Les interfaces

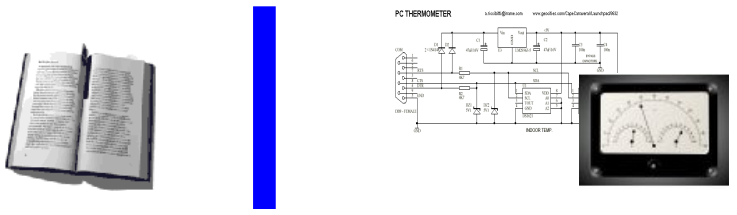
Contenu de la section

LES INTERFACES....1

Contenu de la section....2
Les interfaces et les modèles à Objets (1)....3
les interfaces par opposition aux classes....3
Les interfaces et les modèles à Objets (2)....4
un exemple....4
Une introduction sur le typage....5
la classification "hiérarchique" des informations....5
les relations sur les types....5
Les interfaces et le typage....6
la définition d'un sur-type pour un ensemble de classes....6
Les interfaces et le polymorphisme (1)....7
un rappel sur les références, les classes et les objets....7
Les interfaces et le polymorphisme (2)....8
L'exécution des méthodes....8
La définition des interfaces....9
la syntaxe....9
L'extension des interfaces....10
la syntaxe....10
L'implémentation des interfaces....11
la syntaxe....11

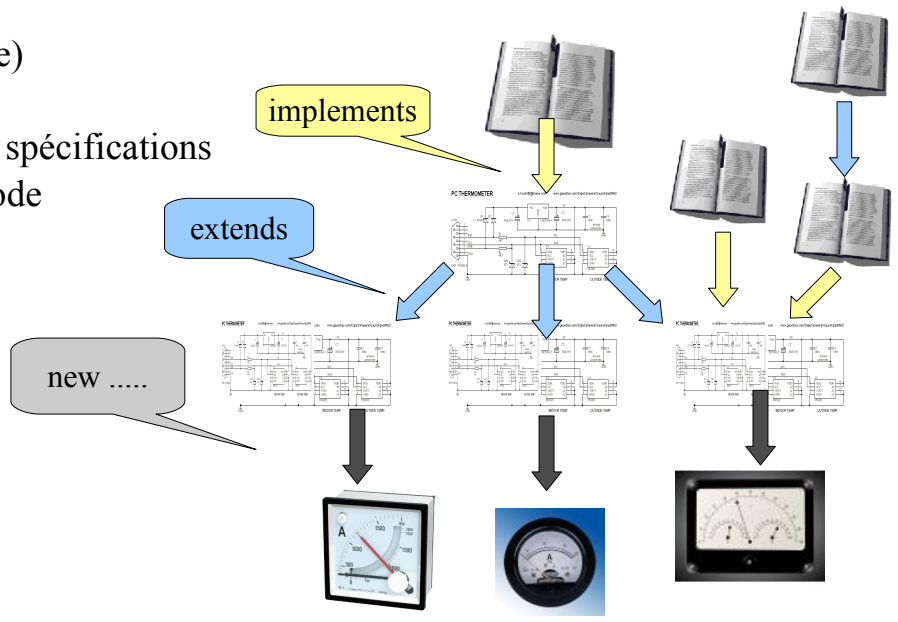
Les interfaces et les modèles à Objets (1)

les interfaces par opposition aux classes



- les interfaces : le mode d'emploi (aucune indication de mise en oeuvre)
- les classes : contiennent des indications de mise en oeuvre
- les classes abstraites correspondent à des mises en oeuvre intermédiaires (souvent incomplètes)

- la relation d'implémentation (mise en oeuvre d'une interface)
- la relation d'extension (factorisation/réutilisation)
  - l'extension d'interfaces : la factorisation/réutilisation des spécifications
  - l'extension de classes : la factorisation/réutilisation du code



Les interfaces et les modèles à Objets (2)

un exemple

- une référence d'interface peut désigner tout objet instancié à partir d'une classe qui implémente directement ou indirectement cette interface
- une interface déclare l'ensemble des méthodes qu'elle permet d'invoquer

```
interface IPoint2D {
    public int getX();
    public int getY();
    public void  deplacer(int dx,int dy);
}
```

un type définit les méthodes qui lui sont associées

```
class Point2DPolaire implements IPoint2D  {
    pr
    pr

    pu
    pu

    pu
    pu
}
```

```
class Point2DCartesien implements IPoint2D {
  private int x;
  private int y;

  public Point2DCartesien(int _x, int _y)  { x = _x; y = _y;  }
  public Point2DCartesien() { x = 0 ; y = 0; }

  public int getX() { return x; }
  public int getY() { return y; }
  public void  deplacer(int dx,int dy)  { x += dx; y += dy;  }
}
```

```
class Segment  {
    private IPoint2D origine;
    private  IPoint2D extremite;

    public Segment(IPoint2D orig, IPoint2D ext) { origine = orig; extremite = ext; }
    public IPoint2D getOrigine() { return origine; }
    public IPoint2D getExtremite() { return extremite; }
    public double longueur() {
        int dx = (origine.getX() -extremite.getX());
        int dy = (origine.getY() -extremite.getY());
        return Math.sqrt(dx*dx + dy * dy);
    }
}
```

Une introduction sur le typage

la classification "hiérarchique" des informations

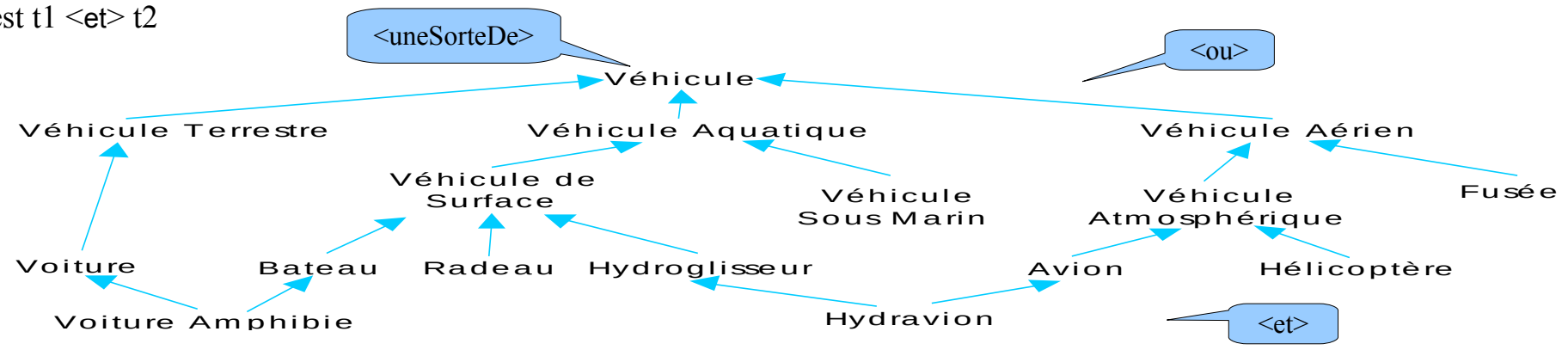
- elle enrichit l'expressivité du langage
  - elle améliore la "compréhensibilité"
  - elle permet la vérification statique et dynamique de correction

```
Un langage élémentaire :
    Types : Truc
    relation : appartientA(Truc, Truc) -> bool
Un programme :
    v1, v2 : Truc
    p1, p2 : truc
    ..............
    appartientA(v1,p1)
    appartientA(v1,v2)
```

```
Un langage élémentaire :
    Types : Voiture, Personne
    relation : appartientA(Voiture, Personne) -> bool
Un programme :
    v1, v2 : Voiture
    p1, p2 : Personne
    ..............
    appartientA(v1,p1)          //OK
    appartientA(v1,v2)          //erreur !!!!
```

les relations sur les types

- les opérateurs usuels
  - t est <uneSorteDe> t1
  - t est t1 <ou> t2
  - t est t1 <et> t2



- Il existe une interprétation ensembliste des relations sur les types
- conversion implicite : Type → SurType
- conversion explicite : Type → SousType

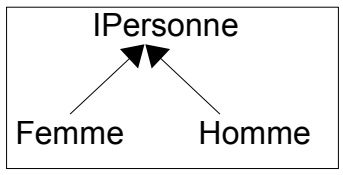
Les interfaces et le typage

la définition d'un sur-type pour un ensemble de classes

- la relation implements est une relation de sous-typage
- la définition d'un type <ou> via une interface

un type définit les méthodes qui lui sont associées

```
interface IPersonne {
   public void quiEsTu() ;
   public IPersonne complement() ;
}
```




```
public class Femme implements IPersonne {

   public void quiEstu() { …… }
   public IPersonne complement() { ..... }
}

public class Homme implements IPersonne {

   public void quiEstu() { …… }
   public IPersonne complement() { ..... }
}
```

```
public class Couple {
   private IPersonne membre1;
   private IPersonne membre2;

   public Couple(IPersonne m1, IPersonne m2) {
      membre1 = m1;
      membre2 = m2;
   }
   public void quiEtesVous() {
      membre1.quiEstu();
      membre2.quiEstu();
   }
}
```

Les interfaces et le polymorphisme (1)

un rappel sur les références, les classes et les objets ....

- une référence ⇔ ensemble des champs et des méthodes accessibles
- accès via une référence ⇔ accès aux champs et méthodes de **l'objet** référencé

- une interface java est une **déclaration** de méthodes (et de constantes)
- une classe est une **définition** (elle définit le code des méthodes déclarées dans l'interface)

```
interface IPersonne {
   public void quiEsTu() ;
   public IPersonne complement() ;
}

class Homme implements IPersonne {
   public void quiEsTu() {  System.out.println("je suis un Ohm."); }
   public IPersonne complement() {  return new Homme() ; }
}

class Test0 {
   public static void main(String[ ] args) {
      IPersonne ptP= new Homme();          //OK
      ptP.quiEsTu() ;                      //OK
   }
}
```

une référence IPersonne peut référencer tout objet d'une classe sous-type de IPersonne. Un tel objet contient donc quiEsTu() et complement()

```
class Homme1 implements IPersonne {
   public void quiEsTu() {  System.out.println("je suis un OhmUn."); }
   public IPersonne complement() {  return new Homme1() ; }
   public void affiche( ) { quiEsTu(); }
}

class Test1 {
   public static void main(String[ ] args) {
      IPersonne ptP= new Homme1();          //OK
      ptP.affiche() ;    //erreur : affiche inconnu dans IPersonne !
   }
}
```

L'interface IPersonne décrit la vision "utilisateur" commune offerte par tout objet d'une classe sous-type de IPersonne.

Les interfaces et le polymorphisme (2)

L'exécution des méthodes

- le code exécuté est celui de l'OBJET

```
interface IPersonne {
    public void quiEsTu();
    public IPersonne complement();
}

class Homme implements IPersonne {
    public void quiEsTu() { System.out.println("je suis un Ohm..."); }
    public IPersonne complement() { return new Femme(); }
    public void specificH() { ..... }
}

class Femme implements IPersonne {
    public void quiEsTu() { System.out.println("je suis une Fam..."); }
    public IPersonne complement() { return new Homme(); }
    public void specificF() { ..... }
}
```

```
class TestPersonne {
    public static void main(String[ ] args) {
        IPersonne refA = new Femme( );
        refA.quiEsTu() ;
        for (int i=0; i<3; i++)   {
            refA= refA.complement() ;
            refA.quiEsTu() ;
        }
    }
}
```

résultat de l'exécution

```
je suis une Fam
je suis un Ohm
je suis une Fam
je suis un Ohm
```

La définition des interfaces

la syntaxe

- définition via le mot clé : **interface**
- contient uniquement des déclarations de méthodes abstraites et de **constantes**

- déclaration de méthodes → méthodes IMPLICITEMENT public abstract
- AUCUNE méthode static

- déclaration de constantes (voir suite) → champs IMPLICITEMENT public static final

```
interface IRadio{
   public static final boolean ON = true;        //voir suite

   public abstract void allumer();
   public abstract  void eteindre();
   public void reglerCanal(double canal);
   void reglerSon(double volume);
}
```

L'extension des interfaces

la syntaxe

- définition via le mot clé : extends
- **extension multiple autorisée pour les interfaces**
- attention aux ambiguités concernant les déclarations de constantes

```
interface IRadio{
    //public static final boolean OFF = false;
    //public static final boolean ON = true;

    public abstract void allumer();
    public abstract  void eteindre();
    public void reglerCanal(double cn);
}

interface IReveil {
    void reglerHeure(int hr, int mn, int sc);
    public abstract void activerAlarme(boolean b);
    void reglerHeureAlarme(int hr, int mn,int sc);
}

interface IRadioReveil extends IRadio, IReveil {
    public abstract void commuterAlarmeSurRadio(boolean b);
}
```

L'implémentation des interfaces

la syntaxe

- la classe comporte la déclaration "implements"
- la classe définit les méthodes déclarée dans l'interface
- l'implémentation de plusieurs interfaces est possible
- l'implémentation partielle est possible → classe abstraite (voir suite)

```
class RadioReveil0 implements IRadioReveil {
    public void allumer() { ..... }
    public void eteindre() { ..... }
    public void reglerCanal(double cn) { ..... }
    void reglerHeure(int hr, int mn, int sc) { ..... }
    public void activerAlarme(boolean b) { ..... }
    ...................
}

class RadioReveil1 implements IRadio, IReveil {
    public void allumer() { ..... }
    public void eteindre() { ..... }
    public void reglerCanal(double cn) { ..... }
    void reglerHeure(int hr, int mn, int sc) { ..... }
    public void activerAlarme(boolean b) { ..... }
    ...................
}
```