'OBJET L'AYANT APPELÉE?

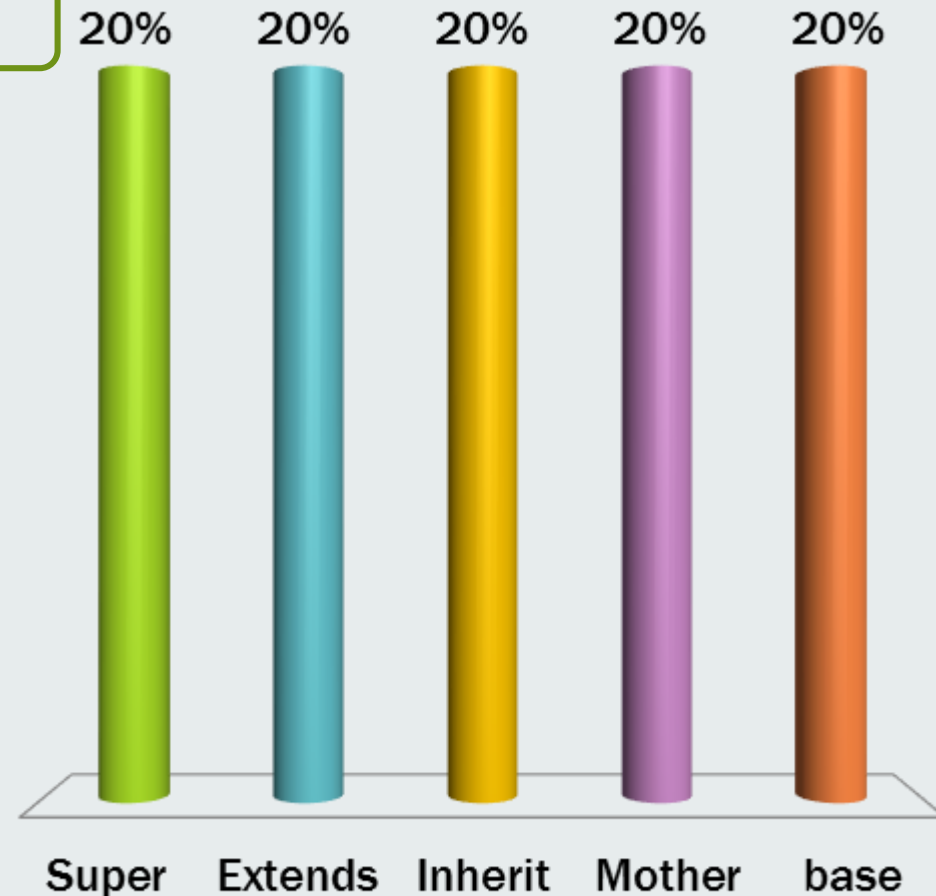
- A. super
- B. this
- C. object
- D. base





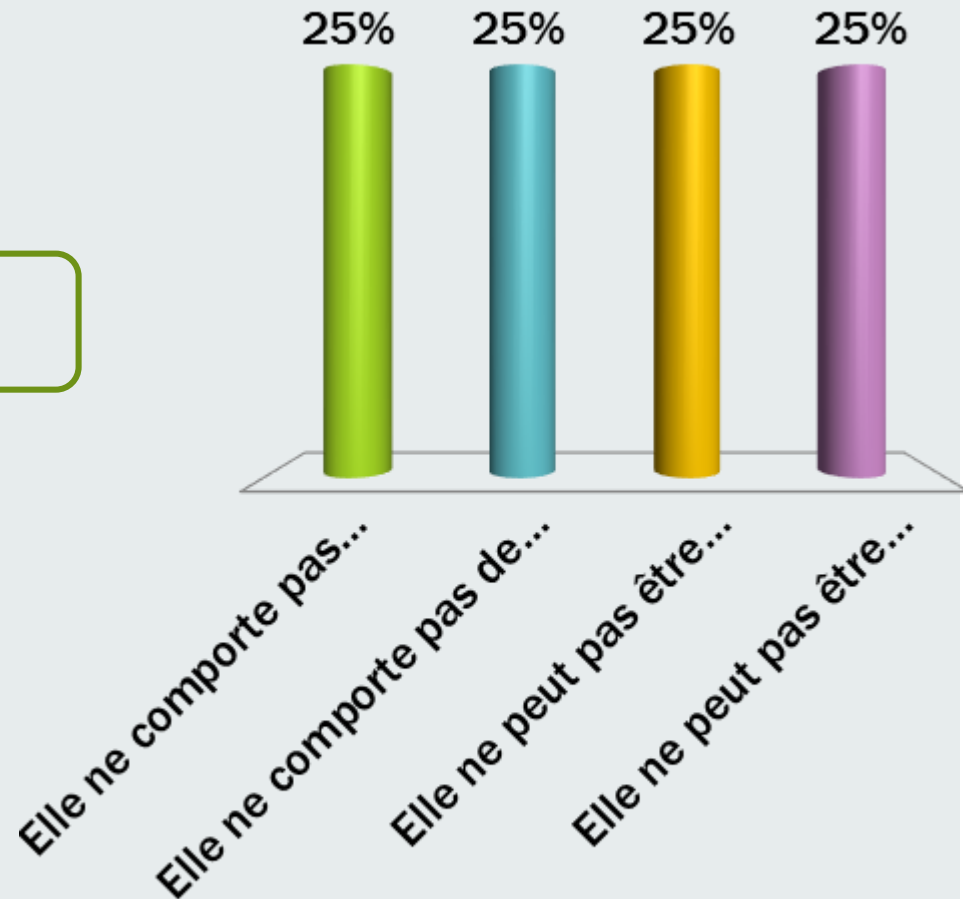
# QUEL MOT CLÉ EST UTILISÉ EN JAVA POUR ACCÉDER À UNE MÉTHODE DE LA CLASSE MÈRE (DEPUIS UNE CLASSE FILLE)

- A. Super
- B. Extends
- C. Inherit
- D. Mother
- E. base



# QUELLE EST LA PARTICULARITÉ D'UNE CLASSE ABSTRAITE ?

- A. Elle ne comporte pas d'attributs
- B. Elle ne comporte pas de méthodes
- C. Elle ne peut pas être instanciée**
- D. Elle ne peut pas être utilisée comme un type dans un programme



# QUELLE EST L'UTILITÉ D'UNE INTERFACE ?

- A. Définir des attributs communs
- B. Définir un comportement commun
- C. Proposer des méthodes à exécuter

