

CLASSES ABSTRAITES

Classes abstraites

Méthodes abstraites

- Une méthode est abstraite (modificateur **abstract**) lorsqu'on la déclare, sans donner son implémentation
- La méthode sera implémentée par les classes filles

Classes abstraites

- Une classe doit être déclarée abstraite (**abstract class**) si elle contient une méthode abstraite
- Il est interdit de créer une instance d'une classe abstraite

Compléments

- Si on veut empêcher la création d'instances d'une classe on peut la déclarer abstraite même si aucune de ses méthodes n'est abstraite
- Une méthode **static** ne peut être abstraite (car on ne peut redéfinir une méthode **static**)

- Certaines classes ne doivent tout simplement pas être instanciées

- Exemple Animal

- Que signifie un objet de type animal?

- `Animal anim = new Animal();`

- Quelle est sa forme? Sa taille? Sa couleur?...

Exemple

```
public abstract class AnimalCompagnie{
    private String nom;
    public AnimalCompagnie(String n){
        nom = n;
    }
    public abstract void parler();
}

public class Chien extends AnimalCompagnie{
    public Chien(String s) {
        super(s);
    }
    public void parler() {
        System.out.println("ouah ouah");
    }
}
```

```
public class Chat extends
    AnimalCompagnie{
    public Chat(String s) {
        super(s);
    }
    public void parler() {
        System.out.println("miaou");
    }
}
```

Exemple (suite)

```
public class TestAnimal{
    public static void main (String args[]){
        Chien f = new Chien("Fifi");
        Chat g = new Chat("Chloe");
        //AnimalCompagnie a = new AnimalCompagnie("bob");
        f.parler();
        g.parler();
        AnimalCompagnie [] a = new AnimalCompagnie [2];
        a[0] = f;
        a[1] = g;
        for(int i=0; i<a.length; i++) {
            a[i].parler();
        }
    }
}
```

Exercice

Le but de l'exercice est de créer une hiérarchie de classes pour représenter les étudiants d'une université. Il y a 3 types d'étudiants : ceux en Licence, ceux en Master, et ceux en Doctorat. Chaque étudiant a un nom, une adresse et un numero. Les étudiants ont un profil. Pour les étudiants en Licence on parle de parcours. Les étudiants en Master une spécialité et les étudiants en Doctorat un directeur de recherche.

1. Vous définirez les classes nécessaires à cette hiérarchie de classes, en leurs ajoutant les membres nécessaires et les méthodes usuelles (constructeurs, toString(), get()et set() etc...).
2. Écrire une application qui construit un ensemble d'étudiants (utiliser la classe Scanner pour que l'utilisateur puisse saisir les données) affiche une liste d'étudiants dans l'ordre selon le numero. Pour chaque étudiant, il faut afficher toutes les informations le concernant.

Exo: classes abstraites

- L'objectif est de créer le concept forme géométrique et une forme doit retourner sa surface, son périmètre et sa couleur. On doit pouvoir aussi modifier sa couleur. Définir des implémentations pour des objets rectangle, carre et cercle.

Classe **Object**

- En Java, la racine de l'arbre d'héritage des classes est la classe **java.lang.Object**
- La classe **Object** n'a pas de variable d'instance ni de variable de classe
- La classe **Object** fournit plusieurs méthodes qui sont héritées par toutes les classes sans Exception:
 - ▣ les plus couramment utilisées sont les méthodes **toString()** et **equals()**

Classe **Object** - méthode **toString()**

- **public String toString()**

renvoie une description de l'objet sous la forme d'une chaîne de caractères

- Elle est utile pendant la mise au point des programmes pour faire afficher l'état d'un objet ; la description doit donc être concise, mais précise

Méthode **toString()** de la classe **Object**

- Elle renvoie le nom de la classe, suivie de « @ » et de la valeur de la méthode **hashCode**
- La plupart du temps (à la convenance de l'implémentation) **hashCode** renvoie la valeur hexadécimale de l'adresse mémoire de l'objet
- Pour être utile dans les nouvelles classes, la méthode **toString()** de la classe **Object** doit donc être redéfinie

- Si **p1** est un objet, **System.out.println(p1)** (ou **System.out.print(p1)**) affiche la chaîne de caractères **p1.toString()** où **toString()** est la méthode de la classe de **p1**
- Il est ainsi facile d'afficher une description des objets d'un programme pendant la mise au point du programme
- Voir exemple **Vehicule.java**

Classe String

- Un type dont des objets sont figés; il n'est pas possible de modifier le contenu de tels objets.

Exemple: Classe String

```
public class StringTest {
    public static void main (String args [] ) {
        String prenom;
        prenom = "Ahmed"; //creation d'un objet
        prenom = "Jamal"; // nouveau objet
        prenom = new String("Ahmed"); // nouveau objet
        String prenom2 = prenom;
        System.out.println(prenom2); // Ahmed
        System.out.println(prenom.length()); // 5
        System.out.println(prenom.charAt(1)); // h
        String nom = "Amrani";
        System.out.println(nom.compareTo(prenom)); // >0
        System.out.println(nom.substring(2,4)); // ran
        System.out.println(nom.indexOf("mra",0)); // 1
        String nom2 = nom;
        System.out.println(nom.replace('a','i')); // Amrini
        System.out.println(nom2); // Amrani
        System.out.println(nom); // Amrani
        nom = " " + nom + " ";
    } }
```

La classe StringBuffer

- Une chaîne de caractères qu'on peut modifier.
- Automatiquement redimensionné en fonction des besoins.


```
public class StringBufferTest {
    public static void main (String args [] ) {
        StringBuffer prenom;
        prenom = new StringBuffer("kamel");
        StringBuffer prenom2 = prenom;
        System.out.println(prenom2); // kamel
        System.out.println(prenom.length()); // 5
        System.out.println(prenom.charAt(1)); // a
        System.out.println(prenom.append(" eddine"));
        System.out.println(prenom); // kamel eddine
        System.out.println(prenom.insert(5," ***")); // kamel***eddine
        System.out.println(prenom2); // kamel***eddine
    }
}
```

La classe StringTokenizer

- Pour décomposer une chaîne de caractère en tokens.
- Par défaut les tokens sont délimités par :
 - « \t \n \r \f » sauf si les délimiteurs sont explicitement spécifiés.

Exemple :La classe StringTokenizer

```
import java.util.*;
public class StringTokenTest {
    public static void main(String args[]) {
        String s = "ceci est un test";
        StringTokenizer st = new StringTokenizer(s);
        System.out.println(st.countTokens());
        while (st.hasMoreTokens()) {
            System.out.println(st.nextToken());
        }
        st = new StringTokenizer(s,"t");
        while (st.hasMoreTokens()) {
            System.out.println(st.nextToken());
        }
    }
}
```

☐ 4

☐ ceci

☐ est

☐ un

☐ test

☐ ceci es

☐ un

☐ es

☐ Résultat?

Définition des interfaces

- Une interface est une « classe » purement abstraite dont toutes les méthodes sont abstraites et publiques
- C'est une liste de noms de méthodes publiques

Exemples

```
public interface Figure {  
    public abstract void dessineToi();  
    public abstract void deplaceToi(int x, int y);  
    public abstract Position getPosition();  
}
```

```
public interface Comparable {  
    /** renvoie vrai si this est plus grand que o */  
    boolean plusGrand(Object o);  
}
```

public abstract
peut être implicite

Les interfaces sont implémentées par des classes

- Une classe implémente une interface **I** si elle déclare « **implements I** » dans son en-tête


Exemple d'implémentation

```
public class Ville implements Comparable {  
    private String nom;  
    private int nbHabitants;  
    ...  
    public boolean plusGrand(Object objet) {  
        if (objet instanceof Ville) {  
            return nbHabitants > ((Ville)objet).nbHabitants;  
        }  
        else {  
            throw new IllegalArgumentException();  
        }  
    }  
}
```

Exactement la même signature
que dans l'interface **Comparable**



Les exceptions sont étudiées
dans la prochaine partie du cours



Classe qui implémente partiellement une interface

- Soit une interface **I1** et une classe **C** qui l'implémente :
public class C implements I1 { ... }
C peut ne pas implémenter toutes les méthodes de **I1**
- Mais dans ce cas **C** doit être déclarée **abstract** (il lui manque des implémentations)
- Les méthodes manquantes seront implémentées par les classes filles de **C**

- Une classe peut implémenter une ou plusieurs interfaces (et hériter d'une classe...) :
- **public class CercleColore extends Cercle implements Figure, Coloriable {**

Contenu des interfaces

- Une interface ne peut contenir que
 - des méthodes **abstract** et **public**
 - des définitions de constantes publiques
(«**public static final** »)
- Les modificateurs **public**, **abstract** et **final** sont optionnels (en ce cas, ils sont implicites)
- Une interface ne peut contenir de méthodes **static**, **final**

Accessibilité des interfaces

- Une interface peut avoir la même accessibilité que les classes :
- **–public** : utilisable de partout
- **–** sinon : utilisable seulement dans le même paquetage

Les interfaces comme types de données

- Une interface peut servir à déclarer une variable, un paramètre, une valeur retour, un type de base de tableau, un *cast*,...
- Par exemple, **Comparable v1**; indique que la variable **v1** référencera des objets dont la classe implémentera l'interface **Comparable**

Interfaces et typage

- Si une classe **C** implémente une interface **I**,
 - ▣ le type **C** est un sous-type du type **I** :
 - ▣ tout **C** peut être considéré comme un **I**
- On peut ainsi affecter une expression de type **C** à une variable de type **I**
- Les interfaces « s'héritent » : si une classe **C** implémente une interface **I**, toutes les sousclasses de **C** l'implémentent aussi (elles sont des sous-types de **I**)

Exemple d'interface comme type de données

```
public static boolean croissant(Comparable[] t) {  
    for (int i = 0; i < t.length - 1; i++) {  
        if (t[i].plusGrand(t[i + 1]))  
            return false;  
    }  
    return true;  
}
```

instanceof

- Si un objet **o** est une instance d'une classe qui implémente une interface **Interface**,
 - ▣ **o instanceof Interface** est vrai

Polymorphisme et interfaces

```
public interface Figure {  
    void dessineToi();  
}  
  
public class Rectangle implements Figure {  
    public void dessineToi() {  
        ...  
    }  
  
public class Cercle implements Figure {  
    public void dessineToi() {  
        ...  
    }  
}
```


Polymorphisme et interfaces (suite)

```
public class Dessin {  
    private Figure[] figures;  
    ...  
    public void afficheToi() {  
        for (int i=0; i < nbFigures; i++)  
            figures[i].dessineToi();  
        }  
    ...  
}
```

A quoi servent les interfaces ?

- Garantir aux « clients » d'une classe que ses instances peuvent assurer certains services, ou qu'elles possèdent certaines propriétés (par exemple, être comparables à d'autres instances)
- Faire du polymorphisme avec des objets dont les classes n'appartiennent pas à la même hiérarchie d'héritage (l'interface joue le rôle de la classe mère, avec *upcast* et *downcast*)

Exemple Interface Comparable

- Contient la méthode nécessaire pour comparer deux objets:
 - Objectif: Définir un ordre naturel sur des objets
 -
- Public interface Comparable {
 Public int compareTo (Object o);
}
- La méthode retourne un integer négatif si
 - this est plus petit que o, 0 si les deux objets sont égaux ou positif sinon.
- Une classe qui implémente l'interface Comparable peut utiliser la méthode: exple la classe Arrays

Exemple Interface Comparable

- On peut toujours faire des *casts* (*upcast* et *downcast*) entre une classe et une interface qu'elle implémente (et un *upcast* entre une interface et la classe **Object**) :

// upcast Personne → Comparable

Comparable c1 = new Personne("Amine", 20);

Comparable c2 = new Personne("Karima", 25);

...

if (c1.plusGrand(c2)) // upcast Comparable → Object

// downcast Comparable → Personne

System.out.println(((Personne)c2).GetAge());

Exemple méthode Arrays.sort(Object [])

▣ Méthodes de tris Arrays.sort

```
Random r = new Random();  
int []t = new int [10];  
for (int i=0; i<10; i++)  
    t[i] = r.nextInt(100);  
for (int i=0; i<10; i++)  
    System.out.print(t[i] + " ");  
Arrays.sort(t);  
System.out.println();  
for (int i=0; i<10; i++)  
    System.out.print(t[i] + " ");  
System.out.println();
```

(exemple Aleatoire.java)

Exemple Interface Comparable

```
public class Nombre implements Comparable
{
    private int i;
    public Nombre(int i){
        this.i = i;
    }
    public int compareTo(Object n){
        if(i<((Nombre)n).i) return -1;
        if (i == ((Nombre)n).i) return 0;
        return 1;
    }
    public String toString(){
        return Integer.toString(i);
    }
}
```

- La méthode Arrays.sort sait trier tout objet implémentant l'interface Comparable

```
Random r = new Random();
Nombre[] n = new Nombre[10];
for (int i=0; i<10; i++)
    n[i] = new Nombre(r.nextInt(100));
for (int i=0; i<10; i++)
    System.out.print(n[i] + " ");
Arrays.sort(n);
System.out.println();
for (int i=0; i<10; i++)
    System.out.print(n[i] + " ");
System.out.println();
```

Exemple: Personne

```
import java.lang.reflect.Array;
import java.util.* ;
```

```
public class Personne implements
    Comparable {
    String nom ;
    int age ;
    Personne(String nom, int age) {
    this.nom = nom ;
    this.age = age ;
    }
```

```
    public String toString() {
    return nom + "(" + age + ")" ;
    }
    public int compareTo(Object o)
    {
    //if (o instanceof Personne)
    //return (this.age - ((Personne)o).age) ;
    //return Integer.MAX_VALUE ;
    if (this.age ==((Personne)o).age) return 0 ;
    if (this.age > ((Personne)o).age) return 1 ;
    return -1;    }
```

Exemple: Personne

```
public static void main(String[] args) {  
    Personne [] personnes = new Personne[4];  
    personnes[0]= new Personne("mohamed",24);  
    personnes[1]= new Personne("jamal",26);  
    personnes[2]=new Personne("hakim",22);  
    personnes[3]=new Personne("hayat",28);  
    Arrays.sort(personnes);  
    for(int i=0; i<personnes.length;i++)  
        System.out.println(" "+personnes[i]);  
}
```


Exemple: Personne

```
List<Personne> personnes = new ArrayList();
```

```
personnes.add(new Personne("Toto", 26)) ;
```

```
personnes.add(new Personne("Jamal", 22)) ;
```

```
personnes.add(new Personne("Salim",60)) ;
```

```
personnes.add(new Personne("Taoufik", 34)) ;
```

```
personnes.add(new Personne("Hayat", 22)) ;
```

```
if (personnes.contains(new Personne("hayat",69)))
```

```
    System.out.println("OK");
```

```
else    System.out.println("Not OK ");
```

Exemple: Personne

```
System.out.println("Liste de depart = " + personnes) ;
Personne pers = (Personne) Collections.max(personnes) ;
System.out.println("Pers la plus agee = " + pers) ;
pers = (Personne) Collections.min(personnes) ;
System.out.println("Pers la plus jeune = " + pers) ;
Collections.reverse(personnes) ;
System.out.println("Liste inversee = " + personnes) ;
Collections.sort(personnes) ;
System.out.println("Liste triee = " + personnes) ;
Collections.shuffle(personnes) ;
System.out.println("Liste melangee = " + personnes) ;
}
```

Interface et classe dérivée

- Une classe dérivée peut implémenter une ou plusieurs interfaces
 - Les fonctionnalités proposés par l'interface sont indépendante de l'héritage
 - Interface I1 {...}
 - Interface I2 {...}
 - Class A implements I1 {...}
 - Class B extends A implements I2 {...}

Dérivation d'interface

- On peut définir une interface comme une généralisation d'une autre

```
interface X{
```

```
...}
```

```
interface Y{
```

```
...}
```

```
interface Z{
```

```
...}
```

```
interface A extends X {
```

```
...}
```

```
interface B extends X, Y, Z {
```

```
...}
```

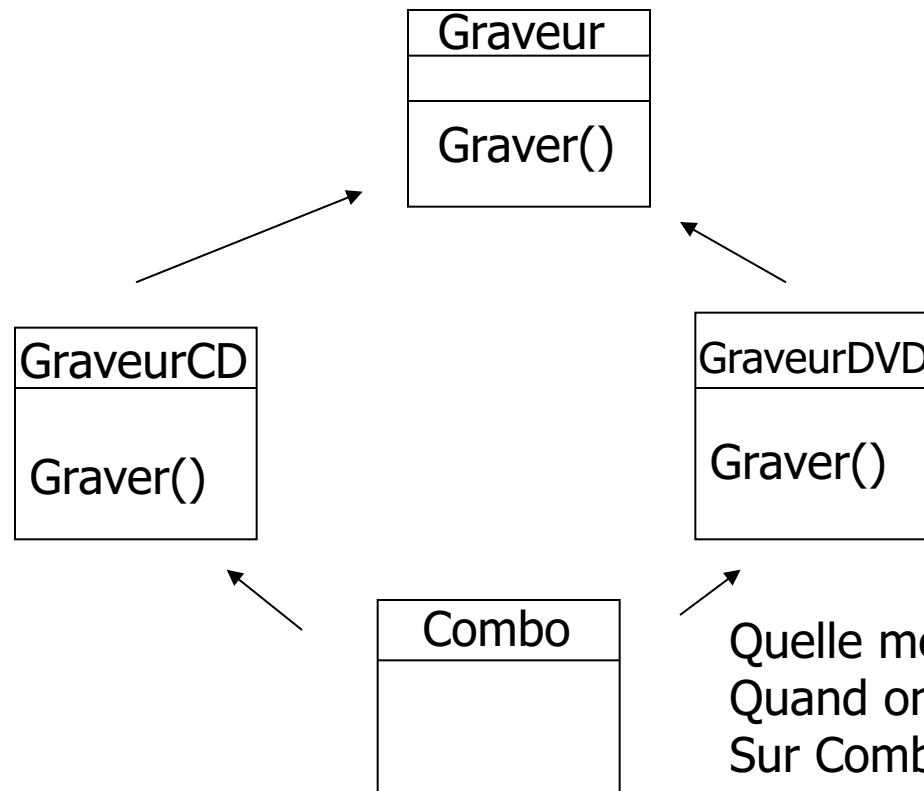
- La dérivation revient simplement à concaténer des déclarations

```
interface A {  
    int info = 1;  
}  
interface B extends A{  
}  
public class C implements B{  
    public void f() {  
        System.out.println(info);  
        System.out.println(A.info);  
    }  
  
    public static void main (String args[])  
    {  
        C c = new C();  
        c.f();  
    }  
}
```

Qu'affiche ce programme?

Le losange de la mort

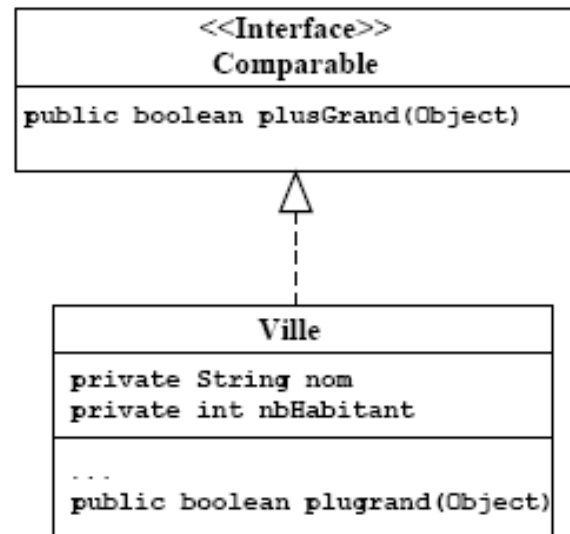
- Héritage multiple interdit en Java
- Exemple



Quelle méthode s'exécute
Quand on appelle `graver()`
Sur `Combo`?

- La notion d'interface esquisse ce problème tout en permettant d'exploiter la notion de polymorphisme
 - Les méthodes sont abstraites et la sous-classe est obligé d'implémenter les méthodes abstraites.
 - Au moment de l'exécution la JVM ne se demandera pas laquelle des deux versions héritées elle est censée appeler

Interfaces en notation UML



- Créez une interface `Rangement` ayant les déclarations de fonctions suivantes:

```
void ajouter(Object o),  
Object retirer(int i),  
Object tete(),  
int getTaille(),  
void setTaille(int i),  
boolean estVide().
```

- Créer une classe abstraite `StructureDeDonnees` qui implante l'interface `Rangement`. Ajoutez à cette classe abstraite l'attribut `nbObjets` et implanter les fonctions `getTaille`, `setTaille`, et `estVide` dans le corps de la classe abstraite.
- Créer ensuite deux classes dérivées de cette classe, l'une, `Tableau`, utilisant un stockage des éléments dans un tableau, l'autre, `Liste` utilisant un stockage des éléments dans une liste. Tester votre implantation.