Héritage et Collections d'Objets

Daniel Porumbel
dp.cnam@gmail.com

nombreaux slides dus à
Pierre Cubaud

L'héritage : classe de base

→ Quel est le résultat du code ci-dessous ?

```
class Automobile{
    float vitesseMaxAutoroute(){
        return 130;
    }
    float tempsTrajet (float distance) {
        return distance/vitesseMaxAutoroute
            ();
    }
};
Automobile maVoiture = new Automobile();
float temps = maVoiture.tempsTrajet(130.0);
System.out.println("Temps de trajet pour 130
   km en voiture:="+ temps+ "heures.");
```

L'héritage : classe dérivée

→ Quel est le résultat du code ci-dessous ?

```
class Bus{
    float vitesseMaxAutoroute(){
        return 90;
    }
}
Bus monBus = new Bus();
float temps = monBus.tempsTrajet(130.0);
System.out.println("Temps de trajet pour 130
   km en voiture:="+ temps+ "heures.");
```

- Comment peut-on appeler la méthode `tempsTrajet` ?

L'héritage : classe dérivée

→ Quel est le résultat du code ci-dessous ?

```
class Bus extends Automobile{
    float vitesseMaxAutoroute(){
        return 90;
    }
}
Bus monBus = new Bus();
float temps = monBus.tempsTrajet(130.0);
System.out.println("Temps de trajet pour 130
   km en voiture :="+ temps+ "heures.");
```

- Comment peut-on appeler la méthode `tempsTrajet` ?

  Réponse : Par héritage → la classe `Bus` hérite toutes les méthodes et tous les `attributs` de la classe `Automobile`

La classe Object

En java, toutes les classes héritent de la classe Object, qui n'a pas de super-classe

Ses méthodes sont donc toujours utilisables

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

http://docs.oracle.com/javase/7/docs/api/java/lang/Object.html

Héritage classes Java

- La classe `Object` appartient au paquetage `java.lang` (paquetage par défaut)
- Les classes Swing appartient à des paquetages comme `javax.awt` ou `javax.swing`
  - `JButton` et `jCheckBox` héritent `AbstractButton` qui hérite `java.awt.JComponent`

Héritage classes Java

- La classe `Object` appartient au paquetage `java.lang` (paquetage par défaut)
- Les classes Swing appartient à des paquetages comme `javax.awt` ou `javax.swing`
  - `JButton` et `jCheckBox` héritent `AbstractButton` qui hérite `java.awt.JComponent`

Bonnes Pratiques de Programmation Orienté Objet

- 1ère lettre d'un nom de classe : Majuscule
- 1ère lettre d'un nom de méthode/variable/paquetage : minuscule

Héritage classes Java

- La classe `Object` appartient au paquetage `java.lang` (paquetage par défaut)
- Les classes Swing appartient à des paquetages comme `javax.awt` ou `javax.swing`
  - `JButton` et `jCheckBox` héritent `AbstractButton` qui hérite `java.awt.JComponent`

Bonnes Pratiques de Programmation Orienté Objet

- 1ère lettre d'un nom de classe : Majuscule
- 1ère lettre d'un nom de méthode/variable/paquetage : minuscule

$\Longrightarrow$ Trouver une petite faute sur cette diapo ! 

Héritage classes Java

- La classe `Object` appartient au paquetage `java.lang` (paquetage par défaut)
- Les classes Swing appartient à des paquetages comme `javax.awt` ou `javax.swing`
  - `JButton` et `jCheckBox` héritent `AbstractButton` qui hérite `java.awt.JComponent`

Bonnes Pratiques de Programmation Orienté Objet

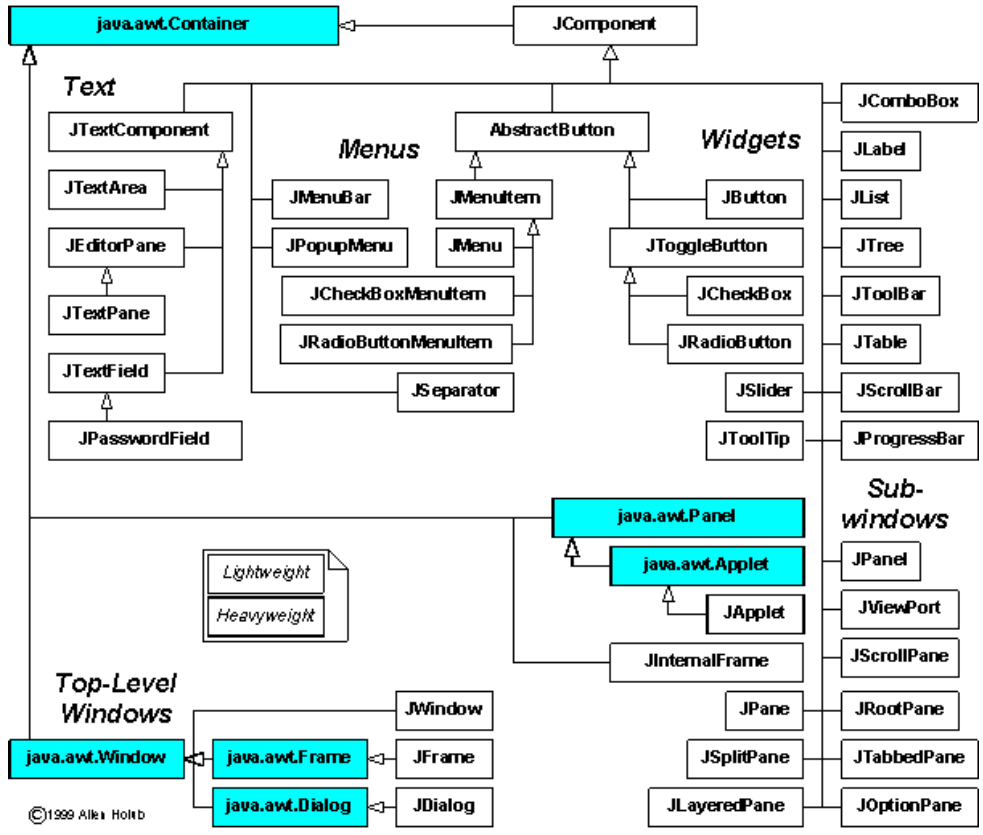
- 1ère lettre d'un nom de classe : Majuscule
- 1ère lettre d'un nom de méthode/variable/paquetage : minuscule

$\Longrightarrow$ Trouver une petite faute sur cette diapo ! 

- La doc d'une classe standard est disponible si on cherche le nom de la classe sur Internet

avec l'héritage, la POO apporte une dimension nouvelle à la programmation

exemple des interfaces graphiques : Swing



Solution : créer une classe **parente** ("super class")

```
class Animal {
 int age;
 Animal(){
  age = 0;
 }
 void manger() {
 // ce que mange l'animal
 }
 void dormir(){
 //
 }
}
```

et ensuite des classes
qui en **héritent** :

```
class Chien extends Animal {
 Chien(){
  super();
 }
 void manger() {
 // ce que mange  le chien
 }
 void aboyer(){
  System.out.println("oua oua");
 }
}
```

fait à la compilation si omis - on peut ajouter d'autres choses

on peut **rédefinir** (override) une méthode parente

(idem pour le chat)

Variante : utiliser aussi la méthode parente

```
class Chien extends Animal {
 Chien(){
  super();
 }
 void manger() {
  // ce que mange tout animal
  super.eat();     <== erreur super.manger()
 // ce que mange specifiquement le chien
 }
 void aboyer(){
  System.out.println("oua oua");
 }
}
```

remarque : on peut empécher la redéfinition de méthode avec le mot-clé **final**

ex dans la classe Animal : final void dormir() { //}

Tableaux d'objets de type différents

```
Bus monBus = new Bus(); OU
Automobile monBus = new Bus(); //polymorphisme
```

Tableaux d'objets de type différents

```
Bus monBus = new Bus(); OU
Automobile monBus = new Bus(); //polymorphisme
```

- On veut simuler une base de données avec des animaux
  - possible d'enregistrer des chiens, chats et poulets
- Comment appliquer une opération sur tous les objets

```
Chien[] leschiens = new Chien[15];
Poulet[] lespoulets = new Poulet[...];
for(int i=0;i<leschiens.length;i++)
                    leschiens[i].dormir();
for(int i=0;i<lespoulets.length;i++)
                    lespoulets[i].dormir();
```

A la place :

```
Animal[] laferme = new Animal[10+5+100];

for (i=0;i<10;i++) laferme[i] = new Chien();
for (i=0;i<5;i++) laferme[i] = new Chat();
for (i=0;i<100;i++) laferme[i] = new Poulet();
//etc

for (i=0;i<laferme.length;i++) laferme[i].manger();
for (i=0;i<laferme.length;i++) laferme[i].dormir();
// et c'est tout :-)
```

Collections Java

Le paquetage `java.util` fournit :

- grandes structures d'organisation de données (collections) : tableau de hachage (`Hashtable`), des tableaux, des listes triés, arbres, queues, piles, etc.

Collections Java

Le paquetage `java.util` fournit :

- grandes structures d'organisation de données (collections) : tableau de hachage (`Hashtable`), des tableaux, des listes triés, arbres, queues, piles, etc.
- des classes utilitaires pour gérer les chaînes (`StringTokenizer`), mais aussi pour gérer les dates et le temps

Documentation → `docs.oracle.com/javase/7/docs/api/java/util/package-summary.html`

Exemple `Hashtable`

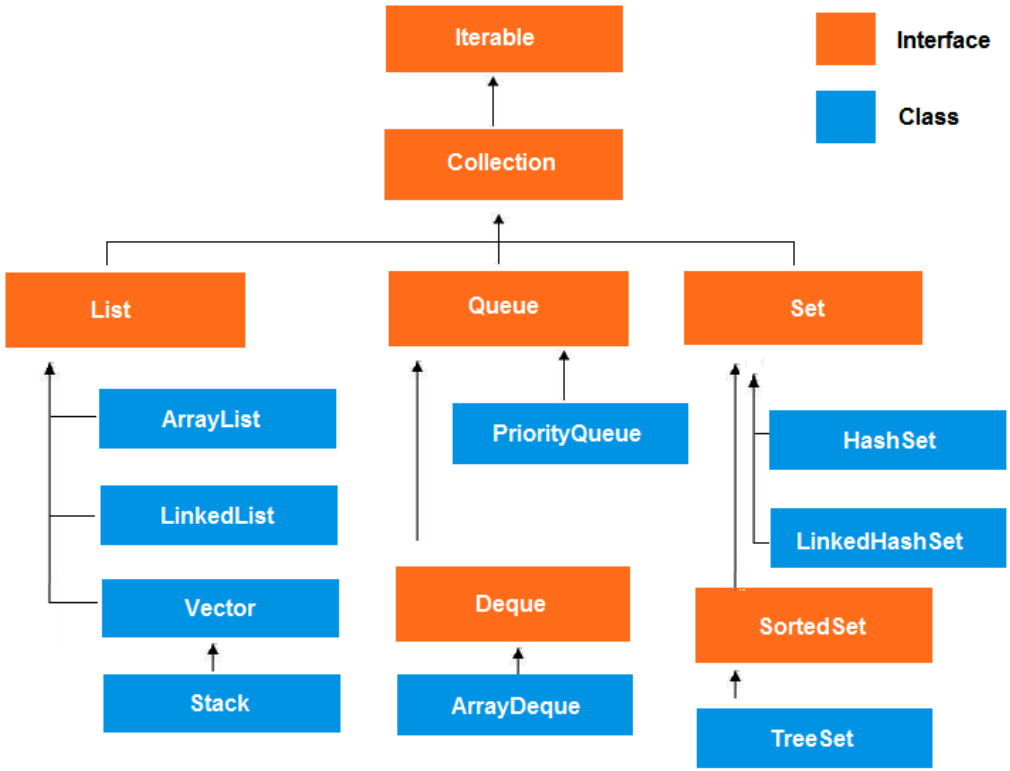
1. Trouver la documentation de la classe : chercher les mots clés “Hashtable javadoc”

Exemple `Hashtable`

1. Trouver la documentation de la classe : chercher les mots clés "Hashtable javadoc"
2. Observer quelques méthodes : `put(clé,valeur)`, `get(clé)`, `containsKey(clé)`
3. Construire un objet `Hashtable` et utiliser ses méthodes

```
import java.util.*;
Hashtable agenda = new Hashtable();
agenda.put("Daniel","1234");
agenda.put("Claire","1234");
if (!agenda.containsKey("Daniel"))
    agenda.put("Daniel","9999");
System.out.println(agenda.get("Daniel"));
```

Classes prédéfinies de Java : les collections



http://www.beingjavaguys.com/2013/03/java-collection-framework.html

Exemple de ArrayList<type>

utilisation de types abstraits (génériques)

principales méthodes :

- par rapport à l'ensemble:

```
int size()          retourne la taille de l'ensemble
boolean isEmpty()   teste si l'ensemble est vide
Object clone()      retourne une copie de l'objet ArrayList<E>
```

- par rapport à la lecture dans l'ensemble:

```
E get(int index)          retourne l'élément d'indice index
boolean contains (E o)    teste si l'ensemble contient o
int indexOf (E o)         retourne l'indice de l'élément o ou -1 s'il n'existe pas
```

- par rapport à l'écriture dans l'ensemble

```
boolean add(E e)          ajouter un élément e à la fin du tableau (retourner true)
void add(int index, E e)  insérer l'élément e à l'indice index
E set(int index, E e)     remplace l'élément d'indice index par e et retourner l'ancien
E remove(int index)       supprimer l'élément d'indice index et le retourne
E remove(E e)             supprimer e et retourn true s'il existait, false sinon
```

Demo ArrayList

```java
import java.util.*;
class ArrayListDemo {
public static void main(String args[]) {
  // creer un arraylist
  ArrayList<String> al = new ArrayList<String>();
  System.out.println("Taille initiale de al: " + al.size());
  //  ajouter des elements
  al.add("C");
  al.add("A");
  al.add("E");
  al.add("B");
  al.add("D");
  al.add("F");
  al.add(1, "A2");
  System.out.println("Taille apres les insertions: " + al.size());
  // afficher le contenu de l'arraylist
  System.out.println("Contenu de al: " + al);
  // enlever des elements
  al.remove("F");
  al.remove(2);
  System.out.println("Taille apres les suppressions: " + al.size());
  System.out.println("Contenu de al: " + al);
 }
}
```

```
fibonacci2:Desktop pcubaud$ javac -Xlint ArrayListDemo.java
fibonacci2:Desktop pcubaud$ java ArrayListDemo
Taille initiale de al: 0
Taille apres les insertions: 7
Contenu de al: [C, A2, A, E, B, D, F]
Taille apres les suppressions: 5
Contenu de al: [C, A2, E, B, D]
fibonacci2:Desktop pcubaud$
```

Exemple programme plus complexe

```
abstract class Nombre{
    public abstract Nombre add(Nombre autreNom);
}
class Frac extends Nombre{
//class Frac implements Comparable<Frac>{
//interface = classe abstraite que avec
//des methods abstraites
    public int a;    // a/b
    public int b;
    //Constructeur
    public Frac (){
        a = 1;
        b = 1;
    }
    public Frac (int valeurFrac){
        a = valeurFrac;
        b = 1;
    }
    public Frac (int m, int n){
        a = m;
        b = n;
    };
...
...
```

Exemple programme plus complexe

```
...
    //Methodes
    public Nombre add(Nombre cd){
        int c = ((Frac)cd).a;
        int d = ((Frac)cd).b;
        int e = a*d+b*c;
        int f = b*d;
        Frac acbd = new Frac(e,f);
        return acbd;
    }
    public float  toFloat(){
        return ((float)a)/b;
    }
    public String toString(){
        String resultat = a + "/" + b;
        return resultat;
    }
    public int compareTo(Frac autre){
        int a2 = autre.a;
        int b2 = autre.b;
        if(a/b<=a2/b2)
            return -1;
        return 1;
    }
...
```

Exemple programme plus complexe

```
....
    boolean equals(int x){
        if(a==b*x)
            return true;
        else
            return false;
    }
}

Nombre [] tab = new Frac[7];
//initialisation
for(int i=1;i<=tab.length;i++){
    tab[i-1] = new Frac(1,i);
}
Frac somme = new Frac(0);
for(int i=0;i<tab.length;i++){
    somme = (Frac)somme.add(tab[i]);
}
System.out.println(somme);
System.out.println(somme.toFloat());
//java.util.Arrays.sort(tab); pour trier si l'interface est active (
    Ligne 6)
```