Formation

JAVA

m2iformation.fr

UTOPIOS
TRANSFORMATION DIGITALE INTELLIGENTE

La Programmation Orientée Objet

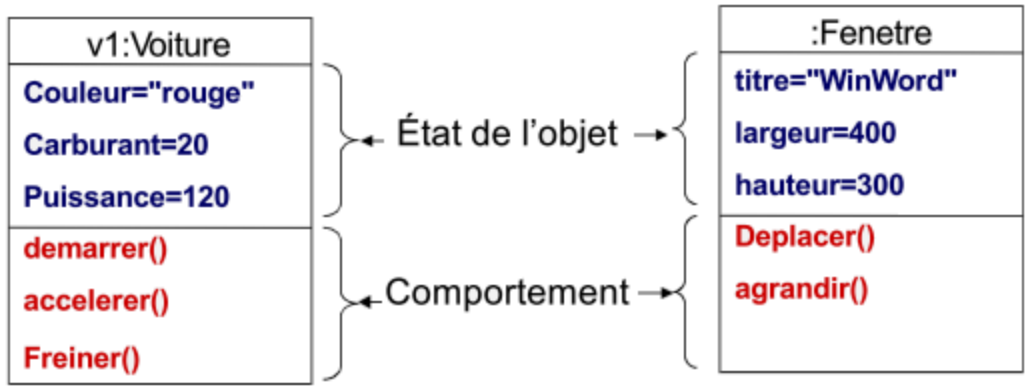
Objet/Classe – Héritage – Encapsulation - Polymorphisme

Méthode orientée objet

- La méthode orientée objet permet de concevoir une application sous la forme d'un ensemble d'objets reliés entre eux par des relations
- Lorsque que l'on programme avec cette méthode, la première question que l'on se pose plus souvent est :
  - **« qu'est-ce que je manipule ? »**
  - **Au lieu de « qu'est-ce que je fait ? »**
- L'une des caractéristiques de cette méthode permet de concevoir de nouveaux objets à partir d'objets existants.
  On peut donc réutiliser les objets dans plusieurs applications.
  La réutilisation du code fut un argument déterminant pour venter les avantages des langages à objets.
  Pour faire la programmation orientée objet il faut maitriser les fondamentaux de l'orienté objet à savoir :
  - **Objet et classe**
  - **Héritage**
  - **Encapsulation (Accessibilité)**
  - **Polymorphisme**

Objet

- Un objet est une structure informatique définie par un état et un comportement
- Objet=état + comportement
  - L'état regroupe les valeurs instantanées de tous les attributs de l'objet.
  - Le comportement regroupe toutes les compétences et décrit les actions et les réactions de l'objet. Autrement dit le comportement est défini par les opérations que l'objet peut effectuer.
- L'état d'un objet peut changer dans le temps.
- Généralement, c'est le comportement qui modifie l'état de l'objet
- Exemples :



Identité d'un objet

- En plus de son état, un objet possède une **identité** qui caractérise son existence propre.
- Cette identité s'appelle également référence ou handle de l'objet
- En terme informatique de bas niveau, l'identité d'un objet représente son adresse mémoire.
- Deux objets ne peuvent pas avoir la même identité: c'est-à-dire que deux objet ne peuvent pas avoir le même emplacement mémoire.

Classes

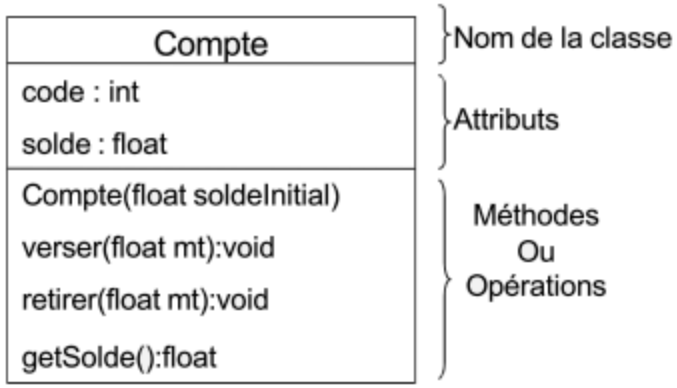
- Les objets qui ont des caractéristiques communes sont regroupés dans une entité appelé classe.
- La classe décrit le domaine de définition d'un ensemble d'objets.
- Chaque objet appartient à une classe
- Les généralités sont contenues dans les classe et les particularités dans les objets.
- Les objets informatique sont construits à partir de leur classe par un processus qui s'appelle l'instanciation.
- Tout objet est une instance d'une classe.

Caractéristique d'une classe

- Une classe est définit par:
  - Les attributs
  - Les méthodes
- Les attributs permettent de décrire l'état de des objets de cette classe.
  - Chaque attribut est défini par:
    - Son nom
    - Son type
    - Éventuellement sa valeur initiale
- Les méthodes permettent de décrire le comportement des objets de cette classe.
  - Une méthode représente une procédure ou une fonction qui permet d'exécuter un certain nombre d'instructions.
- Parmi les méthode d'une classe, existe deux méthodes particulières:
  - Une méthode qui est appelée au moment de la création d'un objet de cette classe. Cette méthode est appelée **CONSTRUCTEUR**
  - Une méthode qui est appelée au moment de la destruction d'un objet. Cette méthode s'appelle le **DESTRUCTEUR**

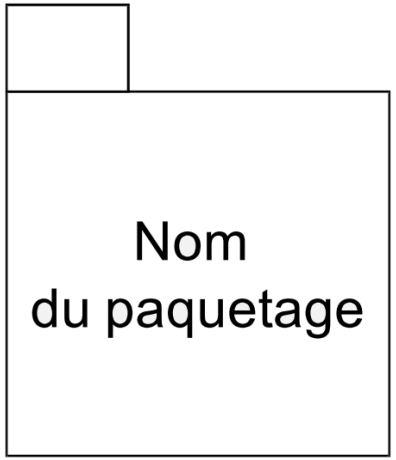
Représentation UML d'une classe

- Une classe est représenté par un rectangle à 3 compartiments :
  - Un compartiment qui contient le nom de la classe
  - Un compartiment qui contient la déclaration des attributs
  - Un compartiment qui contient les méthodes
- Exemples :



Les classes sont stockées dans des packages

- Les packages offrent un mécanisme général pour la partition des modèles et le regroupement des éléments de la modélisation
- Chaque package est représenté graphiquement par un dossier
- Les packages divisent et organisent les modèles de la même manière que les dossier organisent le système de fichier

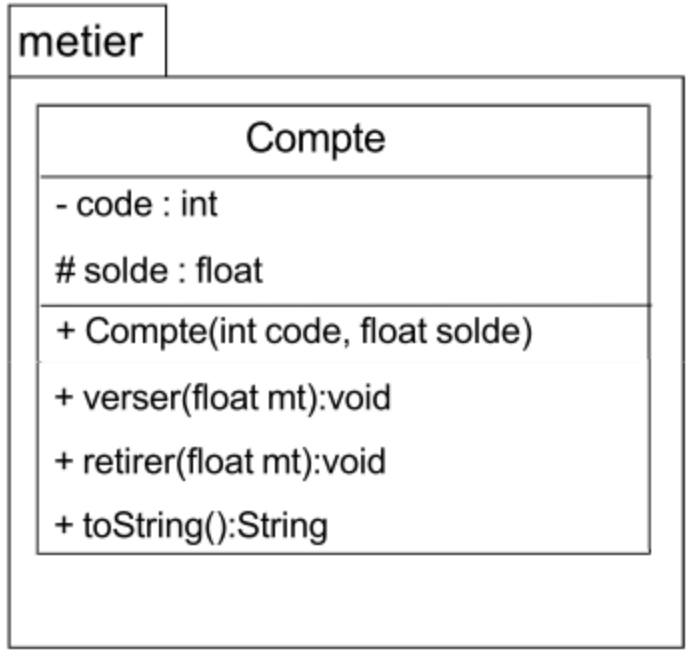


Accessibilité au membres d'une classe

- Dans java, il existe 4 **niveaux** de **protection** :
  - **private (-)** : Un membre privé d'une classe n'est accessible qu'à l'intérieur de cette classe.
  - **protected (#)** : un membre protégé d'une classe est accessible à :
    - L'intérieur de cette classe
    - Aux classes dérivées de cette classe.
  - **public (+)** : accès à partir de toute entité interne ou externe à la classe
  - **Autorisation par défaut** : dans java, en l'absence des trois autorisations précédentes, l'autorisation par défaut est **package**. Cette autorisation indique que uniquement les classes du même package ont l'autorisation d'accès.

Exemple d'implémentation d'une classe

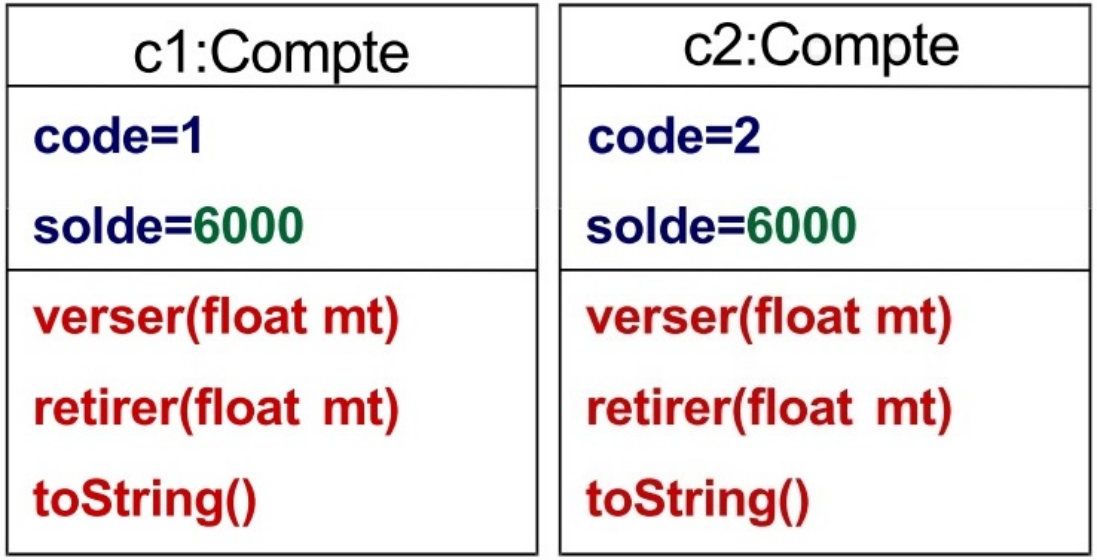
avec JAVA




```java
package metier;

public class Compte {
  // Attributs
  private int code;
  protected float solde;
  // Constructeur
  public Compte( int c,float s){
    code=c;
    solde=s;
  }
  // Méthode pour verser un montant
  public void verser(float mt){
    solde+=mt;
  }
  // Méthode pour retirer un montant
  public void retirer(float mt){
    solde-=mt;
  }
  // Une méthode qui retourne l'état du compte
  public String toString(){
    return(" Code = "+code+" Solde = "+solde)
  }
}
```

Création des objets dans java

- Dans java, pour créer un objet d'une classe , On utilise la commande new suivie du constructeur de la classe.
- La commande new Crée un objet dans l'espace mémoire et retourne l'adresse mémoire de celui-ci.
- Cette adresse mémoire devrait être affectée à une variable qui représente l'identité de l'objet. Cette référence est appelée handle.




```java
package metier;

import metier.compte

public class Application {

  public static void main(String[] args){

    Compte c1=new Compte(1,5000);

    Compte c2=new Compte(2,6000);

    c1.verser(3000);

    c1.retirer(2000);

    System.out.println(c1.toString());
  }
}
```

Constructeur par défaut

- Quand on ne définit aucun constructeur pour une classe, le compilateur crée le constructeur par défaut.
- Le constructeur par défaut n'a aucun paramètre et ne fait aucune initialisation

```java
// Exemple de classe :
public class Personne {
  // Les Attributs
  private int code;
  private String nom;
  // Les Méthodes
  public void setNom(String n){
    this.nom = n;
  }
  public String getNom(){
    return nom;
  }
}
```

Instanciation en utilisant le constructeur par défaut :

```java
Personne P = new Personne();
p.setNom("AZER");
System.out.println(p.getNom());
```

Getters et Setters

- Les attributs privés d'une classe ne sont accessibles qu'à l'intérieur de la classe.
- Pour donner la possibilité à d'autres classes d'accéder aux membres privés, il faut définir dans la classes des méthodes publiques qui permettent de :
  - lire la variables privés. Ce genre de méthodes s'appellent les **accesseurs** ou **Getters**
  - modifier les variables privés. Ce genre de méthodes s'appellent les **mutateurs** ou **Setters**
- Les **getters** sont des méthodes qui commencent toujours par le mot **get** et finissent par le nom de l'attribut en écrivant en majuscule la lettre qui vient juste après le get. Les getters retourne toujours le même type que l'attribut correspondant.
  - Par exemple, dans la classe CompteSimple, nous avons défini un attribut privé :
  - `private String nom;`
  - Le getter de cette variable est :
  - ```
    public String getNom( ){
    return nom;}
    ```
- Les **setters** sont des méthodes qui commencent toujours par le mot **set** et finissent par le nom de l'attribut en écrivant en majuscule la lettre qui vient juste après le set. Les setters sont toujours de type void et reçoivent un paramètre qui est de meme type que la variable:

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

Encapsulation

```
public class Application {
  public static void main(String[] args) {
    Personne p=new Personne();
    p.setNom("AZER");
    System.out.println(p.getNom());
  }
}
```

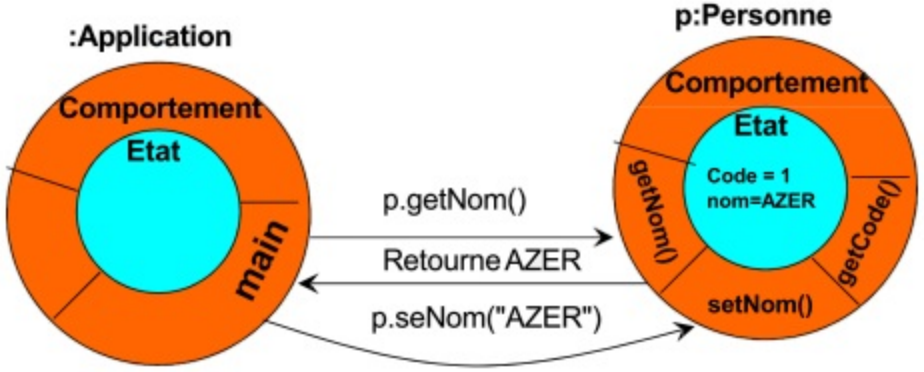
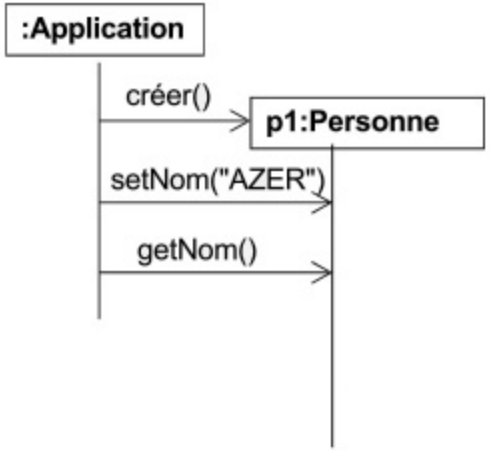


Diagramme de séquence :



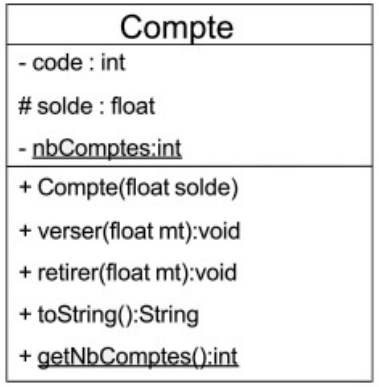
- Généralement, l'état d'un objet est privé ou protégé et son comportement est publique
- Quand l'état de l'objet est privé Seules ses méthodes ont le droit d'y accéder
- Quand l'état de l'objet est protégé, les méthodes des classes dérivées.

Membres statiques d'une classe.

- Dans l'exemple de la classe Compte, chaque objet Compte possède ses propres variables code et solde. Les variables code et solde sont appelées variables d'instances.
- Les objets d'une même classe peuvent partager des mêmes variables qui sont stockées au niveau de la classe. Ce genre de variables, s'appellent les variables statiques ou variables de classes.
- Un attribut statique d'une classe est un attribut qui appartient à la classe et partagé par tous les objets de cette classe.
- Comme un attribut une méthode peut être déclarée statique, ce qui signifie qu'elle appartient à la classe et partagée par toutes les instances de cette classe.
- Dans la notation UML, les membres statiques d'une classe sont soulignés.

Exemple

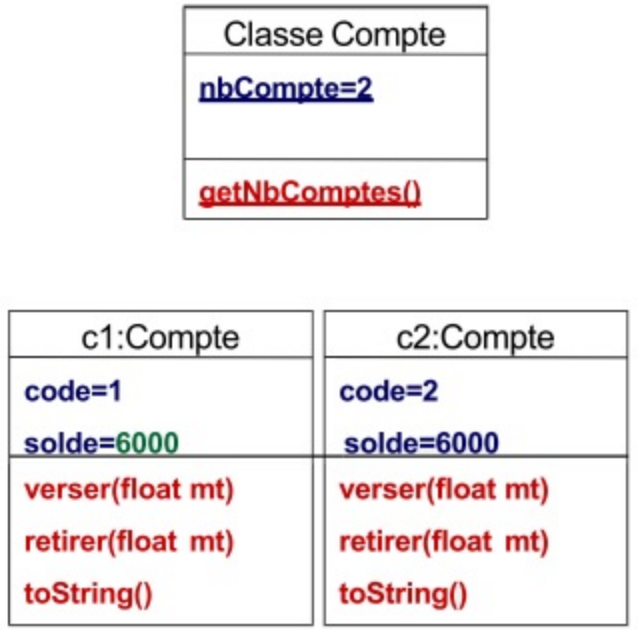
- Supposant que nous voulions ajouter à la classe Compte une variable qui permet de stocker le nombre de comptes créés.
- Comme la valeur de variable nbComptes est la même pour tous les objets, celle-ci sera déclarée statique. Si non, elle sera dupliquée dans chaque nouveau objet créé.
- La valeur de nbComptes est au départ initialisée à 0, et pendant la création d'une nouvelle instance (au niveau du constructeur), nbCompte est incrémentée et on profite de la valeur de nbComptes pour initialiser le code du compte.




```java
package metier;
public class Compte{
  // Variables d'instances
  private int code;
  private float solde;
  // Variables de classe ou statique
  private static int nbComptes;
  // Constructeur
  public Compte(float solde){
    this.code = ++nbComptes;
    this.solde = solde;
  }
  // Méthode pour verser un montant
  public void verser(float mt){
    solde += mt;
  }
  // Méthode pour retirer un montant
  public void retirer(flaot mt){
    solde -= mt;
  }
  // retourne l'état du compte
  public String toString(){
    return(" Code="+code+" Solde="+solde);
  }
  // retourne la valeur de nbComptes
  public static int getNbComptes(){
    return nbComptes;
  }
}
```

Application de test

```
package test;
import metier.Compte;

public class Application  {

  public static void main(String[] args) {
    Compte c1=new  Compte(5000);
    Compte c2=new Compte(6000);
    c1.verser(3000);
    c1.retirer(2000);
    System.out.println(c1.toString());
    System.out.println(Compte.nbComptes)
    System.out.println(c1.nbComptes)
  }

}
```



Héritage

- Dans la programmation orientée objet, l'héritage offre un moyen très efficace qui permet la réutilisation du code.
- En effet une classe peut hériter d'une autre classe des attributs et des méthodes.
- L'héritage, quand il peut être exploité, fait gagner beaucoup de temps en terme de développement et en terme de maintenance des applications.
- La réutilisation du code fut un argument déterminant pour venter les méthodes orientées objets.

UTOPIOS
TRANSFORMATION DIGITALE INTELLIGENTE

Exercice(s)

Exercice : La chaise

1. Créer une classe Chaise possédant comme variables d'instance le nombre de pieds, le matériaux et la couleur de l'objet
2. Afficher ses informations en surchargeant une méthode de la classe Object
3. La classe Chaise pourra être instanciée avec ou sans paramètres (Constructeur par défaut)
4. Afficher toutes les chaises (Possibilité de simplifier avec une méthode ToString)

```
Je suis une chaise avec 4 pied(s) en chaine de couleur bleu à un prix de 14.4
Je suis une chaise avec 3 pied(s) en bambou de couleur gris à un prix de 130.99
Je suis une chaise avec 1 pied(s) en métal de couleur orange à un prix de 76.25
```

Exercice : Film

1. Créer une classe "Film"
2. Ajouter les attributs suivants : titre, réalisateur, année de sortie et genre
3. Ajouter un constructeur, des getters et des setters pour ces attributs, ainsi qu'une méthode pour afficher les informations sur le film

```
Film{titre='La Haine', realisateur='Mathieu Kassovitz', dateSortie=1995-05-31, genre='Drame'}
Film{titre='Avatar 2', realisateur='James Cameron', dateSortie=2022-12-14, genre='Action'}
```

Exercice : Joueur

1. Créer une classe Joueur
2. Ajouter les attributs suivants : nom, niveau et points d'expérience
3. Ajouter un constructeur, des getters et des setters pour ces attributs
4. Créer une méthode effectuerUneQuete() qui ajoute 10 points d'expérience au joueur
5. Créer qu'une méthode qui augmente le niveau du joueur de +1 si son expérience dépasse 100 points et réinitialise son expérience
6. Tester le programme

Exemple exercice : Joueur:

```
Le joueur WarriorDu59 effectue la quête n° 1
Le joueur WarriorDu59 effectue la quête n° 2
Le joueur WarriorDu59 effectue la quête n° 3
Le joueur WarriorDu59 effectue la quête n° 4
Le joueur WarriorDu59 effectue la quête n° 5
Le joueur WarriorDu59 effectue la quête n° 6
Le joueur WarriorDu59 effectue la quête n° 7
Le joueur WarriorDu59 effectue la quête n° 8
Le joueur WarriorDu59 effectue la quête n° 9
Le joueur WarriorDu59 effectue la quête n° 10
Le joueur WarriorDu59 passe au niveau :2
Le joueur WarriorDu59 effectue la quête n° 11
Le joueur WarriorDu59 effectue la quête n° 12
Le joueur WarriorDu59 effectue la quête n° 13
Le joueur WarriorDu59 effectue la quête n° 14
Le joueur WarriorDu59 effectue la quête n° 15
Le joueur WarriorDu59 effectue la quête n° 16
Le joueur WarriorDu59 effectue la quête n° 17
Le joueur WarriorDu59 effectue la quête n° 18
Le joueur WarriorDu59 effectue la quête n° 19
Le joueur WarriorDu59 effectue la quête n° 20
Le joueur WarriorDu59 passe au niveau :3
WarriorDu59 change de pseudo : LeGigaBossDuJava
```

Exercice : WaterTank

1. Créer une classe WaterTank
2. Ajouter les attributs suivants: poids à vide, capacité maximale, niveau de remplissage
3. Créer une méthode remplir() qui ajoutera une volume d'eau à la citerne
4. Créer une méthode vider() qui enlèvera un volume d'eau à la citerne
5. Créer un attribut de classe qui contiendra la totalité des volumes d'eau de la citerne
6. /!\ le volume d'eau d'une citerne ne peut pas être négatif ou dépasser le volume maximum

Exemple : exercice WaterTank

```
WaterTank 1 volume de départ : 10.0
WaterTank 2 volume de départ : 25.0
Volume total des WaterTank : 35.0
Volume total des WaterTank : 40.0
Volume total des WaterTank : 100.0
Volume total des WaterTank : 95.0
Volume total des WaterTank : 75.0
WaterTank 1 volume de départ : 75.0
WaterTank 2 volume de départ : 0.0
Poids total de la citerne 1 : 175.0Kg
Poids total de la citerne 2 : 70.0Kg
```

Exercice : Plante

1. Créer une classe "Plante" avec les attributs suivants : nom, hauteur et couleur des feuilles
2. Ajouter un constructeur, des getters et des setters pour ces attributs
3. Créer une méthode pour afficher les informations sur la plante
4. Créer une classe "Arbre" qui hérite de "Plante" et qui ajoute un attribut supplémentaire pour la circonférence du tronc
5. Surcharger la méthode afficher pour qu'elle ajoute les informations à propos de l'arbre

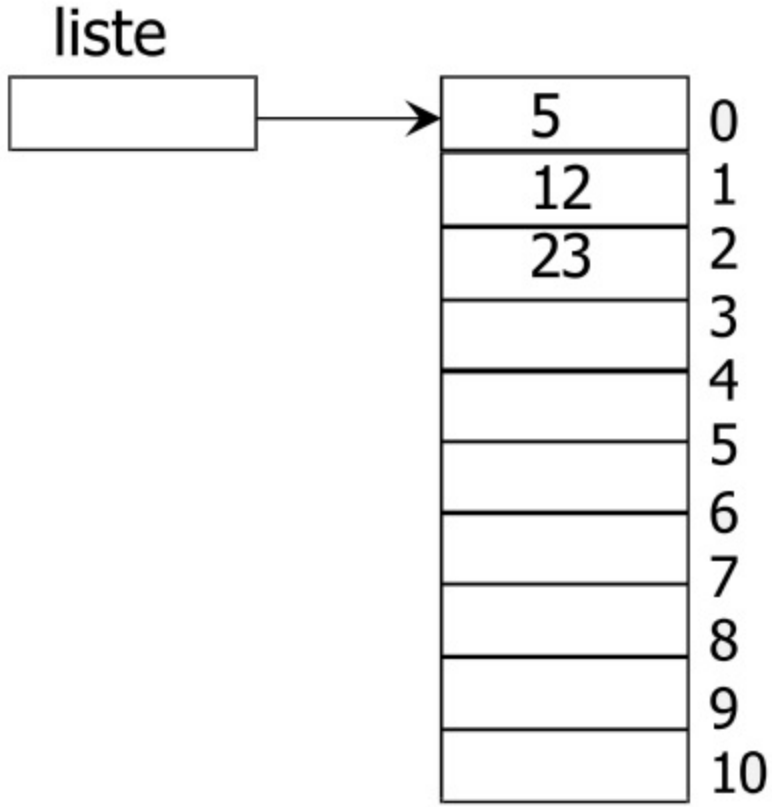
UTOPIOS
TRANSFORMATION DIGITALE INTELLIGENTE

Les Collections

Tableaux de primitives

- Tableaux de primitives:
  - Déclaration :
    - Exemple : Tableau de nombres entiers
      - `int[] liste;`
    - liste est un handle destiné à pointer vers un tableau d'entier
  - Création du tableau
    - `liste = new int[11];`
  - Manipulation des éléments du tableau:

```
liste[0]=5; liste[1]=12; liste[3]=23;
for(int  i=0;i<liste.length;i++){
System.out.println(liste[i]);
}
```

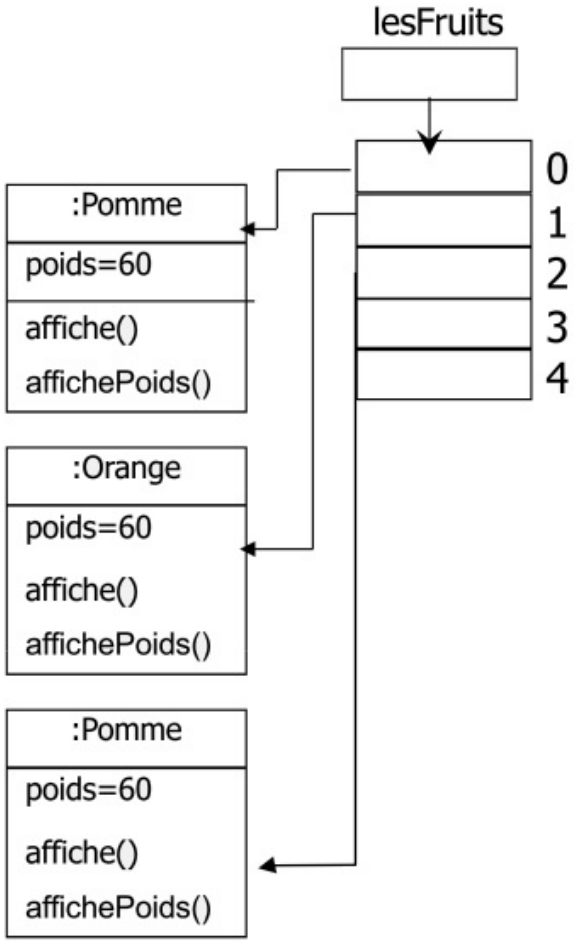


Tableaux d'objets

- Déclaration :
  - Exemple : Tableau d'objets Fruit
  - `Fruit[] lesFruits;`
- Création du tableau
  - lesFruits = new Fruit[5];
- Création des objets :
  - `lesFruits[0]=new Pomme(60);`
  - `lesFruits[1]=new Orange(100);`
  - `lesFruits[2]=new Pomme(55);`
- Manipulation des objets :

```
for(int i=0;i<lesFruits.length;i++){
  lesFruits[i].affiche();
  if(lesFruits[i] instanceOf Pomme)
  ((Pomme)lesFruits[i]).affichePoids();
  else
  ((Orange)lesFruits[i]).affichePoids();
  }
```

- Un tableau d'objets est un tableau de handles



Collections

Java propose l'API Collections qui offre un socle riche et des implémentations d'objets de type collection enrichies au fur et à mesure des versions de Java.

L'API Collections possède deux grandes familles chacune définies par une interface :

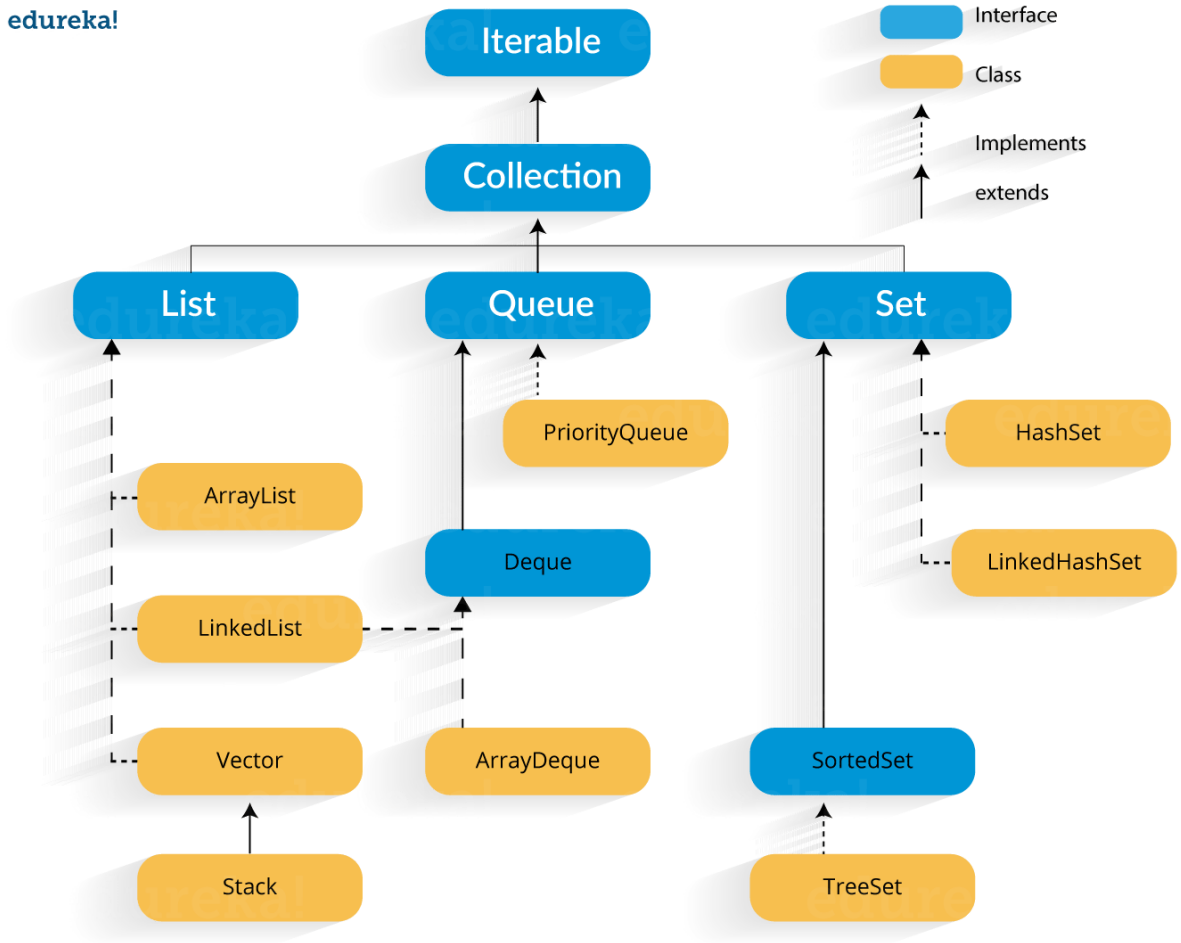
- **java.util.Collection** : pour gérer un groupe d'objets
- **java.util.Map** : pour gérer des éléments de type paires de clé/valeur

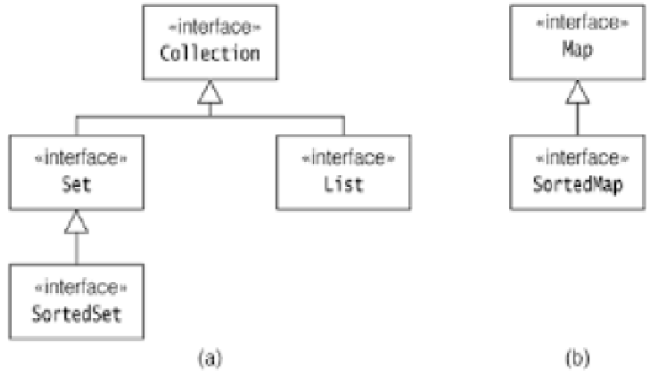
Collections

l'API Collections définit enfin :

- Deux interfaces pour le parcours de certaines collections : Iterator et ListIterator.
- Une interface et une classe pour permettre le tri de certaines collections : Comparable et Comparator
- Des classes utilitaires : Arrays, Collections

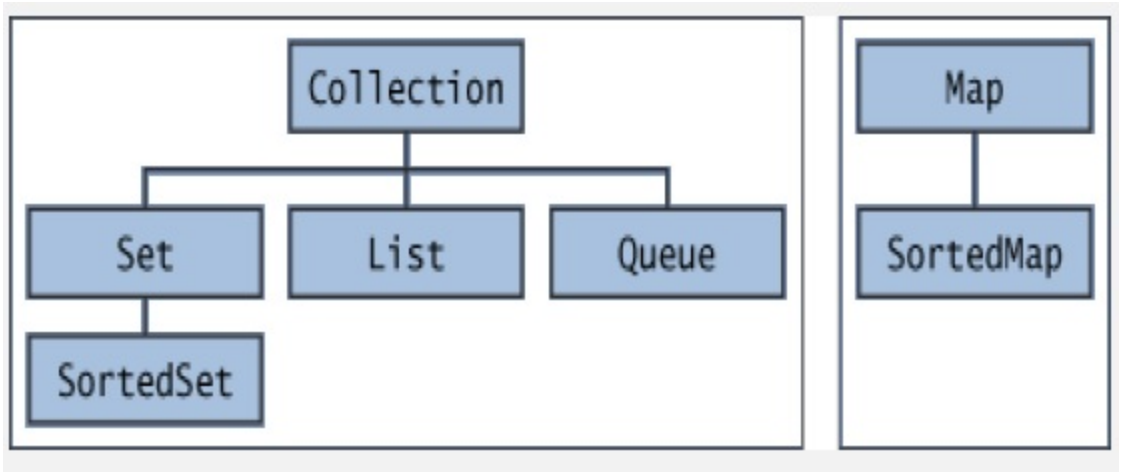
Collections





Collection interfaces

- Les interfaces de collection principales encapsulent différents types de collections. Ils représentent les types de données abstraites qui font partie de l'infrastructure de collections. Ce sont des interfaces donc elles ne fournissent pas d'implémentation !



Collections

- Une collection est un tableau dynamique d'objets de type Object.
- Une collection fournit un ensemble de méthodes qui permettent :
  - D'ajouter un nouveau objet dans le tableau
  - Supprimer un objet du tableau
  - Rechercher des objets selon des critères
  - Trier le tableau d'objets
  - Contrôler les objets du tableau
  - Etc…
- Dans un problème, les tableaux peuvent être utilisés quand la dimension du tableau est fixe.
- **Dans le cas contraire, il faut utiliser les collections**
- Java fournit plusieurs types de collections :
  - ArrayList
  - Vector
  - Iterator
  - HashMap
  - Etc…
- Dans cette partie du cours, nous allons présenter uniquement comment utiliser les collections ArrayList, Vector, Iterator et HashMap

Collections

- Java fournit plusieurs types de collections:
  - ArrayList
  - Vector
  - Iterator
  - HashMap
  - Etc...
- **Il existe trois composants qui étendent l'interface de Collection, à savoir la List, Queue et les Map.**

Collections

Collection	Ordonné	Accès direct	Clé / valeur	Doublons	Null	Thread Safe
ArrayList	Oui	Oui	Non	Oui	Oui	Non
LinkedList	Oui	Non	Non	Oui	Oui	Non
HashSet	Non	Non	Non	Non	Oui	Non
TreeSet	Oui	Non	Non	Non	Non	Non
HashMap	Non	Oui	Oui	Non	Oui	Non
TreeMap	Oui	Oui	Oui	Non	Non	Non
Vector	Oui	Oui	Non	Oui	Oui	Oui
Hashtable	Non	Oui	Oui	Non	Non	Oui
Properties	Non	Oui	Oui	Non	Non	Oui
Stack	Oui	Non	Non	Oui	Oui	Oui
CopyOnWriteArrayList	Oui	Oui	Non	Oui	Oui	Oui
ConcurrentHashMap	Non	Oui	Oui	Non	Non	Oui
CopyOnWriteArraySet	Non	Non	Non	Non	Oui	Oui

public interface Collection<E> extends Iterable<E>

```java
public interface Collection<E> extends Iterable<E> {
    // Basic operation
    int size();
    boolean isEmpty();
    boolean contains(Object element);
    boolean add(E element);          //optional
    boolean remove(Object element); //optional
    Iterator<E> iterator();

    // Bulk operations
    boolean containsAll(Collection<?> c);
    boolean addAll(Collection<? extends E> c);  //optional
    boolean removeAll(Collection<?> c);         //optional
    boolean retainAll(Collection<?> c);         //optional
    void clear();                               //optional

    // Array operations
    Object[] toArray();
    <T> T[] toArray(T[] a);
}
```

Iterable

Il existe trois manières d'itérer les objets de Iterable.

1. Utilisation de la boucle for améliorée (boucle for-each)
2. Utilisation de la boucle Iterable forEach
3. Utilisation de l' interface Iterator

Iterator

- **Iterator** est une interface qui itère les éléments. Il permet de parcourir la liste et de modifier les éléments.
- **L'interface Iterator a trois méthodes qui sont mentionnées ci-dessous :**
  1. **public boolean hasNext()** — Cette méthode renvoie true si l'itérateur a plus d'éléments.
  2. **public object next()** — Il renvoie l'élément et déplace le pointeur du curseur vers l'élément suivant.
  3. **public void remove()** — Cette méthode supprime les derniers éléments renvoyés par l'itérateur.

Iterator

- La collection de type Iterator du package java.util est souvent utilisée pour afficher les objets d'une autre collection
- En effet il est possible d'obtenir un iterator à partir de chaque collection.
- Exemple :
  - Création d'un vecteur de Fruit.
  - `Vector<Fruit> fruits=new Vector<Fruit>();`
  - Ajouter des fruits aux vecteur
  - `fruits.add(new Pomme(30));`
  - `fruits.add(new Orange(25));`
  - `fruits.add(new Pomme(60));`
  - Création d'un Iterator à partir de ce vecteur
  - `Iterator<Fruit> it=fruits.iterator();`
  - Parcourir l'Iterator:

```
while(it.hasNext()){
  Fruit f=it.next();
  f.affiche();
  }
```

UTOPIOS
TRANSFORMATION DIGITALE INTELLIGENTE

List

public interface List<E> extends Collection<E>

List — une collection ordonnée (parfois appelée séquence). Les listes peuvent contenir des éléments en double. L'utilisateur d'une liste a généralement un contrôle précis sur l'endroit où chaque élément est inséré dans la liste et peut accéder aux éléments par leur index entier (position). Si vous avez utilisé Vector, vous connaissez la saveur générale de List.

Collection ArrayList

- ArrayList est une classe du package java.util, qui implémente l'interface List.
- Déclaration d'une collection de type List qui devrait stocker des objets de type Fruit:
  - `List<Fruit> fruits;`
- Création de la liste:
  - `fruits=new ArrayList<Fruit>();`
- Ajouter deux objets de type Fruit à la liste:
  - `fruits.add(new Pomme(30));`
  - `fruits.add(new Orange(25));`
- Faire appel à la méthode affiche() de tous les objets de la liste:
  - En utilisant la boucle classique for

```
for(int i=0;i<fruits.size();i++){
    fruits.get(i).affiche();
}
```

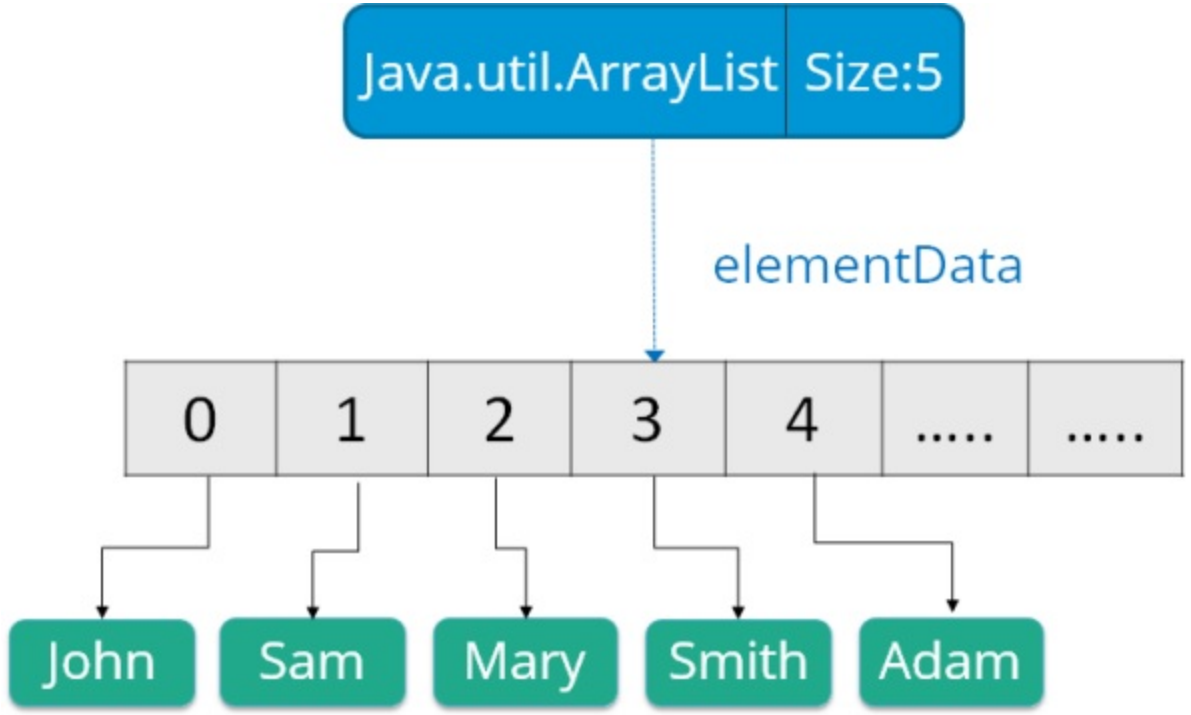
  - En utilisant la boucle for each

```
for(Fruit f:fruits)
  f.affiche();
```

- Supprimer le deuxième Objet de la liste
  - `fruits.remove(1);`

Exemple d'utilisation de ArrayList

```java
import java.util.ArrayList;
import java.util.List;

public class App1 {
    public static void main(String[] args) {
        // Déclaration d'une liste de type Fruit
        List<Fruit> fruits;
        // Création de la liste
        fruits = new ArrayList<Fruit>();
        // Ajout de 3 objets Pomme,Orange et Pomme à la liste
        fruits.add(new Pomme(30));
        fruits.add(new Orange(25));
        fruits.add(new Pomme(60));
        // Parcourir tous les objets
        for (int i= 0;i<fruits.size();i++){
            //Faire appel à la méthode affiche() de chaque Fruit de la liste
            fruits.get(i).affiche();
        }
        // Une autre maniére plus simple pour parcourir une liste
        for(Fruit f:fruits){ // Pour chaque Fruit de la lste
            f.affiche(); // Faire appel à la méthode affiche() du Fruit f
        }
    }
}
```



Collection Vector

- Vector est une classe du package java.util qui fonctionne comme ArrayList
- Déclaration d'un Vecteur qui devrait stocker des objets de type Fruit:
  - `Vector<Fruit> fruits;`
- Création de la liste:
  - `fruits=new Vector<Fruit>();`
- Ajouter deux objets de type Fruit à la liste:
  - `fruits.add(new Pomme(30));`
  - `fruits.add(new Orange(25));`
- Faire appel à la méthode affiche() de tous les objets de la liste:
  - En utilisant la boucle classique for

```
for(int i=0;i<fruits.size();i++){
    fruits.get(i).affiche();
}
```

  - En utilisant la boucle for each

```
for(Fruit f:fruits)
  f.affiche();
```

- Supprimer le deuxième Objet de la liste
  - `fruits.remove(1);`

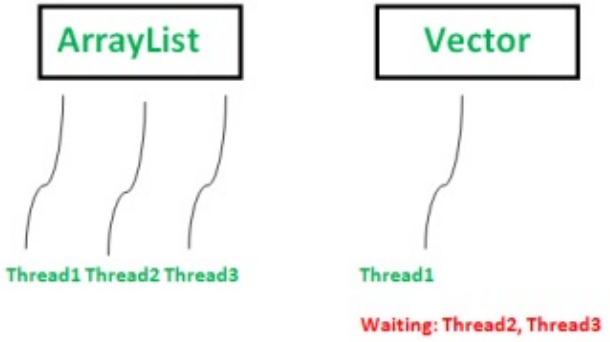
Exemple d'utilisation de Vector

```
import java.util.Vector;

public class App2 {
    public static void main(String[] args) {
        // Déclaration d'un vecteur de type Fruit
        Vector<Fruit> fruits;
        // Création du vecteur
        fruits = new Vector<Fruit>();
        // Ajout de 3 objets Pomme, Orange et Pomme au vecteur
        fruits.add(new Pomme(30));
        fruits.add(new Orange(25));
        fruits.add(new Pomme(60));
        // Parcourir tous les objets
        for (int i=0;i< fruits.size();i++){
            // Faire appel à la méthode affiche() de chaque Fruit
            fruits.get(i).affiche();
        }
        // Une autre maniére plus simple pour parcourir un vecteur
        for (Fruit f:fruits) { // Pour chaque Fruit du vecteur
            f.affiche(); // Faire appel à la méthode affiche() du Fruit f
        }
    }
}
```

ArrayList vs Vector

N°	ArrayList	Vector
1.	ArrayList n'est pas synchronisé.	Le vecteur est synchronisé.
2.	ArrayList incrémente 50 % de la taille actuelle du tableau si le nombre d'éléments dépasse sa capacité.	Les incréments vectoriels de 100 % signifient que la taille du tableau double si le nombre total d'éléments dépasse sa capacité.
3.	ArrayList n'est pas une classe héritée. Il est introduit dans JDK 1.2.	Vector est une classe héritée.
4.	ArrayList est rapide car non synchronisé.	Vector est lent parce qu'il est synchronisé, c'est-à-dire que dans un environnement multithreading, il maintient les autres threads dans un état exécutable ou non jusqu'à ce que le thread actuel libère le verrou de l'objet.
5.	ArrayList utilise l'interface Iterator pour parcourir les éléments.	Un vecteur peut utiliser l'interface Iterator ou l'interface Enumeration pour parcourir les éléments.

ArrayList vs Vector



Performances	ArrayList est plus rapide. Comme il n'est pas synchronisé, alors que les opérations vectorielles ralentissent les performances car elles sont synchronisées (thread-safe), si un thread travaille sur un vecteur, il a acquis un verrou sur celui-ci, ce qui oblige tout autre thread voulant travailler dessus à doivent attendre que le verrou soit libéré.
Croissance des données	ArrayList et Vector **augmentent et diminuent de manière dynamique** pour maintenir une utilisation optimale du stockage, mais la façon dont ils se redimensionnent est différente. ArrayList incrémente 50 % de la taille actuelle du tableau si le nombre d'éléments dépasse sa capacité, tandis que vector incrémente 100 % - doublant essentiellement la taille actuelle du tableau.
Traversal	Vector peut utiliser à la fois Enumeration et Iterator pour parcourir les éléments vectoriels, tandis que ArrayList ne peut utiliser que **Iterator** pour parcourir.
Applications	Vector peut utiliser à la fois Enumeration et Iterator pour parcourir les éléments vectoriels, tandis que ArrayList ne peut utiliser que **Iterator** pour parcourir.

public interface List<E> extends Collection<E>

```
public interface List<E> extends Collection<E> {
    // Positional access
    E get(int index);
    E set(int index, E element);    //optional
    boolean add(E element);         //optional
    void add (int index, E element);//optional
    E remove(int index);            //optional
    boolean addAll(int index,Collection<? extends E> c);    //optional

    // Search
    int indexOf(Object o);
    int lastIndexOf(Object o);

    // Iteration
    ListIterator<E> listIterator();
    ListIterator<E> listIterator(int index);

    // Range-view
    List<E> subList(int from, int to);
}
```

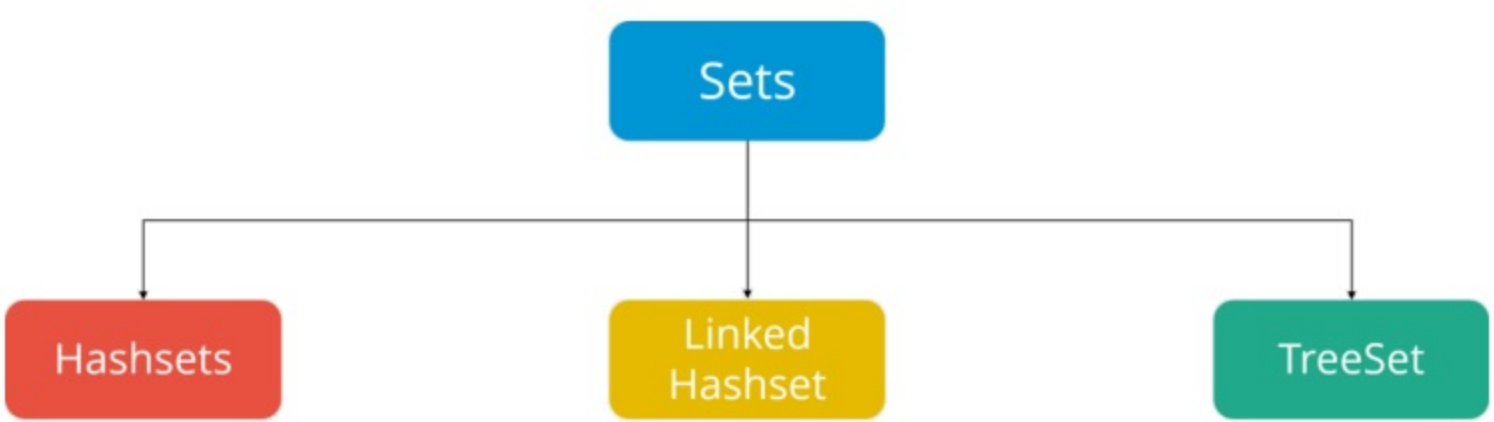
UTOPIOS
TRANSFORMATION DIGITALE INTELLIGENTE

Set

public interface Set<E> extends Collection<E>

Set — une collection qui ne peut pas contenir d'éléments en double.Cette interface modélise l'abstraction des ensembles mathématiques et est utilisée pour représenter des ensembles, tels que les cartes comprenant une main de poker, les cours constituant l'emploi du temps d'un étudiant ou les processus exécutés sur une machine.

Set<E> extends Collection<E>



Set<E> extends Collection<E>

- La classe Java HashSet crée une collection qui utilise une table de hachage pour le stockage.
- HashSet ne contient que des éléments uniques et hérite de la classe AbstractSet et implémente l'interface Set.
- Aussi, il utilise un mécanisme de hachage pour stocker les éléments.

Set<E> extends Collection<E>

Voici quelques-unes des méthodes de la classe Java HashSet :

Method	Description
boolean add(Object o)	Adds the specified element to this set if it is not already present.
boolean contains(Object o)	Returns true if the set contains the specified element.
void clear()	Removes all the elements from the set.
boolean isEmpty()	Returns true if the set contains no elements.
boolean remove(Object o)	Remove the specified element from the set.
Object clone()	Returns a shallow copy of the HashSet instance: the elements themselves are not cloned.
Iterator iterator()	Returns an iterator over the elements in this set.
int size()	Return the number of elements in the set.

public interface Set<E> extends Collection<E>

```
public interface Set<E> extends Collection<E>{
    //Basic operations
    int size();
    boolean isEmpty();
    boolean contains(Object element);
    boolean add(E element);              //optional
    boolean remove(Object element);      //optional
    Iterator<E> iterator();
    //Bulk operations
    boolean containsAll(Collection<?> c);
    boolean addAll(Collection<? extends E> c);  //optional
    boolean removeAll(Collection<?> c);         //optional
    boolean retainAll(Collection<?> c);         //optional
    void clear();                               //optional
    // Array Operations
    Object[] toArray();
    <T> T[] toArray(T[] a);
}
```

Remarque : rien n'a été ajouté à l'interface de collecte , sauf aucun doublon autorisé

public interface SortedSet<E> extends Set<E>

```
public interface SortedSet<E> extends Set<E> {
    // Range-view
    SortedSet<E> subSet(E fromElement, E toElement);
    SortedSet<E> headSet(E toElement);
    SortedSet<E> tailSet(E fromElement);

    // Endpoints
    E first();
    E last();

    // Comparator access
    Comparator<? super E> comparator();
}
```

public interface SortedSet<E> extends Set<E>

- **SortedSet** — un ensemble qui maintient ses éléments dans l'ordre croissant. Plusieurs opérations supplémentaires sont prévues pour profiter de la commande.
- Les ensembles triés sont utilisés pour les ensembles ordonnés naturellement, tels que les listes de mots et les rouleaux d'appartenance.

UTOPIOS
TRANSFORMATION DIGITALE INTELLIGENTE

Queue

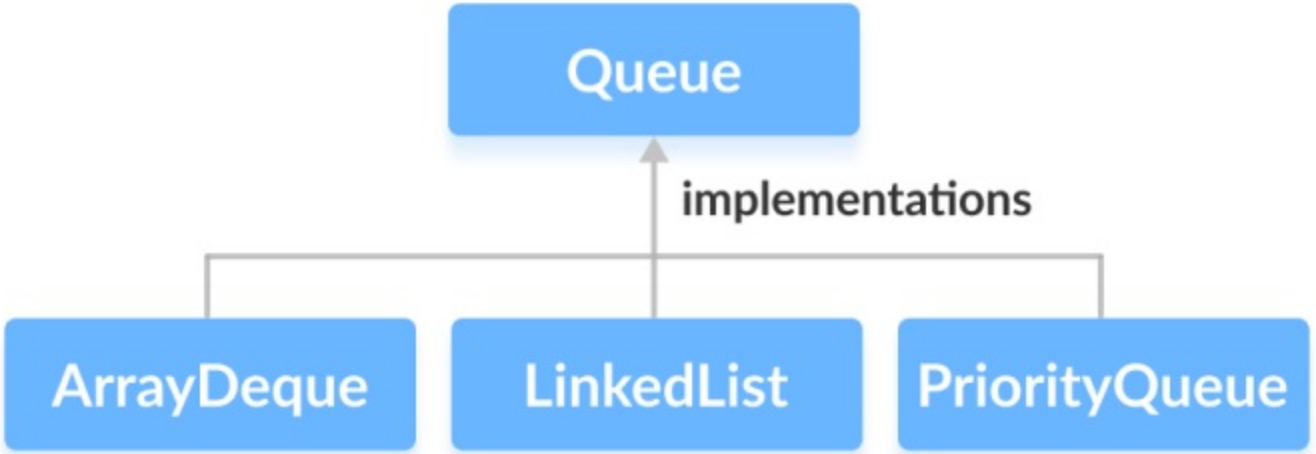
public interface Queue<E> extends Collection<E>

- **Queue** — une collection utilisée pour contenir plusieurs éléments avant le traitement. Outre les opérations de collecte de base, une file d'attente (Queue) fournit des opérations d'insertion, d'extraction et d'inspection supplémentaires.

public interface Queue<E> extends Collection<E>

```
public interface Queue<E> extends Collection<E> {
    E element();     //throws
    E peek();        //null
    boolean offer(E e);     //add - bool
    E remove();      //throws
    E poll();        //null
}
```

L'interface Queue



Fonctionnement Queue

Les méthodes de Queue

- **add()** - Insère l'élément spécifié dans la file d'attente. Si la tâche réussit, add() renvoie true, sinon il lève une exception.
- **offer()** - Insère l'élément spécifié dans la file d'attente. Si la tâche réussit, offer() renvoie true, sinon false.
- **element()** - Renvoie la tête de la file d'attente. Lève une exception si la file d'attente est vide.
- **peek()** - Renvoie la tête de la file d'attente. Renvoie null si la file d'attente est vide.
- **remove()** - Renvoie et supprime la tête de la file d'attente. Lève une exception si la file d'attente est vide.
- **poll()** - Renvoie et supprime la tête de la file d'attente. Renvoie null si la file d'attente est vide.

Les méthodes de Queue

- La file d'attente est utilisée pour insérer des éléments à la fin de la file d'attente et supprime depuis le début de la file d'attente. Il suit le concept FIFO.
- La file d'attente Java prend en charge toutes les méthodes de l'interface Collection, y compris l'insertion, la suppression, etc.
- LinkedList , ArrayBlockingQueue et PriorityQueue sont les implémentations les plus fréquemment utilisées.

Les caractéristiques de Queue

- Si une opération nulle est effectuée sur BlockingQueues, NullPointerException est levée.
- Les files d'attente disponibles dans le package java.util sont des files d'attente illimitées.
- Les files d'attente disponibles dans le package java.util.concurrent sont les files d'attente limitées.
- Toutes les files d'attente, à l'exception des Deques, prennent en charge l'insertion et la suppression respectivement à la fin et à la tête de la file d'attente. Les Deques prennent en charge l'insertion et le retrait d'éléments aux deux extrémités.

UTOPIOS
TRANSFORMATION DIGITALE INTELLIGENTE

Map

public interface Map<K,V>

Map — un objet qui mappe les clés aux valeurs. Une carte ne peut pas contenir de clés en double ; chaque clé peut correspondre à au plus une valeur. Si vous avez utilisé Hashtable, vous connaissez déjà les bases de Map.

public interface Map<K,V>

```
public interface Map<K,V>{
  // Basic operation
  V put(K key,V value);
  V get(Object key);
  V remove(Object key);
  boolean containsKey(Object key);
  boolean containsValue(Object value);
  int size();
  boolean isEmpty();
  // Bulk opeartions
  void putAll(Map<? extends K, ? extends V> m);
  void clear();
  // Collections Views
  public Set<K> keySet();
  public Collection<V> values();
  public Set<Map.Entry<K,V>> entrySet();
  // Interface for entrySet elements
  public interface Entry {
    K getKey();
    V getValue();
    V setValue(V value);
  }
}
```

public interface SortedMap<K,V> extends Map<K,V>

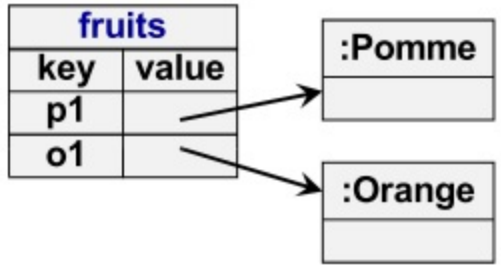
SortedMap — une carte qui maintient ses mappages dans l'ordre croissant des clés. Il s'agit de l'analogue Map de SortedSet. Les cartes triées sont utilisées pour les collections naturellement ordonnées de paires clé/valeur, telles que les dictionnaires et les annuaires téléphoniques.

public interface SortedMap<K,V> extends Map<K,V>

```
public interface SortedMap<K,V> extends Map<K,V> {

  SortedMap<K, V> subMap(K fromKey, K toKey);
  SortedMap<K, V> hedMap(K toKey);
  SortedMap<K, V> tailMap(K fromKey);
  K firstKey();
  K lastKey();

  Comparator<? super K> comparator();

}
```

Collection de type HashMap

- La collection HashMap est une classe qui implémente l'interface Map. Cette collection permet de créer un tableau dynamique d'objet de type Object qui sont identifiés par une clé.
- Déclaration et création d'une collection de type HashMap qui contient des fruits identifiés par une clé de type String :
  - `Map<String, Fruit> fruits=new HashMap<String, Fruit>();`
- Ajouter deux objets de type Fruit à la colelction
  - `fruits.put("p1", new Pomme(40));`
  - `fruits.put("o1", new Orange(60));`
- Récupérer un objet ayant pour clé "p1"
  - `Fruit f=fruits.get("p1");`
  - `f.affiche();`
- Parcourir toute la collection:

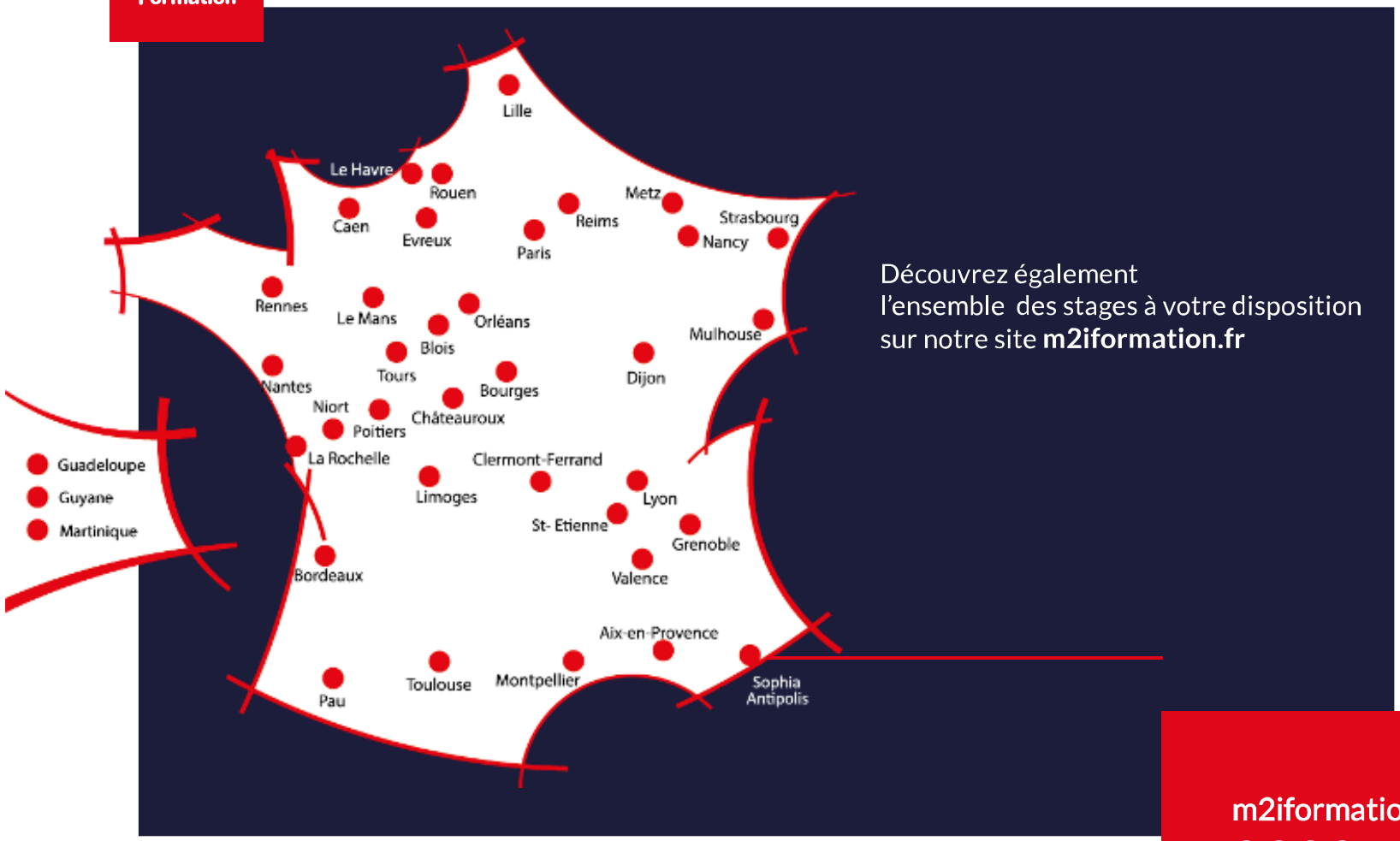



```
Iterator<String> it=fruits.keySet().iterator();
  while(it.hasNext()){
  String key=it.next();
  Fruit ff=fruits.get(key);
  System.out.println(key);
  ff.affiche();
}
```

UTOPIOS
TRANSFORMATION DIGITALE INTELLIGENTE

Merci pour votre attention

Des questions ?



Découvrez également
l'ensemble des stages à votre disposition
sur notre site **m2iformation.fr**

m2iformation.fr