Héritage et langage JAVA

"Nous autres, mordus d'informatique, préférons par-dessus tout passer notre temps à bidouiller nos ordinateurs, plutôt que les utiliser pour faire quelque chose de productif."

Dave Barry

Rappels semaine dernière

Soient les classes ci-dessous

```
public class Personne {
   private String prenom, nom;

   public Personne(String prenom, String nom) {
      this.prenom = prenom;
      this.nom = nom;
   }

   public String getPrenom() {…}
   public String getNom() {…}
   public void setPrenom(String prenom) {…}
   public void setNom(String nom) {…}
}
```

```
public class Voiture {
   private String marque="", modele="";

   public String getMarque() {…}
   public String getModele() {…}
   public void setMarque(String marque) {…}
   public void setModele(String modele) {…}
}
```

Quelles instructions sont correctes ?

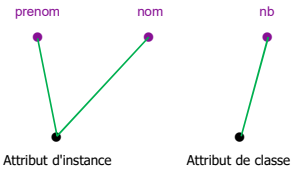
- ☐ `Personne x=new Personne();` **non**
- ☑ `Personne y=new Personne("Pierre","Kiroule");`
- ☑ `Voiture v=new Voiture();`
- ☐ `Voiture w=new Voiture("Renault","Espace");` **non**

Rappels semaine dernière

Soit la classe ci-dessous

```
public class Personne {
   private String prenom, nom;
   private static int nb =0;

   public Personne(String prenom, String nom) {
      this.prenom = prenom;
      this.nom = nom;
      Personne.nb++;
   }

   public _____ getNb(){
      return nb;
   }
}
```

Quels sont les différents types d'attributs ?



Rappels semaine dernière

Soit la classe ci-dessous

```
public class Personne {
   private String prenom, nom;
   private static int nb =0;

   public Personne(String prenom, String nom) {
      this.prenom = prenom;
      this.nom = nom;
      Personne.nb++;
   }

   public __static int___ getNb(){
      return nb;
   }
}
```

D'après le code, à quoi sert l'attribut *nb* ?

Il contient le nombre d'objets de type Personne créés

Dans le programme principal, on veut pouvoir afficher **à tout moment**, le nombre d'objets Personne. Complétez le prototype de la méthode getNb() et les appels dans le main.

```
public static void main(String[] args) {

   System.out.println(_Personne_.getNb());

   Personne x=new Personne("Ella","Paletan");
   Personne y=new Personne("Pierre","Kiroule");

   System.out.println(_Personne_.getNb());
}
```

ou x ou y

Paradigme Objet

- La vision Objet de la programmation possède 3 grands principes:
  - **L'encapsulation**: principe d'abstraction de données et abstraction procédurale
  - **L'héritage**: principe de Généralisation/Spécification
  - **Le polymorphisme**: permet à une méthode de s'adapter à plusieurs classes

Classes (exemples)

Personne
• prénom, nom
• seDeplacer()

Enseignant
• prénom, nom • section
• seDeplacer() • preparerCours()

Etudiant
• prénom, nom • numéroEtu
• seDeplacer() • étudier()

Véhicule
• marque
• seDeplacer()

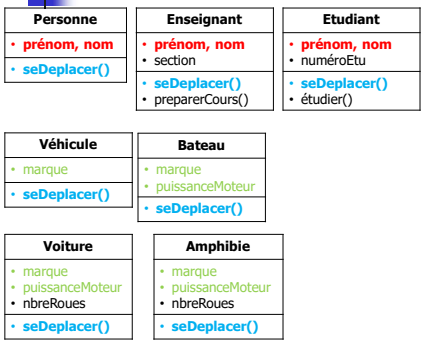
Bateau
• marque • puissanceMoteur
• seDeplacer()

Voiture
• marque • puissanceMoteur • nbreRoues
• seDeplacer()

Amphibie
• marque • puissanceMoteur • nbreRoues
• seDeplacer()

7

Classes (exemples)

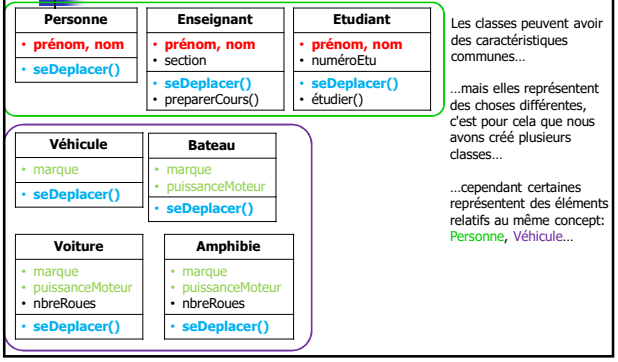


Les classes peuvent avoir des caractéristiques communes…

…mais elles représentent des choses différentes, c'est pour cela que nous avons créé plusieurs classes…

8

Classes (exemples)



Les classes peuvent avoir des caractéristiques communes…

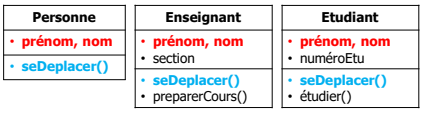
…mais elles représentent des choses différentes, c'est pour cela que nous avons créé plusieurs classes…

…cependant certaines représentent des éléments relatifs au même concept: Personne, Véhicule…

9

Héritage

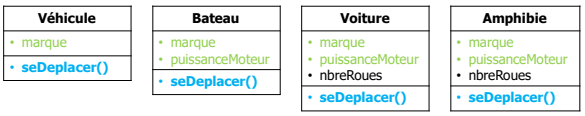
- L'héritage sert à préciser une classe existante
  - Exemples:
    - Enseignant et Etudiant sont des Personne
      - mêmes attributs, mêmes méthodes
      - **plus** des attributs/méthodes spécifiques



10

Héritage

- L'héritage sert à préciser une classe existante
  - Exemples:
    - Bateau, Voiture, Amphibie sont des Vehicule
      - mêmes attributs, mêmes méthodes
      - **plus** des attributs/méthodes spécifiques



11

L'héritage

- Traduit le principe de Généralisation / Spécification
- Permet une classification hiérarchique des classes et objets
- Principe
  - un objet spécialisé bénéficie (ou hérite) des caractéristiques d'un objet plus général auquel il ajoute ses propres caractéristiques.

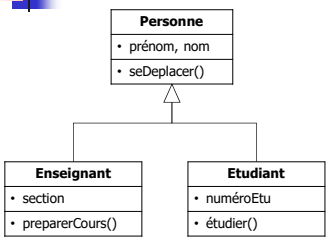
12

L'héritage

- Représentation
  - Classe associée au concept général: *classe mère, super-classe, classe de base*
  - Concept spécialisé = classe dérivée du concept général: *classe fille, sous-classe, classe dérivée*
- La relation d'héritage se traduit par:
  " La classe dérivée **est une** spécialisation de sa classe de base"
  - Exple: un Enseignant **est une** Personne (mais une Personne n'est pas un Enseignant)

13

Héritage - Représentation



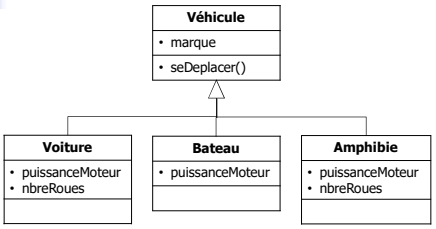
une **Personne** possède:
- 2 attributs: prénom, nom
- 1 méthode: seDeplacer()

un **Enseignant** possède:
- 3 attributs
  - 2 hérités: prénom, nom
  - 1 spécifique: section
- 2 méthodes
  - 1 héritée: seDeplacer()
  - 1 spécifique: préparerCours()

un **Etudiant** possède:
- 3 attributs
  - 2 hérités: prénom, nom
  - 1 spécifique: numéroEtu
- 2 méthodes
  - 1 héritée: seDeplacer()
  - 1 spécifique: étudier()

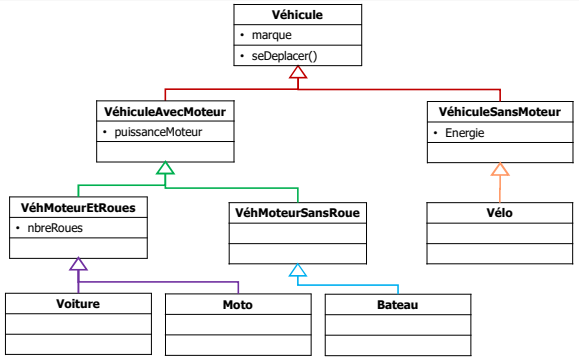
14

Héritage - Représentation



On peut avoir plusieurs niveaux, pour avoir une généralisation plus complète

15



Une **Voiture** est un **VéhMoteurEtRoues**, qui est un **VéhiculeAvecMoteur**, qui est un **Véhicule**:
- 3 attributs: marque, puissanceMoteur, nbreRoues
- 1 méthode: seDeplacer()

16

Héritage - avantages

- Limite le code
  - factorisation dans les classes mères
  - classes dérivées: seul le code spécifique reste à écrire
- Si hiérarchie bien pensée: facile de rajouter de nouvelles classes
  - Dans l'exemple précédent, il est facile de rajouter une classe Camion ou une classe Tricycle

17

Héritage - Java

- Syntaxe:
  public class NvelleClass **extends** *AutreClass*

  NvelleClass est une classe fille de AutreClass
- Accès aux éléments
  - Comme les attributs/méthodes sont hérités de la classe mère, on y accède comme d'habitude: nomObjet.nomAttribut ou nomObjet.nomMéthode()

18

Héritage - Accessibilité

- Attributs/méthodes **public**: accès depuis l'extérieur de la classe
- Attributs/méthodes **private**: accès restreint à la classe ⇒ pas d'accès depuis la classe fille

```
public class Personne {
   private String prenom,nom;

   void seDeplacer(){
      System.out.println("il se déplace");
   }
}
```

```
public class Etudiant extends Personne{
   int numero;

   public void maFonction(){
      this.prenom="Pierre";
   }
}
```

Erreur de compilation, pas accès à l'attribut

19

Héritage - Accessibilité

- Accès **protected**: accès interdit depuis l'extérieur de la classe (comme private) mais autorisé pour les classes filles

```
public class Personne {
   protected String prenom,nom;

   void seDeplacer(){
      System.out.println("il se déplace");
   }
}
```

```
public class Etudiant extends Personne{
   int numero;

   public void maFonction(){
      this.prenom="Pierre";
   }
}
```

autorisé

```
public class MainClass {
   public static void main(String[]args){
      Personne p=new Personne();
      p.prenom="Alain";
   }
}
```

interdit

20

Héritage - Accessibilité

- 3 modes d'accessibilité (attributs et méthodes)
  - public
  - private
  - protected
- Comment initialiser les attributs private hérités de la classe mère ?
  - En utilisant son constructeur
  - Rappel: constructeur = méthode permettant d'initialiser les attributs lors de l'instanciation

21

Classe dérivée - constructeur

- Objectif: initialisation des attributs de la classe fille et ceux de la classe mère
  - Attributs classe mère public (ou protected): OK
  - Attributs classe mère private
    - pas d'accès depuis la classe fille
    - utilisation du constructeur de la classe mère
    - Java: appel **super**(*liste des arguments*);
- L'utilisation du constructeur de la classe mère peut aussi se faire même pour des arguments public ou protected

22

Héritage: exemple (1/2)

```
public class Personne {
   private String nom,prenom;
   private int age;

   public Personne(String P, String N, int age){…}
   public Personne(){…}
}
```

L'appel au constructeur de la classe mère est la **1ère chose à faire** dans le constructeur de la classe fille

```
public class Enseignant extends Personne{
   private int section;

   public Enseignant(String N, String P, int a, int sect) {
      super(P, N, a);
      section=sect;
   }
}
```

23

Héritage: exemple (2/2)

```
public class Personne {
   private String nom,prenom;
   private int age;

   public Personne(String P, String N, int age){…}
   public Personne(){…}
}
```

L'appel au constructeur de la classe mère est la **1ère chose à faire** dans le constructeur de la classe fille

Si aucun appel n'est spécifié, tout se passe comme s'il y avait un appel au **constructeur par défaut**

```
public class Enseignant extends Personne{
   private int section;

   public Enseignant(String N, String P, int a, int sect) {

      section=sect;
   }
}
```

Problème si pas de constructeur par défaut

24

Héritage - récapitulatif

- Jusqu'à maintenant, on sait
  - Instancier un objet d'une classe dérivée
    - Constructeur: appel constructeur classe mère
  - Un objet de classe fille accède aux méthodes des classes fille + mère (public)
- Est-ce suffisant ?
  - Et si la méthode héritée n'est pas satisfaisante ? Si on veut la modifier (ou la compléter pour la classe fille) ?

25

Exemple

```
public class Personne {
   protected String prenom,nom;

   public Personne(String prenom, String nom) {
      this.prenom = prenom;
      this.nom = nom;
   }

   public void afficher() {
      System.out.println(this.prenom + " " + this.nom);
   }
}
```

```
public class Etudiant extends Personne{
   int numero;

   public Etudiant(String prenom, String nom, int numero) {
      super(prenom, nom);
      this.numero = numero;
   }
}
```

```
public class MainClass {
   public static void main(String[]args){
      Etudiant e=new Etudiant("Phil","Defer",23);
      e.afficher();
   }
}
```

Qu'affiche le programme ? Phil Defer

Comment faire apparaître le numéro étudiant ?

En recréant une fonction afficher dans Etudiant

26

Redéfinition

- Redéfinition possible d'une méthode de la classe mère
  - Même nom
  - Mêmes arguments (types et nombre)
  - Même type de retour
- Lors de l'appel
  - L'interprète cherche la méthode à partir de la classe de l'objet appelant
  - Il remonte la hiérarchie jusqu'à la 1ère implémentation de la méthode

27

Exemple

```
public class Personne {
   protected String prenom,nom;

   public Personne(String prenom, String nom) {
      this.prenom = prenom;
      this.nom = nom;
   }

   public void seDeplacer(){
      System.out.println("il se déplace");
   }

   public void afficher() {
      System.out.println(this.prenom + " " + this.nom);
   }
}
```

```
public class Etudiant extends Personne{
   int numero;

   public Etudiant(String prenom, String nom, int numero) {
      super(prenom, nom);
      this.numero = numero;
   }

   public void afficher(){
      System.out.println(this.prenom + " " + this.nom +
                         " " + this.numero);
   }
}
```

```
public class MainClass {
   public static void main(String[]args){
      Personne p=new Personne("Alex","Terieur");
      p.afficher();
      p.seDeplacer();
      Etudiant e=new Etudiant("Phil","Defer",23);
      e.afficher();
      e.seDeplacer();
   }
}
```

28

Java: appel à une méthode de la classe mère

- Depuis une méthode de la classe fille, il est possible d'appeler une méthode de la classe mère: mot clé **super**

```
public class Personne {
   protected String prenom,nom;

   public void seDeplacer(){
      System.out.println("il se déplace");
   }

   public void afficher() {
      System.out.println(this.prenom + " " + this.nom);
   }
}
```

```
public class Etudiant extends Personne{
   int numero;

   public void afficher(){
      super.afficher();
      System.out.println(this.numero);
   }
}
```

Facultatif si pas d'ambiguïté

29

Polymorphisme

- Le polymorphisme permet à un objet d'une classe C de représenter
  - Un objet de la classe C
  - Un objet d'une classe dérivée de C

```
public class MainClass {
   public static void main(String[]args){
      //classique
      Personne p=new Personne("Alex","Terieur");
      Etudiant e=new Etudiant("Phil","Defer",23);

      p=e;
      e=p;
   }
}
```

un Etudiant est une Personne

une Personne n'est pas un Etudiant

30

Polymorphisme

- A quoi ça sert ?
  - On peut référencer plusieurs types d'objets (classes filles) avec une même référence
  - Cela va permettre d'avoir une utilisation plus générique de nos classes hiérarchisées
- Attention à l'accès aux méthodes: une référence à un objet de classe mère n'accède qu'aux méthodes de cette classe
  - voir exemple

31

Polymorphisme: exemple

```
public class Personne {
   protected String prenom,nom;

   public Personne(String prenom, String nom) {…}

   public void afficher() {
      System.out.println("Personne");
   }

   public void fctPersonne(){
      System.out.println("fctPersonne");
   }
}
```

```
public class Etudiant extends Personne{
   int numero;

   public Etudiant(String prenom, String nom, int numero) {…}

   public void afficher(){
      System.out.println("Etudiant");
   }

   public void fctEtudiant(){
      System.out.println("fctEtudiant");
   }
}
```

```
public class Enseignant extends Personne{
   int section;

   public Enseignant(String prenom, String nom, int section) {…}

   public void afficher(){
      System.out.println("Enseignant");
   }

   public void fctEnseignant(){
      System.out.println("fctEnseignant");
   }
}
```

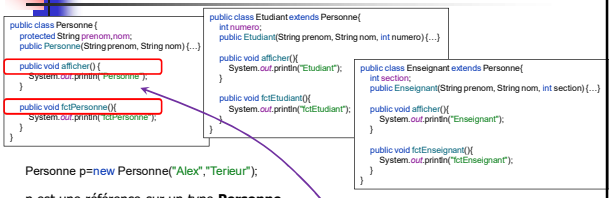
Etudiant et **Enseignant** filles de **Personne**

méthode **afficher** redéfinie dans les classes filles

chaque classe a aussi une fonction spécifique

32

Polymorphisme: exemple



Personne p=new Personne("Alex","Terieur");

p est une référence sur un type **Personne**
l'objet qu'elle référence est du type **Personne**

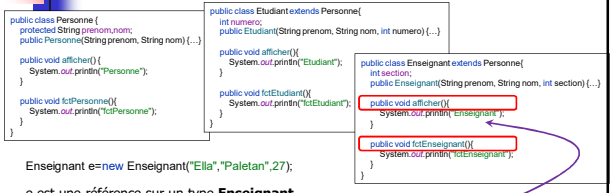
Q1: quelles sont les méthodes auxquelles p a accès ?

p.afficher();
p.fctPersonne();

Q2: qu'affiche l'instruction *p.afficher()* ? Personne

33

Polymorphisme: exemple



Enseignant e=new Enseignant("Ella","Paletan",27);

e est une référence sur un type **Enseignant**
l'objet qu'elle référence est du type **Enseignant**

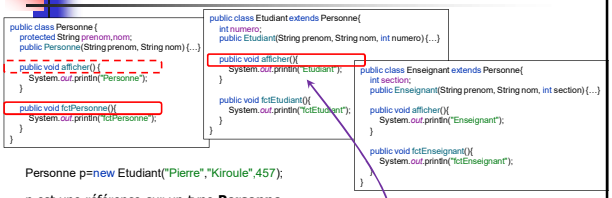
Q1: quelles sont les méthodes auxquelles e a accès ?

e.afficher();
e.fctEnseignant();

Q2: qu'affiche l'instruction *e.afficher()* ? Enseignant

34

Polymorphisme: exemple



Personne p=new Etudiant("Pierre","Kiroule",457);

p est une référence sur un type **Personne**
l'objet qu'elle référence est du type **Etudiant**

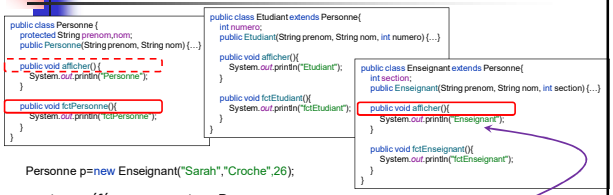
Q1: quelles sont les méthodes auxquelles p a accès ?

p.afficher();
p.fctPersonne();

Q2: qu'affiche l'instruction *p.afficher()* ? Etudiant

35

Polymorphisme: exemple



Personne p=new Enseignant("Sarah","Croche",26);

p est une référence sur un type **Personne**
l'objet qu'elle référence est du type **Enseignant**

Q1: quelles sont les méthodes auxquelles p a accès ?

p.afficher();
p.fctPersonne();

Q2: qu'affiche l'instruction *p.afficher()* ? Enseignant

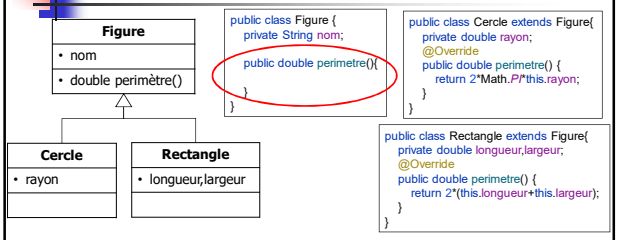
36

Polymorphisme

- Une référence R sur une classe C
  - peut référencer des objets de C et de ses classes filles C'
  - ne voit que l'interface publique de C
- Cependant
  - si R est une référence sur un objet de classe fille C'
  - et si la méthode de C est redéfinie dans C'
  - c'est cette redéfinition qui est utilisée

37

Classes abstraites




```
public class Figure {
    private String nom;

    public double perimetre(){
    }
}
```

```
public class Cercle extends Figure{
    private double rayon;
    @Override
    public double perimetre() {
        return 2*Math.PI*this.rayon;
    }
}
```

```
public class Rectangle extends Figure{
    private double longueur,largeur;
    @Override
    public double perimetre() {
        return 2*(this.longueur+this.largeur);
    }
}
```

On sait comment calculer le périmètre d'un **Cercle** ou d'un **Rectangle**, mais d'une **Figure** ?

Si on veut instancier *Figure f = new Figure();* il faut obligatoirement définir chaque méthode de la classe Figure.
Peut-on réellement décrire le calcul du périmètre d'une **Figure** (au sens large) ?

38

Classe abstraite

- Possibilité de déclarer une méthode sans la définir
  - on déclare son prototype
  - on ne donne pas son contenu
- La classe est alors incomplète
  - il est interdit d'instancier un objet de cette classe
- La classe devient **abstraite**

39

Classe abstraite - syntaxe

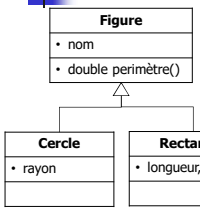
- Mot clé **abstract**
  - à préciser lors de la déclaration de la classe
  - à préciser devant les méthodes non définies

```
public abstract class Figure {
    private String nom;

    public abstract double perimetre();

    public void autreFct(){
        System.out.println("autreFct");
    }
}
```

Dans une classe abstraite il peut y avoir des méthodes abstraites et des méthodes concrètes.

40

Classes abstraites



Comme la classe **Figure** est abstraite:
- on ne peut plus créer d'instance de Figure

~~Figure f = new Figure();~~

- pour pouvoir créer une instance de Rectangle ou de Cercle il faut **obligatoirement** que les classes Rectangle et Cercle redéfinissent les méthodes abstraites de la classe mère

Rectangle r = new Rectangle();

- le polymorphisme fonctionne toujours

Figure f = new Cercle();

```
public abstract class Figure {
    private String nom;

    public abstract double perimetre();
}
```

```
public class Cercle extends Figure{
    private double rayon;
    @Override
    public double perimetre() {
        return 2*Math.PI*this.rayon;
    }
}
```

```
public class Rectangle extends Figure{
    private double longueur,largeur;
    @Override
    public double perimetre() {
        return 2*(this.longueur+this.largeur);
    }
}
```

41

Classe Object

- En Java il existe une classe particulière
  - class **Object**
- Toutes les classes que vous créerez seront automatiquement (et implicitement) des classes filles de la classe Object
  - donc une référence de type Object peut référencer tout objet (polymorphisme)

Object obj = new Cercle();

  - la classe Object contient déjà des méthodes que l'on peut redéfinir

42

Classe Object – toString()

- La méthode toString() est une méthode déjà implémentée dans la classe Object
  - son but: afficher l'objet
  - appelée implicitement quand on affiche un objet

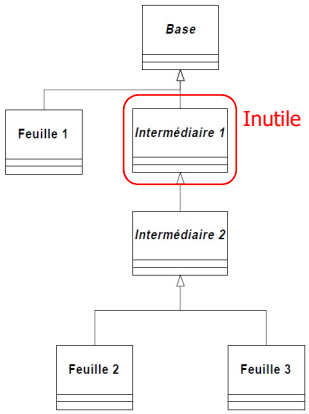
```
public class Personne {
    private String prenom, nom;

    public Personne(String prenom, String nom) {…}

    @Override
    public String toString() {
        return this.prenom + " " + this.nom;
    }
}
```

```
public static void main(String[] args) {
    Personne p =new Personne("Pierre","Kiroule");

    System.out.println(p.toString()); Pierre Kiroule

    System.out.println(p); Pierre Kiroule
}
```

43

Héritage -

- Attention aux hiéra
  - Ne pas alourdir la c
  - Exemple
    - Animal -> Chien, C
    - Chien -> ChienNoir
      chien n'est pas disc
  - Eviter les classes ir
- Héritage multiple…

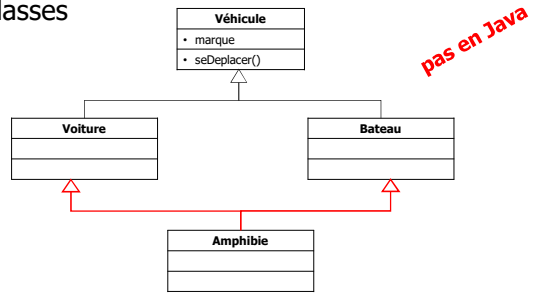


44

Héritage multiple

- Il existe la possibilité d'hériter de plusieurs classes

pas en Java



45

Polymorphisme - compléments

- Il existe un opérateur permettant de tester la nature d'un objet:
  - objet **instanceof** classe
  - le résultat est un booléen
  - Exemple: Personne p = new Personne();
    boolean x = p instanceof Personne; *//x vaut true*

46

Polymorphisme - compléments

- Opérateur de cast
  - Il est possible de caster une référence d'une classe mais la transformation doit être correcte

```
Personne p = new Etudiant("Michelle","Lamère", 233);
Etudiant e = new Etudiant("Alex", "Terieur",215);
Personne q = new Enseignant("Pierre","Kiroule", 27);

Personne x;
Etudiant y;

x=e; //possible (polymorphisme x peut référencer les Personne et classes filles)
y=p; //impossible (p est une réf sur une Personne), compilateur refuse
y=(Etudiant) p; //possible (car p référence un Etudiant)
y=(Etudiant) q; //compilateur OK car q ref sur Personne, mais plante à l'exé (q n'est pas un Etudiant)
```

Attention: une référence n'accède qu'à la partie publique de son type (exple: y n'accède qu'aux méthodes de Etudiant, x n'accède qu'aux méthodes de Personne + celles redefinies dans Etudiant)