Le langage Java

Approche orientée objet – Héritage, constructeurs et abstraction

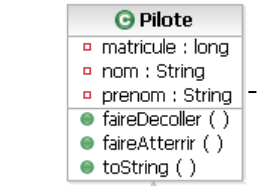
Programme détaillé ou sommaire

Héritage et constructeurs
Mot-clé final
Classe abstraite: problématique
Déclaration de la classe abstraite
Classe abstraite et constructeurs
Méthode abstraite
Exemple d'abstraction
Contraintes liées à l'abstraction

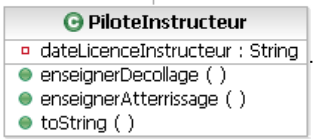
Héritage et constructeurs (1/3)

Si la **classe mère possède un constructeur avec au moins un paramètre**, toute classe fille **est obligée** d'appeler le constructeur de la classe mère.

Exemple du constructeur de la classe **Pilote** qui a 2 paramètres:




```
public Pilote(String nom, String prenom){
  // appel implicite à super();
  this.nom = nom;
  this.prenom = prenom;
}
```



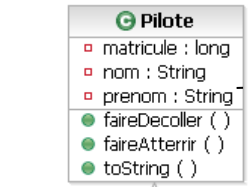

```
public PiloteInstructeur(String nom, String prenom, Date date){
  super(nom, prenom);
  dateLicenseInstructeur = date;
}
```

super(…) doit être la première instruction du constructeur

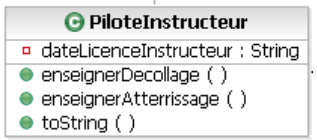
Héritage et constructeurs (2/3)

Appel d'un constructeur de la classe-mère (ou super-classe) avec **super(…)** puis traitements spécifiques

Attention à ne pas confondre avec utilisation de this(…)




```
public Pilote(String nom, String prenom){
  // appel implicite à super();
  this.nom = nom;
  this.prenom = prenom;
}
```




```
public PiloteInstructeur(String nom, String prenom, Date date){
  super(nom, prenom);
  dateLicenseInstructeur = date;
}
```

super(…) doit être la première instruction du constructeur

Héritage et constructeurs (3/3)

Si la classe mère a un constructeur sans paramètre ou pas de constructeur, l'appel du constructeur de la classe mère depuis la classe fille est **réalisé de manière implicite**.

Exemple de la classe Animal et d'une classe fille Mammifere:

```
class Animal {
       public Animal(){
       }
}
```

```
class Mammifere extends Animal {
       public Mammifere(){
         super(); ➔ l'instruction super() est exécutée que vous la mettiez ou non
       }
}
```

Mot clé final

Classe avec le mot-clé final

interdit la création de sous-classes

```
public final class PiloteInstructeur {
   // attributs, constructeurs, méthodes
}
```

Méthode avec le mot-clé final

interdit de redéfinir la méthode dans une sous-classe

```
public final void faireDecoller(Avion unAvion){
    // code pour faire décoller l'avion
}
```

Attribut avec le mot-clé final

pour un attribut, **final** signifie « constant »

Atelier (TP)

OBJECTIFS : mettre en œuvre les constructeurs dans le cadre d'une hiérarchie de classes

DESCRIPTION : Dans le TP qui suit, vous allez mettre en place un héritage et implémenter les constructeurs.

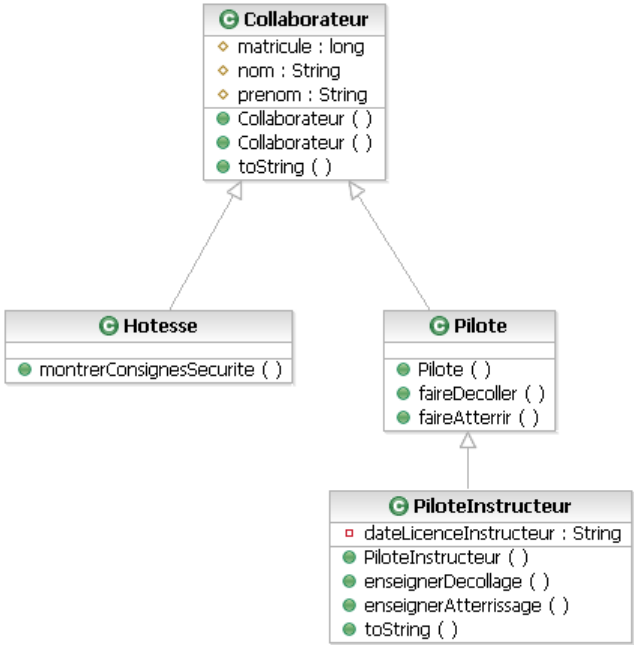
Classe abstraite : Problématique

Classe Collaborateur :

- Créée pour factoriser attributs et méthodes
- Un collaborateur est une notion abstraite

new Collaborateur() *n'a pas de sens*

Comment l'interdire ?



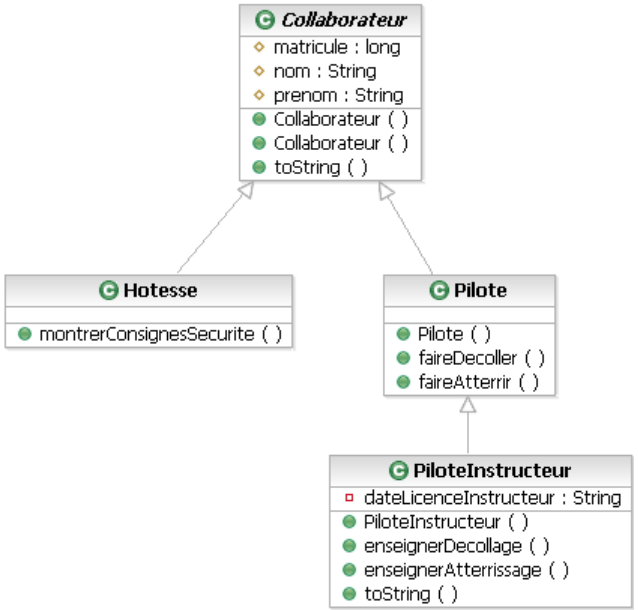
Déclaration de classe abstraite

Collaborateur est abstraite

Notée en italique en UML

Mot-clé abstract en Java

```
public abstract class Collaborateur {
  protected long matricule;
  protected String nom;
  protected String prenom;

  public Collaborateur(String nom, String prenom){
    this.nom = nom;
    this.prenom = prenom;
  }
  public String toString() {
    return prenom + " " + nom;
  }
}
```



Classe abstraite et constructeurs

Une classe abstraite peut avoir des constructeurs

Factorisation de l'initialisation

Appelés par les constructeurs des sous-classes

Mais appel direct (new) interdit

```
public abstract class Collaborateur {
  protected String nom;
  protected String prenom;

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

```
public class Pilote extends Collaborateur{
  public Pilote(String nom, String prenom){
    super(nom, prenom);
  }
}
```

Méthode abstraite

Mot-clé **abstract**

```
public abstract long calculerSalaire();
```

Une méthode abstraite :

- n'a pas de corps

  Pas d'accolades { }. Se termine par ';'
- Ne possède qu'une signature

Redéfinition obligatoire dans les sous-classes concrètes

Ne peut pas être privée, ni finale

Exemple abstraction

Collaborateur

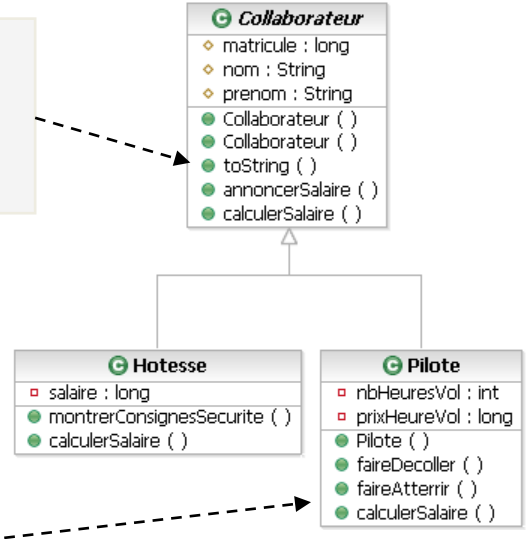
```
public abstract long calculerSalaire();

public void annoncerSalaire(){
  System.out.println("Salaire : " + calculerSalaire());
}
```

Hotesse

```
public long calculerSalaire() {
  return salaire;
}
```

Pilote

```
public long calculerSalaire() {
  return nbHeuresVol*prixHeureVol;
}
```



Rappel : on recherche toujours les méthodes à partir de la classe courante

Contraintes liées à l'abstraction

La redéfinition d’une méthode abstraite

- est obligatoire dans une sous-classe concrète
- sinon la sous-classe doit elle-même être abstraite

Une classe abstraite

- ne peut pas avoir d’instance
- peut contenir des méthodes concrètes
- peut contenir des constructeurs

Atelier (TP)

OBJECTIFS : mettre en œuvre un héritage

DESCRIPTION : Dans le TP qui suit, vous allez mettre en place un héritage et mettre en œuvre les divers mécanismes vus durant ce cours: chaîne de construction et abstraction.