Développement en JAVA

Objets et héritage

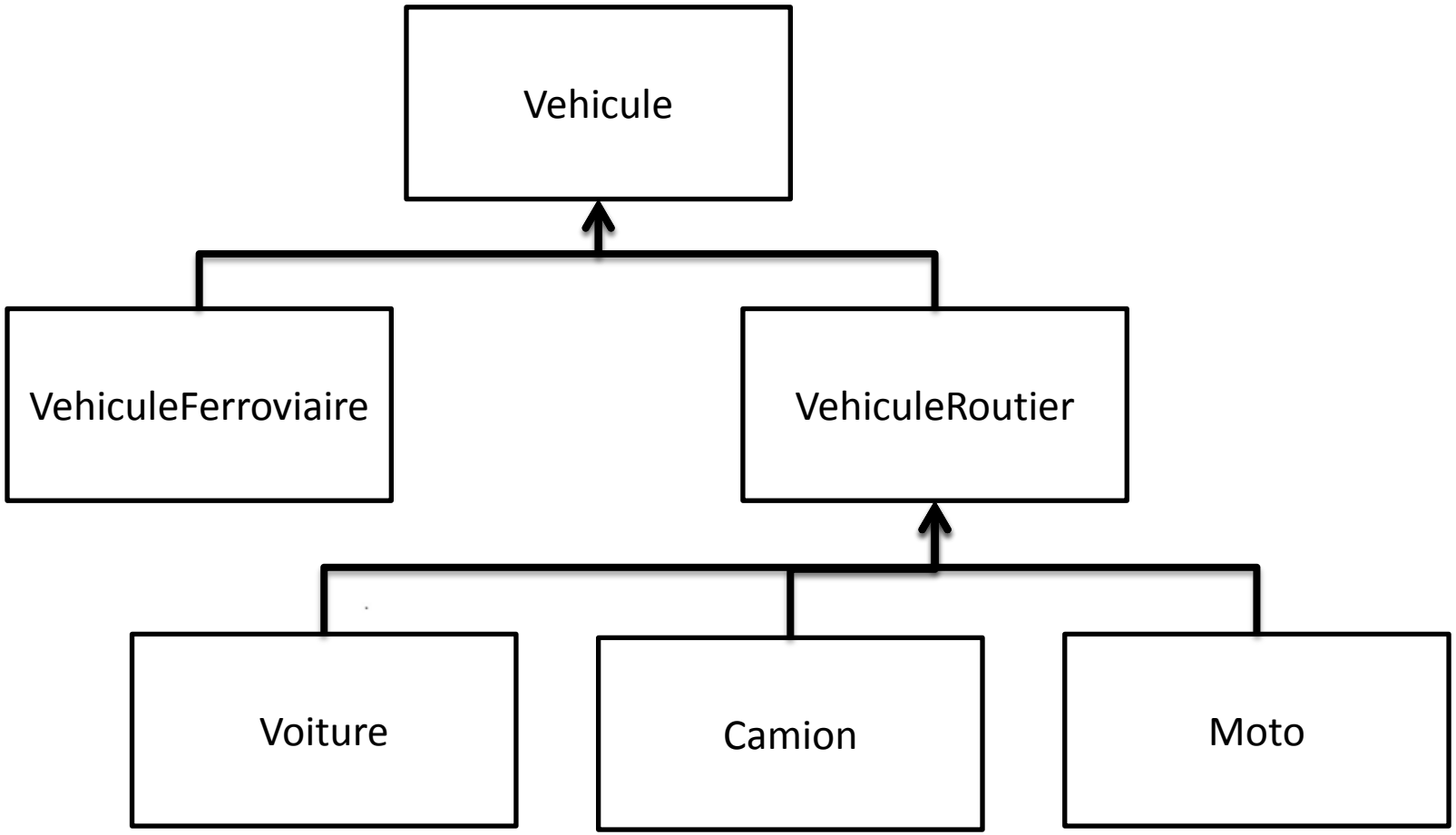
Agenda

- Héritage
- Interface
- Polymorphisme
- Classe anonymes
- Héritage multiple ?
- Classe représentant les types primitifs, auto boxing et parsing

Héritage

- L'héritage est un des atouts les plus puissants de la programmation orientée objet
- L'héritage permet de
  - Structurer les modèles de données
  - Factoriser des fonctionnalités
- Contrairement au C++, le Java ne supporte pas l'héritage multiple

Exemple de hiérarchie



Héritage

- Le mot clé `extends` permet de définir une relation d'héritage

```
public class Vehicule{
  protected String nom;
  protected short capacite;
}

public class VehiculeFeroviaire extends Vehicule{

}

public class VehiculeRoutier extends Vehicule{

}
```

Constructeur

- Le mot clé super permet de faire appels à la classe

```
public class VehiculeRoutier extends Animal{
  protected String carburant;
  protected short nbRoues;

  public VehiculeRoutier(String nom, short capacite, String
carburant, short nbRoues){
    super(nom,capacite);
    this.carburant = carburant;
    this.nbRoues = nbRoues;
 }
}
```

Surcharge et redéfinition

- On parle de surcharge lorsque une Classe propose plusieurs implémentations pour une même méthode
  - Par exemple lorsqu'on propose plusieurs constructeurs on parle de surcharge
- On parle de redéfinition lorsque une Classe propose une implémentation pour une méthode définie d'une classe mère
  - La signature doit être la même
  - On dit que le contrat proposé par la classe doit être respecté
  - Par exemple lorsqu'on implémente la méthode toString() heritée de Object, on parle de redéfinition

Classe abstraite

- Une classe abstraite ne peut pas être instanciée

```
public abstract class Vehicule{
  protected String couleur;
  public Vehicule(){
    this.couleur = "inconnue";
  }
}

public static void main(String args[]){
  Meuble m = new Vehicule(); // Erreur de compilation
}
```

- Elle permet de définir un ensemble d'attributs et de comportement partagée par une hiérarchie de classe qui n'ont de sens que redéfinis

Classe abstraite

- Une classe abstraite peut notamment posséder des méthodes abstraites

```
public abstract class Vehicule{
  protected String couleur;
  public Meuble(){
    this.couleur = "inconnue";
  }

   // La de placer un vehicule devra être définie par les
classes filles
   public abstract deplacer();
}
```

Polymorphisme

- Le polymorphisme désigne la capacité d'une méthode héritée à avoir différentes implémentations

```
abstract class Forme {
  abstract float aire() ;
}

class Carre extends Forme {
  float cote;
  float aire() {
    return cote * cote;
   }
}

class Cercle extends Forme {
  float rayon;
  float aire() {
    return Math.PI*rayon*rayon;
 }
}
```

Polymorphisme

- Pour qu'une méthode soient considéré comme polyphorme elle doit redéfinir une méthode de la classe mère
- On peut ainsi utiliser des méthodes spécifiques aux classes filles en ne connaissant pas leur type

```
for(Forme forme: formes){
  System.out.println("Aire de la forme: " + forme.aire());
}
```

Classe final

- Une classe déclarée final est une classe ne pouvant plus être dérivée
- Déclarée une classe final permet de « figer » une partie hiérarchie
- Cela permet également d'éviter qu'une classe fille brise l'encapsulation

Interfaces

- Le langage Java ne supporte pas l'héritage multiple
- Cependant la complexité du métier nous amène souvent à avoir besoin de fonctionnalités supportées par une seule partie de notre hiérarchie de classe
- Les interfaces répondent à ce problème
- On leur attribue généralement un nom correspondant à un comportement

Interfaces

- Les interfaces sont des classes:
  - Abstraites
  - Qui ne possèdent pas d'attributs
  - Dont toutes les méthodes sont abstraites
- En résumé, c'est juste une liste de signatures
- Une classe **peut implémenter une ou plusieurs interfaces**
- Pour implémenter une interface on utilise le mot clé **implements**
- Une classe qui implémente une interface doit implémenter toutes les méthodes de cette interface

Interfaces

```
public interface immatriculable {
  public void immatriculer();
}
// Seuls certains véhicules sont immatriculables

// Tous les vehicules routier sont immatriculables
public abstract VehiculeRoutier extends Vehicule
implements Immatriculable{
}

// L'implémentation se fera dans les classes filles
public voiture extends VehiculeRoutier{
  public void immatriculer(){
    System.out.println("Voilà comment on immatricule une
voiture");
  }
}
```

Interfaces

- Le mot clé extends doit toujours déclaré avant le mot clé implements
  - Sinon on obtient une erreur de compilation
- En général, on préféra définir des méthodes abstraites dans des interfaces plutôt que dans des classes abstraites
- Définir un comportement dans une classe abstraite implique que toutes classes dérivées propose une implémentation pour ce comportement
- Dans l'exemple précédent, la méthode immatriculer n'aurait pas de sens pour un train.

Interfaces standards

- Le JDK inclue de nombreuses interfaces standards:
  - Serializable
  - Clonable
  - Comparable
  - …

Classe génériques

- Les classes génériques sont des classes qui doivent être paramétriser en fonction d'une autre classe
- C'est notamment le cas de l'ensemble des classes de l'API Collection