



Java orienté objet

Cours 2

les objets, les classes, les méthodes, l'héritage, les packages, collection d'objets



Programme de la séance

- Notion d'objets et de classe en java
- Les attributs de classe
- Les constructeurs de classe
- Les méthodes de classe
- L'héritage
- Les packages
- Collection d'objets



Objets et classes d'objets

- La classe

La classe est la généralisation du type défini par l'utilisateur qui permet le regroupement des données et des traitements qui leur sont associés sous une même entité.

Sous une classe se trouvent associées à la fois :

- Les attributs (les données)**
- Les méthodes relatives à ces données.**

Les méthodes agissent sur les données de la classe qui sont protégées de toute ingérence externe.

⇒ ***Une classe génère une famille d'objets***

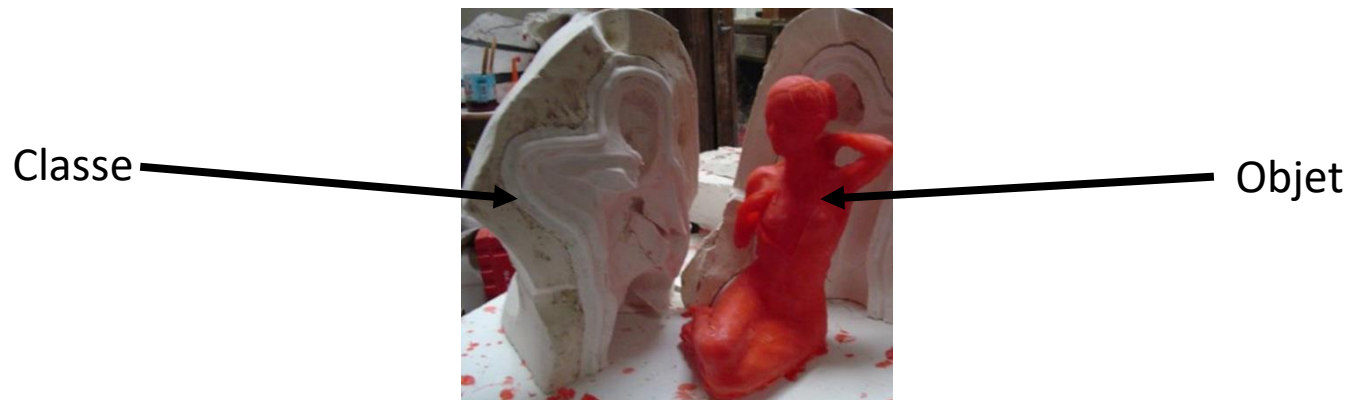
Objets et classes d'objets

- **L'objet**

L'objet est une instance de la classe correspondante.

une structure et un comportement.

- Dans la réalité, physique ou abstraite, les objets sont présents partout : *étudiants, employés, avions, chèques....*
- Dans un système graphique : *des points, des lignes, cercles...*



- **La POO** manipule ces différents objets.
- Pour chacun d'entre eux, on a besoin d'un minimum d'information sur leur structure et sur leurs comportements

Objets et classes d'objets

- Exemple d'une classe et de ces instances (objets)

Etudiant
nom : String prenoms : String age : int pays: String genre : char
concevoir() produire() vendre() ...



eleve1= new Etudiant(...)
nom = Fatim prenoms = Zahra age = 20 Pays = Maroc genre = F



Représentation
physique

	Fatim Zahra 20 ans Femme Marocaine
	



Visibilité des attributs et méthodes

- En programmation, la visibilité détermine qui peut faire appel à une classe, une méthode ou une variable à l'intérieur du programme.
- Visibilité des attributs et méthodes :
 - ❑ **public**: un attribut ou une méthode est dit public (public) si leur utilisation est autorisée en dehors de la classe.
 - ❑ **private**: un attribut ou une méthode est dit private (privé) si leur utilisation est interdite en dehors de la classe.
 - ❑ **protected**: un attribut ou une méthode est dit protected (protégé) si leur utilisation est limitée à la classe et ses descendantes.

Déclaration d'une classe

- Syntaxe :

```
visibilité class nom_classe {  
    1 - attributs ...  
    2 - constructeurs ...  
    3 - accesseurs ...  
    4 - méthodes ...  
}
```

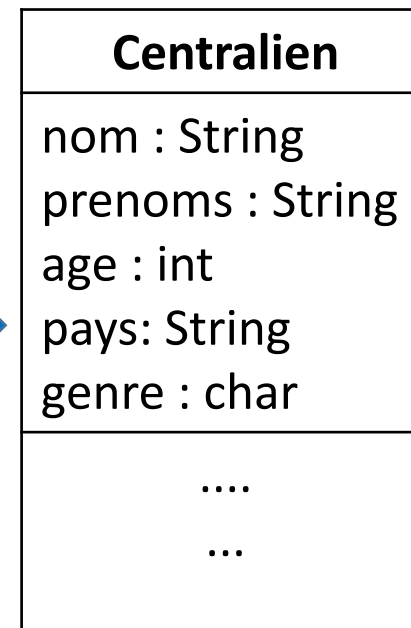
- Les attributs de la classe peuvent être de types primitives ou des objets.
- Par défaut, la classe est de type public.
- Par défaut, les attributs de la classe sont de types public.

Les attributs d'une classe

- La déclaration est presque similaire à celle d'une variable.
- Syntaxe : **visibilité type nom_attribut**

```
public class Centralien {  
  
    // 1 - les attributs  
    private String nom;  
    private String prenom;  
    private int age;  
    private String pays;  
    private char genre;  
  
}
```

Code Java



Modèle

Les constructeurs

- Un constructeur est une méthode d'instance qui sert à créer un objet et, le cas échéant, d'initialiser ses variables de classe.
- Le constructeur est une méthode qui n'a aucun type de retour et prend le nom sa classe.
- Il existe deux types de constructeurs:
 - ❑ **Constructeur par défaut**
 - ❑ **Constructeur avec paramètres**
- **Le constructeur par défaut** est sans paramètres et permet de créer un objet vide.
- **Le constructeur avec paramètres** prend des valeurs en paramètres et permet d'initialiser un objet lors de sa création.

Les constructeurs

- Exemple

```
*Centralien.java ☒
1
2 public class Centralien {
3
4     // 1 - les attributs
5     private String nom;
6     private String prenoms;
7     private int age;
8     private String pays;
9     private char genre;
10
11     // 2 - Constructeur par défaut
12     public Centralien() {
13     }
14
15     // 2 - Constructeur avec paramètres
16     public Centralien(String nom, String prenoms, int age, String pays, char genre) {
17         this.nom = nom;
18         this.prenoms = prenoms;
19         this.age = age;
20         this.pays = pays;
21         this.genre = genre;
22     }
23 }
```

- Le mot clé **this** permet de faire référence à l'objet courant, il permet de désigner une méthode ou une variable de l'objet.



Les accesseurs et mutateurs

- Un **accesseur** est une méthode qui permet d'accéder et afficher les attributs des objets d'une classe.

```
public type getNomattribut() {  
    return this.nomattribut;  
}
```

- Un **mutateur** est une méthode qui permet de modifier la valeur des attributs.

```
public setNomattribut(type valeur) {  
    this.nomattribut = valeur;  
}
```

- Par convention de nommage, les accesseurs commencent par **get** suivies du nom de l'attribut et les mutateurs commencent par **set**.



Les accesseurs et mutateurs

- Exemple

```
// 3 - Accesseurs et mutateurs
public String getNom() {
    return this.nom;
}

public void setNom(String n) {
    this.nom = n;
}

public String getPrenoms() {
    return this.prenoms;
}

public void setPrenoms(String pn) {
    this.prenoms = pn;
}

public int getAge() {
    return this.age;
}

public void setAge(int a) {
    this.age = a;
}

public String getPays() {
    return pays;
}

public void setPays(String p) {
    this.pays = p;
}
```

Les accesseurs et mutateurs

- Exemple

```
Centralien.java  *Principale.java X
public class Principale {

    public static void main(String[] args) {

        // instantiation d'un objet vide
        Centralien eleve1 =new Centralien();
        eleve1.setNom("KAMMEGNE");
        eleve1.setPrenoms("Pamela");
        eleve1.setAge(22);
        eleve1.setPays("Cameroun");
        eleve1.setGenre('F');

        //instantiation d'un objet avec paramètres
        Centralien eleve2 =new Centralien("ETTAMRI", "Ihssane", 22, "Marocaine", 'M');

        System.out.println("Eleve 1 : nom : " + eleve1.getNom() + ", prenom: " + eleve1.getPrenoms());
        System.out.println("Eleve 2 : nom " + eleve2.getNom() + ", age: " + eleve2.getAge() + "ans");
    }
}

Problems @ Javadoc Declaration Console Error Log
<terminated> Principale (1) [Java Application] C:\Program Files\Java\jre1.8.0_144\bin\javaw.exe (30 sept. 2017 00:24:26)
Eleve 1 : nom :KAMMEGNE, prenom: Pamela
Eleve 2 : nom ETTAMRI, age: 22ans
```

Les méthodes

- Les **méthodes** sont des sous programmes ou fonctions gérant le comportement d'un objet.
- Les méthodes peuvent prendre des paramètres (entrée/sortie, entrée ou sortie).
- Exemple de méthode de la classe Centralien

```
// 4 - les méthodes
public void presenteToi(){
    System.out.println("Bonjour, je m'appel " + this.nom + " " + this.prenoms + ", j'ai " + this.age + " ans");
}

public boolean testMaturite(){
    if (this.age >= 18){
        return true;
    }
    else {
        return false;
    }
}
```

Les méthodes

- Appel d'une méthode sur un objet ce fait via la syntaxe **objet.methode**

```
4 public class Principale {
5
6     public static void main(String[] args) {
7
8         Centralien eleve =new Centralien("ETTAMRI", "Ihssane", 22, "Marocaine", 'M');
9         eleve.presenteToi();
10        boolean test=eleve.testMaturite();
11        if(test==true){
12            System.out.println("Tu es mature");
13        }
14    }
15 }
```

Problems @ Javadoc Declaration Console Error Log

<terminated> Principale (1) [Java Application] C:\Program Files\Java\jre1.8.0_144\bin\javaw.exe (30 sept. 2017 14:47:02)

Bonjour, je m'appel ETTAMRI Ihssane et j'ai 22 ans
Tu es mature



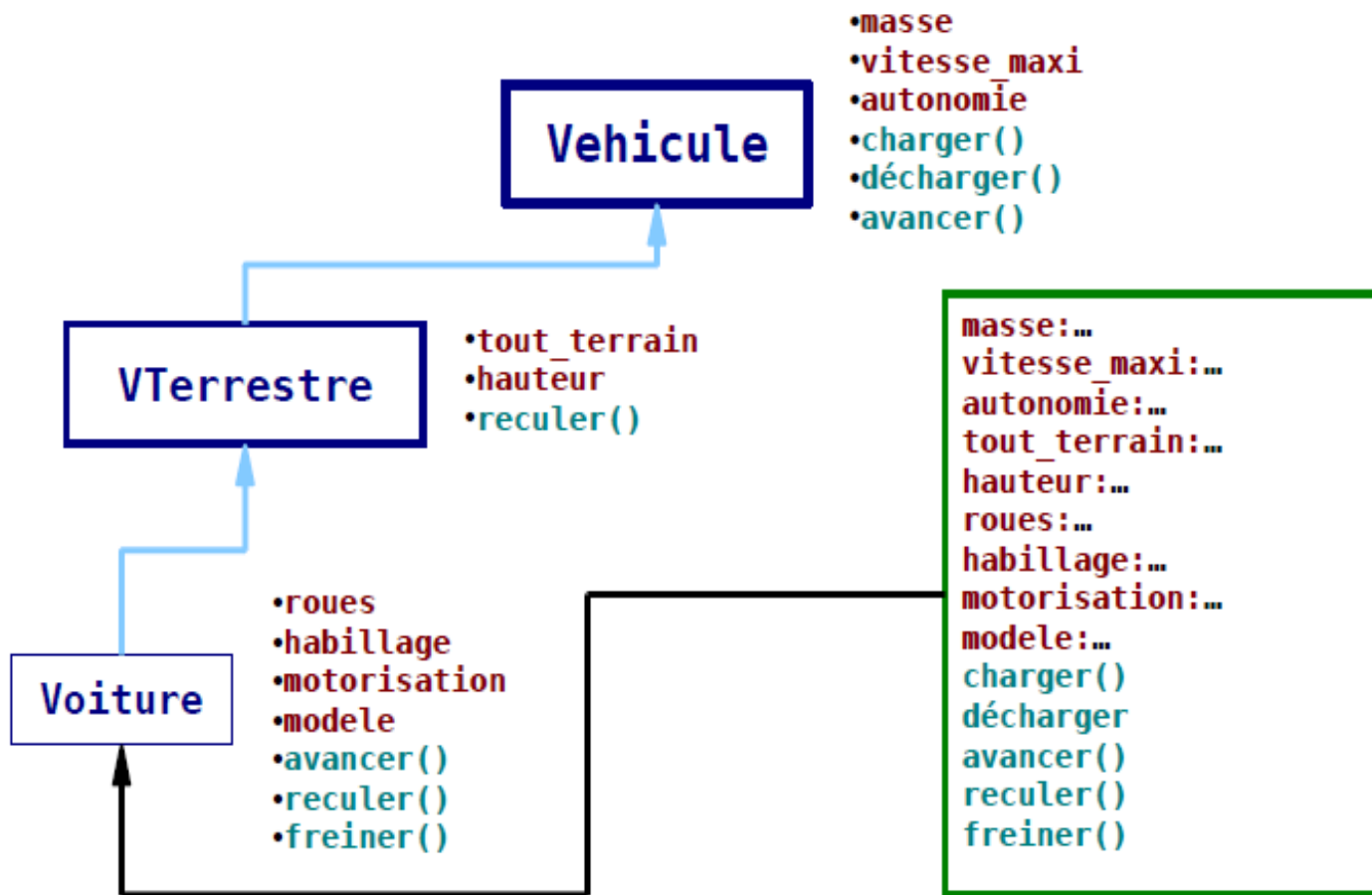
• Héritage:

- ✓ **Principe** : définir une nouvelle classe par spécialisation d'une (ou plusieurs) classes existantes.
- ✓ **Exemple** : Définir une classe **VMarin** sachant que la classe **Vehicule** existe.
- ✓ La classe **VMarin** :
 - **hérite** (récupère) tous les éléments de la classe Vehicule
 - **ajoute** un nom (par exemple) et les opérations pour manipuler le nom
 - **redéfinit** la méthode afficher (pour afficher les nouveaux attributs)
 - Véhicule est la **super-classe**, VMarin la **sous-classe**
- ✓ **Redéfinition** : Donner une nouvelle implantation à une méthode déjà présente dans une super-classe (override, en anglais ou encore surcharge en Français).
- ✓ **Polymorphisme** : plusieurs formes pour la même méthode.
 - ✓ **Principe**: Le **polymorphisme** est la faculté pour une méthode portant le même nom mais appartenant à des classes distinctes héritées d'effectuer un travail différent. Cette propriété est acquise par la technique de la surcharge
 - ✓ **Exemple** : plusieurs versions de la méthode afficher dans **VMarin** et celle de **Vehicule**



- **Exemple:**

Bénéficier des méthodes et attributs définis dans une classe **parente**.



- Déclaration des classes:
- Pour déclarer qu'une classe B hérite d'une classe A, il faut l'indiquer le mot clé **extends** lors de la déclaration de la classe B :
 - Le nom de la classe "mère" est indiquée après le nom de la classe "fille"

```
public class ClasseA {  
    # Détails de la classe ClasseA  
}  
class ClasseB extends ClasseA {  
    # Détails de la classe ClasseB  
}
```

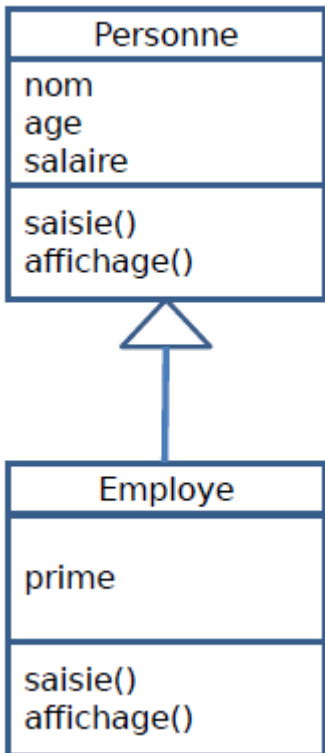
- Exemple

Héritage

```
public class Voiture extends VTerrestre {  
    //Une voiture pour le transport.  
    public void freiner(int dmax) {  
        //Arrêt de la voiture."  
        if (this.masse*this.vitesse**2/2 > dmax*this.kfrein ){  
            .....  
        }  
    }  
}
```

Attributs d'une classe héritée

- En plus de ces attributs, la classe fille hérite des attributs de la classe mère.



La classe **Employe** est une classe **Personne**, avec le champ supplémentaire **prime**. Cela nécessitera la reprogrammation de l'attribut **prime**. On peut éventuellement ajouter d'autres attributs spécifiques à **Employe**.

Déclaration en Java :

```
class Employe extends Personne {
    private double prime;
}
```

- La classe fille, ne peut pas accéder aux attributs de type **private** de la classe parent.
- En fait, seules les attributs déclarées **public** ou **protected** peuvent être utilisées dans une classe héritée.

Attributs d'une classe héritée

- En plus de ces attributs, la classe fille hérite des attributs de la classe mère.

```
public class Personne {  
    private String nom;  
    private int age;  
    private double salaire;  
}
```



```
class Employe extends Personne {  
    private double prime;  
}
```

Impossible d'accéder au attribut de la classe parent.

```
public class Personne {  
    protected String nom;  
    protected int age;  
    protected double salaire;  
}
```

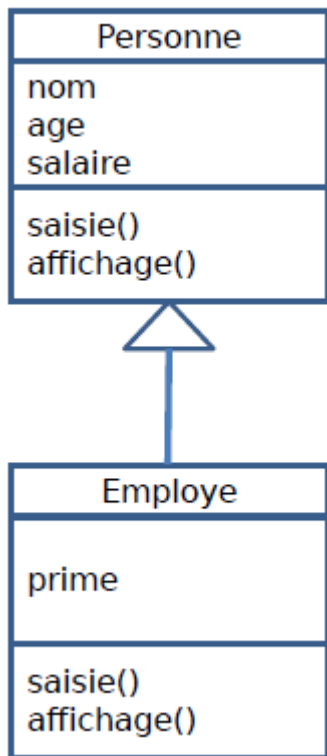


```
class Employe extends Personne {  
    private double prime;  
}
```

Les attributs de la classe parent **Personne** sont accessibles depuis la classe fille **Employe**.

Constructeurs d'une classe héritée

- On fait appel aux variables de la classe mère dans nos constructeurs grâce au mot clé **super**. Cela aura pour effet de récupérer les éléments de l'objet de base, et de les envoyer à l'objet hérité.



Le constructeur de la classe **Employe** réutilise les éléments des constructeurs de la classe **Personne**, grâce à la méthode **super**. le champ supplémentaire **prime** est reprogrammé dans le constructeur de la classe **Personne**.

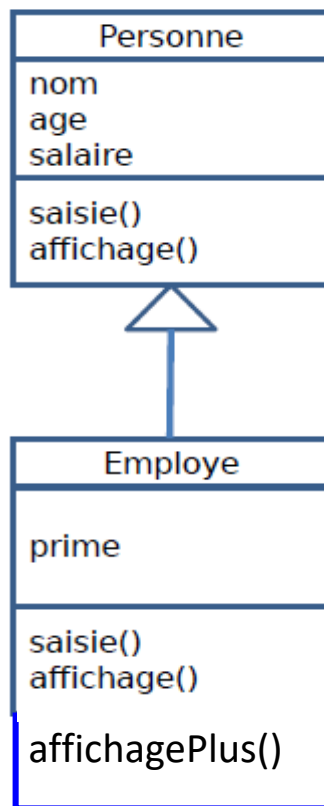
Déclaration en Java :

```
class Employe extends Personne {
    private double prime;

    // constructeur par défaut
    public void Employe() {
        super();
    }
    // constructeur avec paramètres
    public void Employe(double p) {
        super();
        this.prime = p;
    }
}
```

Méthodes d'une classe héritée

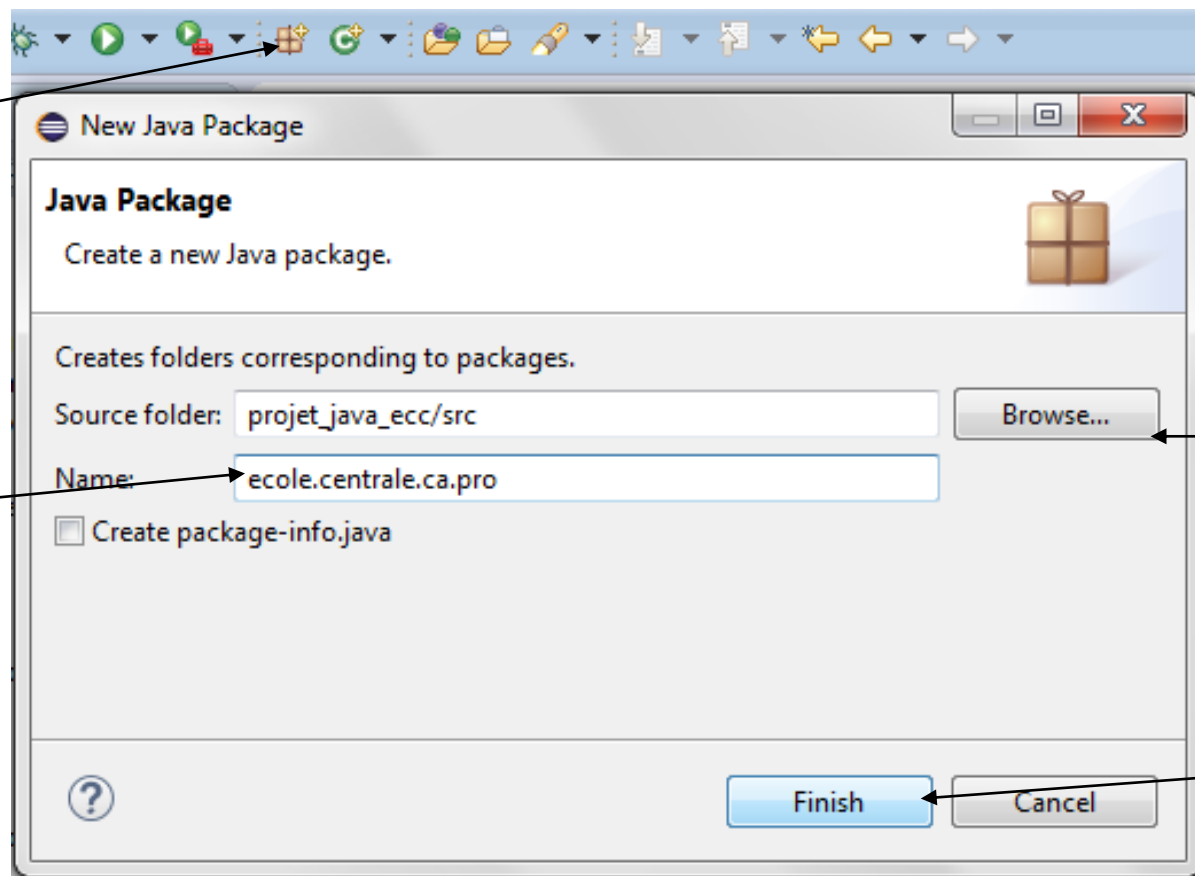
- En plus de ces méthodes, la classe fille hérite des méthodes de la classe mère.
- La classe **Employe** reprogramme une nouvelle méthode : **affichagePlus()** et utilise les méthodes de la classe **Personne** : **saisie()** et **affichage()**



- La classe fille, ne peut pas utiliser les méthodes de type **private** de la classe parent.
- Seules les méthodes déclarées **public** ou **protected** peuvent être utilisées dans une classe héritée.

Les packages

- Les packages sont des petits dossiers permettant de ranger les classes.
- Charger un package, permet d'utiliser les classes qu'il contient.
- Un des avantages des packages est qu'ils permettent de gagner en lisibilité et sont facilement transportables d'une application à l'autre.
- Création d'un nouveau package



1: Cliquer

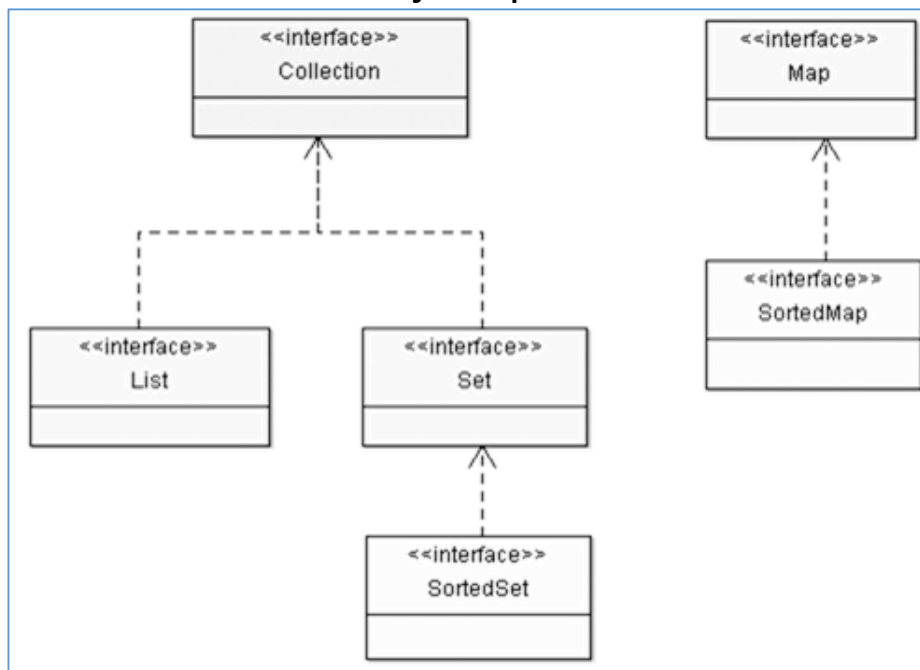
3: Saisir le nom
du package

2: Sélectionner le
projet en cours

4: Cliquer sur
« Finish »

Les collections d'objets

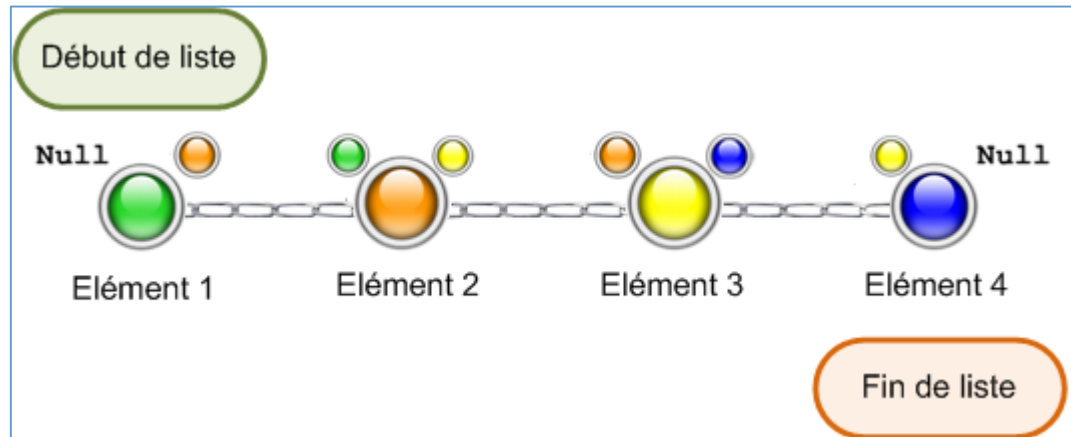
- Les collections d'objets servent à stocker ou collectionner plusieurs variables. Elles sont dynamiques et n'ont pas de taille prédéfinie. Il est donc impossible de dépasser leur capacité.
- Les objets collections sont classés en trois catégories :
 - ❑ Les objets de type **List** : servent à stocker des objets sans condition particulière sur la façon de les stocker. Ils acceptent toutes les valeurs, même les valeurs *null* ;
 - ❑ Les objets de type **Set** : sont un peu plus restrictifs, car ils n'autorisent pas deux fois la même valeur (le même objet), ce qui est pratique pour une liste d'éléments uniques ;
 - ❑ Les objets de type **Map** : sont particulières, car elles fonctionnent avec un système clé - valeur pour ranger et retrouver les objets qu'elles contiennent.



Hiérarchie d'interfaces

Les objets List : LinkedList

- L'objet **LinkedList** (liste chaînée) : est une liste dont chaque élément est lié aux éléments adjacents par une référence à ces derniers.



Fonctionnement de la LinkedList

- Les méthodes associées à l'objet **LinkedList** :

méthode	rôle
<code>add(Object element)</code>	Ajoute un élément à la liste
<code>get(int index)</code>	Retourne l'élément à l'indice demandé
<code>remove(int index)</code>	Efface l'élément à l'indice demandé
<code>isEmpty()</code>	Renvoie « True » si la l'objet liste est vide
<code>removeAll()</code>	Efface tout les éléments de l'objet liste
<code>contains(Object element)</code>	Retourne « True » si l'élément en paramètre est dans la liste

Les objets List : LinkedList

- Exemple :

```
import java.util.LinkedList;
import java.util.ListIterator;

public class Principale {

    public static void main(String[] args) {
        LinkedList list=new LinkedList();
        list.add("Ecole");
        list.add(2017);
        list.add(12.20);

        // parcours avec la boucle for
        System.out.println("-----Parcours avec la boucle for-----");
        for(int i=0;i<list.size();i++){
            System.out.println("Élément à l'index " + i + " = " + list.get(i));
        }

        // parcours avec un itérateur
        System.out.println("\n-----Parcours avec un itérateur-----");
        ListIterator iterator = list.listIterator();
        while(iterator.hasNext()){
            System.out.println(iterator.next());
        }
    }
}
```

```
Problems @ Javadoc Declaration Console Error
<terminated> Principale (1) [Java Application] C:\Program Files\Java\jre
-----Parcours avec la boucle for-----
Élément à l'index 0 = Ecole
Élément à l'index 1 = 2017
Élément à l'index 2 = 12.2
-----Parcours avec un itérateur-----
Ecole
2017
12.2
```

Les objets List : ArrayList

- L'objet **ArrayList** : est une liste d'objets n'ayant pas de taille limite et qui, en plus, acceptent n'importe quel type de données, y compris *null*.
- Contrairement aux **LinkedList**, les **ArrayList** sont rapides en lecture, même avec un gros volume d'objets. Elles sont cependant plus lentes si vous devez ajouter ou supprimer des données en milieu de liste.
- Si vous effectuez beaucoup de lectures sans vous soucier de l'ordre des éléments, optez pour une **ArrayList** ; en revanche, si vous insérez beaucoup de données au milieu de la liste, optez pour une **LinkedList**.
- Les méthodes associées à l'objet **ArrayList** :

méthode	rôle
<code>add(Object element)</code>	Ajoute un élément à la liste
<code>get(int index)</code>	Retourne l'élément à l'indice demandé
<code>remove(int index)</code>	Efface l'élément à l'indice demandé
<code>isEmpty()</code>	Renvoie « True » si la l'objet liste est vide
<code>removeAll()</code>	Efface tout les éléments de l'objet liste
<code>contains(Object element)</code>	Retourne « True » si l'élément en paramètre est dans la liste

Les objets List : ArrayList

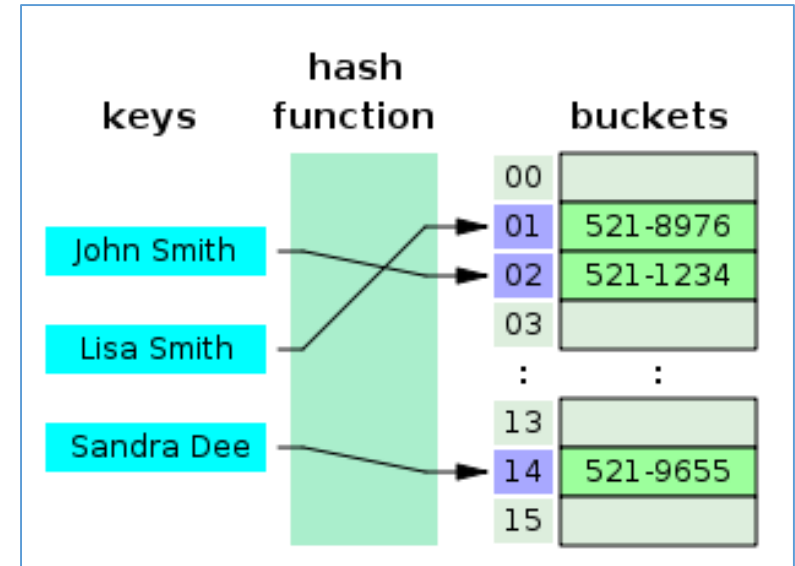
- Exemple:

```
Principale.java ✖
3 import java.util.ArrayList;
4 import java.util.ListIterator;
5
6 public class Principale {
7
8     public static void main(String[] args) {
9         ArrayList list=new ArrayList();
10        list.add("Ecole");
11        list.add(2017);
12        list.add(12.20);
13
14        // parcours avec la boucle for
15        System.out.println("-----Parcours avec la boucle for-----");
16        for(int i=0;i<list.size();i++){
17            System.out.println("Élément à l'index " + i + " = " + list.get(i));
18        }
19
20        // parcours avec un itérateur
21        System.out.println("\n-----Parcours avec un itérateur-----");
22        ListIterator iterator = list.listIterator();
23        while(iterator.hasNext()){
24            System.out.println(iterator.next());
25        }
26    }
27 }
28
29
30
31
```

```
Problems @ Javadoc Declaration Console Error
<terminated> Principale (1) [Java Application] C:\Program Files\Java\jre
-----Parcours avec la boucle for-----
Élément à l'index 0 = Ecole
Élément à l'index 1 = 2017
Élément à l'index 2 = 12.2
-----Parcours avec un itérateur-----
Ecole
2017
12.2
```

Les objets Map : Hashtable

- L'objet **Hashtable** : (table de hachage) , est une collection qui fonctionne avec un couple clé - valeur. On y trouve les objets.
- Il n'accepte pas les valeurs *null*
- Il est **Thread Safe**.
- Les méthodes associées à l'objet **Hashtable** :



méthode	rôle
put(Object key, Object value)	Ajoute un couple key-value dans la table de hachage
keys()	Retourne la liste des clés sous forme d'énumération
elements()	Retourne une énumération des éléments de la Hashtable
isEmpty()	Renvoie « True » si la Hashtable est vide
containsKey(Object Key)	Renvoie « True » si la clé passée en paramètre est dans la Hashtable.
contains(Object value)	Retourne « True » si l'élément en paramètre est dans la Hashtable

Les objets Map : Hashtable

- Exemple :

```
package ecole.centrale.ca.programme;

import java.util.Enumeration;
import java.util.Hashtable;

public class Principale {

    public static void main(String[] args) {
        Hashtable hashtab=new Hashtable();
        hashtab.put(1, "Ecole");
        hashtab.put("annee", 2017);
        hashtab.put(5, 12.20);

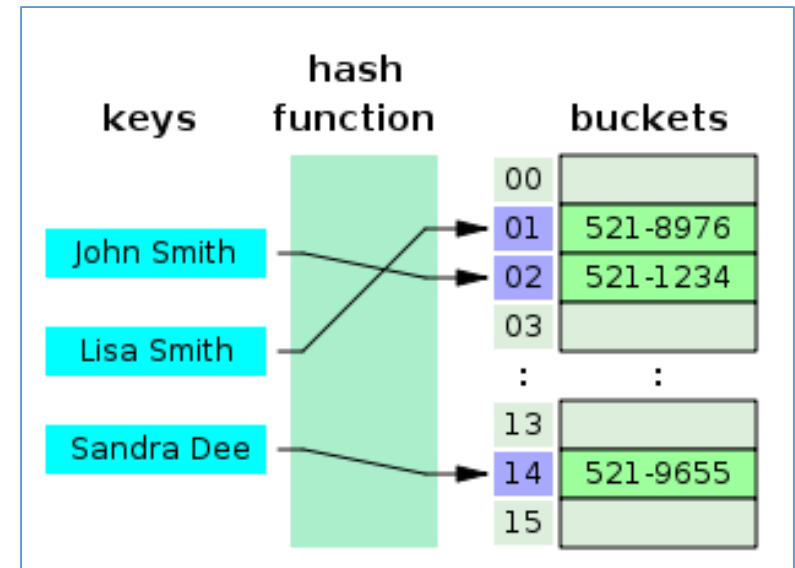
        Enumeration e = hashtab.keys();
        while(e.hasMoreElements()){
            System.out.println(e.nextElement());
        }
    }
}
```



```
Problems @ Javadoc Declaration Console ✕
<terminated> Principale (1) [Java Application] C:\Program Files
Objet à l'index = annee
Objet à l'index = 5
Objet à l'index = 1
```

Les objets Map : HashMap

- L'objet **HashMap** ne diffère que très peu de l'objet **Hashtable**.
- Il accepte les valeurs *null*
- Il n'est pas **Thread Safe**.
- Les méthodes associées à l'objet **HashMap** :



méthode	rôle
<code>put(Object key, Object value)</code>	Ajoute un couple key-value dans la table de hachage
<code>keySet()</code>	Retourne la liste des clés sous forme d'énumération
<code>values()</code>	Retourne une énumération des éléments de la HashMap
<code>isEmpty()</code>	Renvoie « True » si la HashMap est vide
<code>containsKey(Object Key)</code>	Renvoie « True » si la clé passée en paramètre est dans la HashMap.
<code>containsValue(Object value)</code>	Retourne « True » si l'élément en paramètre est dans la HashMap

Les objets Map : HashMap

- Exemple :

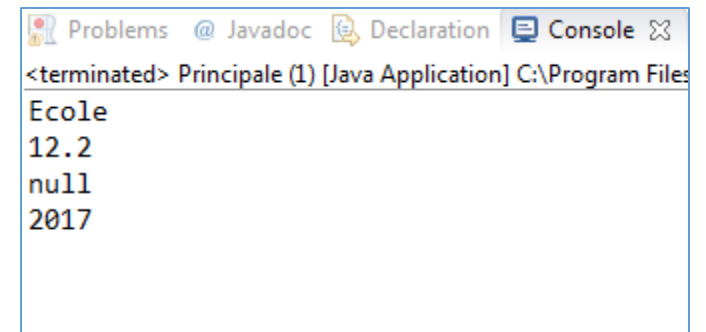
```
package ecole.centrale.ca.programme;

import java.util.HashMap;

public class Principale {

    public static void main(String[] args) {
        HashMap hashmap=new HashMap();
        hashmap.put(1, "Ecole");
        hashmap.put("annee", 2017);
        hashmap.put(5, 12.20);
        hashmap.put(6,null);

        for(Object e : hashmap.values()){
            System.out.println(e);
        }
    }
}
```



Problems @ Javadoc Declaration Console

<terminated> Principale (1) [Java Application] C:\Program Files

Ecole
12.2
null
2017



- Documentation officielle :
 - ❑ <https://docs.oracle.com/javase/7/docs/api/>
- Tutoriel Java officiel :
 - ❑ <https://docs.oracle.com/javase/tutorial/>
- Livre :
 - ❑ “Elements of Programming Interviews in Java” par Adnan Aziz, Tsung-Hsien Lee et Amit Prakash.