Héritage

Noury Bouraqadi
http://car.mines-douai.fr/noury

Option ISIC
Ecole des Mines de Douai

Définition d'un prof

- Champs :
  - **nomPrenom** : String
  - **dateNaissance** : Date
  - **matieresEnseignees** : Set of String
- Messages valides :
  - **nomPrenom** (lecture nomPrenom)
  - **nomPrenom: chaineDeCaracteres** (modif. nomPrenom)
  - **dateNaissance: uneDate** (modif. dateNaissance)
  - **age** (calcul et retourne l'age)
  - **ajouteMatiere: nomMatiere** (ajoute une matière)
  - **enseigneMatiere: nomMatiere** (retourne true si le prof enseigne une matière)

Définition d'un élève

- Champs :
  - **nomPrenom** : String
  - **dateNaissance** : Date
  - **notes** : Dictionary (nomMatiere : String -> note : Float)
- Messages valides :
  - **nomPrenom** (lecture nomPrenom)
  - **nomPrenom: chaineDeCaracteres** (modif. nomPrenom)
  - **dateNaissance: uneDate** (modif. dateNaissance)
  - **age** (calcul et retourne l'age)
  - **note: unReel matiere: nomMatiere** (ajout d'une note)
  - **noteMatiere: nomMatiere** (lecture d'une note)
  - **moyenne** (calcule et retourne la moyenne)

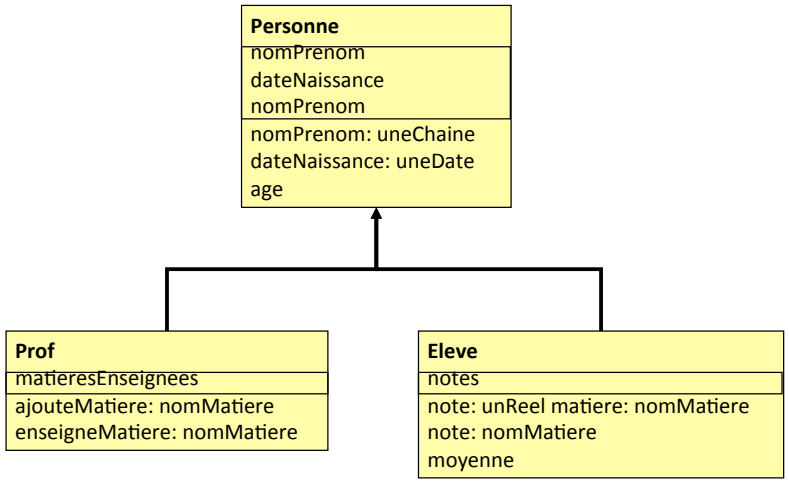
Besoin de factoriser le code

- Deux classes différentes
  - Prof et Eleve
- Mais, …
  - des champs identiques :
    - nomPrenom (String)
    - dateNaissance (Date)
  - et des méthodes identiques :
    - nomPrenom
    - nomPrenom: chaineDeCaracteres
    - dateNaissance: uneDate
    - age
- Solution pour éviter la duplication de code : **l'héritage**

Héritage

- Héritage = partage de code
- Le code partagé est défini dans une classe
  - La **super-classe**
- Partage le code entre plusieurs classes
  - Les **sous-classes**
  - On dit : les sous-classes héritent de la super-classe

Prof et Eleve héritent de la classe Personne



Définition de la super-classe Personne

Object subclass: #Personne
instanceVariableNames: '**nomPrenom dateNaissance**'
classVariableNames: ''
category: 'Exemple'

Définition des sous-classes Prof et Eleve

- Personne subclass: #Prof
  instanceVariableNames: '**matieresEnseignees**'
  classVariableNames: ''
  category: 'Exemple'
- Personne subclass: #Eleve
  instanceVariableNames: '**notes**'
  classVariableNames: ''
  category: 'Exemple'

Conséquences de l'héritage - 1

- Les instances des sous-classes disposent des champs déclarés dans les superclasses
  - Champs d'une instance de Prof :
    - nomPrenom, dateNaissance,
    - matieresEnseignees
  - Champs d'une instance de Eleve :
    - nomPrenom, dateNaissance,
    - notes

Conséquences de l'héritage - 2

- Les instances des sous-classes savent répondre aux messages correspondant aux méthodes des superclasses
  - Message valides pour les instances de Prof :
    - nomPrenom, nomPrenom: uneChaine, dateNaissance: uneDate, age,
    - ajouteMatiere: nomMatiere, enseigneMatiere: nomMatiere
  - Messages valides pour les instances de Eleve :
    - nomPrenom, nomPrenom: uneChaine, dateNaissance: uneDate, age,
    - note: unReel matiere: nomMatiere, note: nomMatiere, moyenne

L'héritage est transitif

- Si : Eleve hérite de Personne
  - Eleve est une sous-classe directe de Personne
- Et : EleveSalarié hérite de Eleve
  - EleveSalarié est une sous-classe directe de Eleve
- Alors : EleveSalarié hérite de Personne
  - EleveSalarié est une sous-class indirecte de Personne
  - Les instances de EleveSalarié
    - Possèdent les champs déclarés par EleveSalarié et toutes ses superclasses (Eleve, Personne, Object)
    - Savent répondre aux messages correspondants aux méthodes de EleveSalarié et toutes ses superclasses

Multiplicité d'héritage

- Une classe peut ne pas avoir de super-classes
  - C'est une classe racine (de l'arbre d'héritage)
- Héritage simple
  - maxi 1 super-classe directe
  - c'est le cas de Smalltalk
- Héritage multiple
  - possibilité d'avoir plusieurs super-classes directes
  - disponible dans d'autres langages comme le C++
    - Pose pas mal de problèmes
    - N'est pas adopté dans les langages récents

Quelle super-classe utiliser ?

- Soit x une instance de la classe A
  - On dit que x est un A
  - exemple : jean est un Eleve
- Soit B une autre classe
- B est une super-classe valide pour A si on peut dire que x est un B
  - exemple 1 : jean est une Personne
    - Personne est une super-classe valide de Eleve
  - exemple 2 : jean n'est pas une Voiture
    - La classe Voiture n'est pas une super-classe valide de Eleve
- Par défaut, utiliser **Object**
  - Object définit diverses méthodes nécessaires à l'utilisation des objets

Héritage
&
Liaison message - méthode

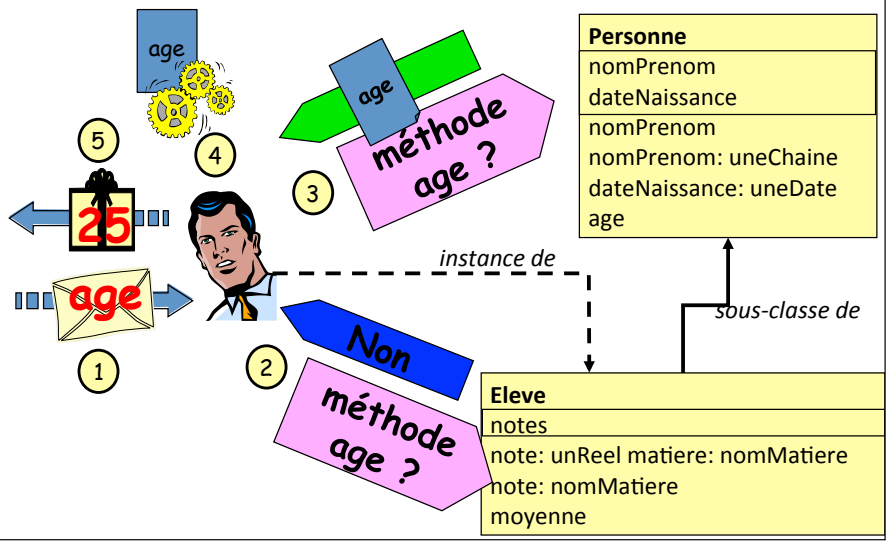
Recherche de la méthode à exécuter - 1

- Method lookup =
  - "La recherche de méthode" en anglais
- La liaison message-méthode =
  - Retrouver la méthode à exécuter en réponse à un message
- La liaison
  - peut-être dynamique
    - si la recherche est effectuée à l'exécution
    - C'est le cas dans Smalltalk
  - ou statique
    - si la recherche est effectuée à la compilation

Recherche de la méthode à exécuter - 2

- Quand un objet reçoit un message,
- il cherche dans sa classe s'il y a une méthode avec le même sélecteur
- s'il ne la trouve pas, il poursuite la recherche dans la super-classe directe
- la recherche se poursuit de super-class en super-classe jusqu'à :
  - trouver une méthode
    - exécuter la méthode trouvée
  - l'échec de la recherche dans la super-classe racine
    - erreur

Recherche de la méthode à exécuter



Masquage & Redéfinition

- Une sous-classe n'a pas le droit de déclarer un champ qui a le même nom qu'un champ hérité
  - EleveSalarié ne doit pas déclarer un champ nomPrenom
- En revanche, une classe peut définir une méthode de même sélecteur qu'une méthode héritée
  - On dit que la sous-classe **redéfinit** la méthode en question
  - EleveSalarié peut redéfinir la méthode moyenne

Réutilisation des méthodes masquées

- Redéfinition de la méthode moyenne dans EleveSalarie
  1. Effectue le calcul de la moyenne des notes
     - Traitement identique pour les instances de Eleve
  2. Ajoute un bonus : 1 point
  3. Retourne la moyenne améliorée
- Comment éviter la duplication du code de l'étape 1 ?
  - Réutiliser la méthode moyenne définie dans Eleve
  - Impossible avec l'envoi du message **moyenne** à **self**
    - Boucle infinie
- Solution : envoi de message à super

La pseudo-variable super

- super référence l'objet courant…
  - Comme self
- … mais, dans le contexte de la super-classe…
  - Début de la recherche de méthode dans la superclasse de la classe qui contient l'envoi de message
  - Contrairement à self : début de la recherche toujours dans la classe de l'objet
- … et sert uniquement pour l'envoi de message
  - Contrairement à self qui peut être utilisée comme paramètre d'un message ou comme valeur de retour

Illustration du "super" avec la représentation textuelle des objets

- monObjet printString
  - Retourne une chaîne de caractère décrivant l'objet
  - **(BankAccount new) printString**

    ⇒ **'a BankAccount'**

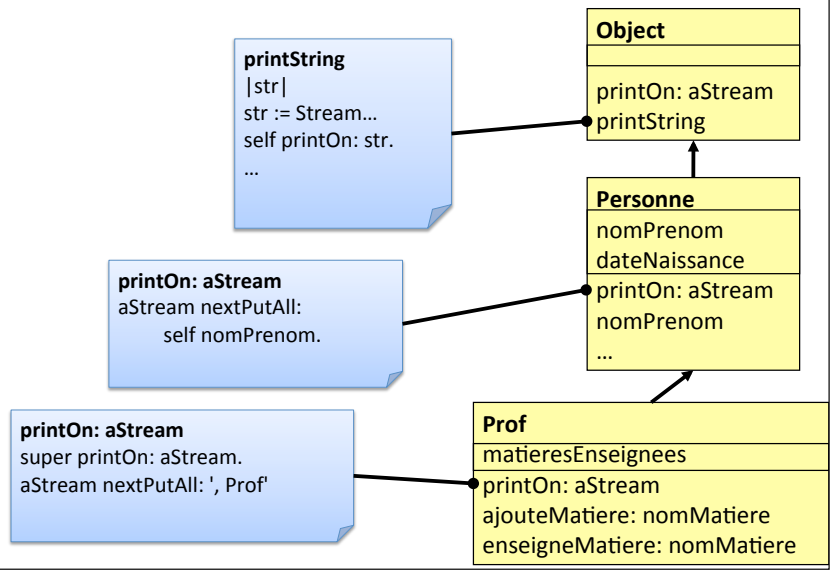
- La méthode printString est définie dans la classe Object
  - Elle utilise la méthode **printOn:**
  - Les sous-classes doivent redéfinir **printOn:** pour modifier la représentation textuelle

Illustration du "super" avec la représentation textuelle des objets

- Exemple de méthode printOn:

```
Lampe>>printOn: aStream
        aStream
          nextPutAll: 'Lampe' ;
          space.
        self estAllumee ifFalse: [
            ^aStream nextPutAll: 'eteinte'].
        aStream
          nextPutAll: 'allumée de couleur';
          space.
        self couleur printOn: aStream
```

Représentation textuelle des profs



Envoi du message printString à une instance de Prof - 1

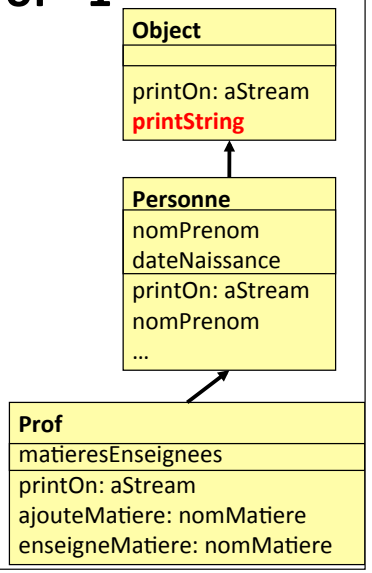
- |unProf|

unProf nomPrenom: 'Michel Vaillant'.

unProf **printString**

- Début de la recherche dans Prof
  - pas de méthode printString
- Poursuite de la recherche dans Personne
  - pas de méthode printString
- Poursuite de la recherche dans Object
  - méthode printString trouvée
- Exécution de la méthode trouvée

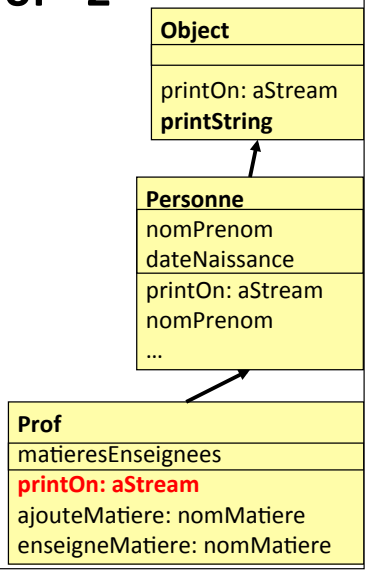
  |str|
  str := Stream...
  self printOn: str.
  ...
  ^str contents



Envoi du message printString à une instance de Prof - 2

- self printOn: str
  - Dans une méthode de Object
- Début de la recherche dans Prof
  - méthode printOn: trouvée
- Exécution de la méthode trouvée

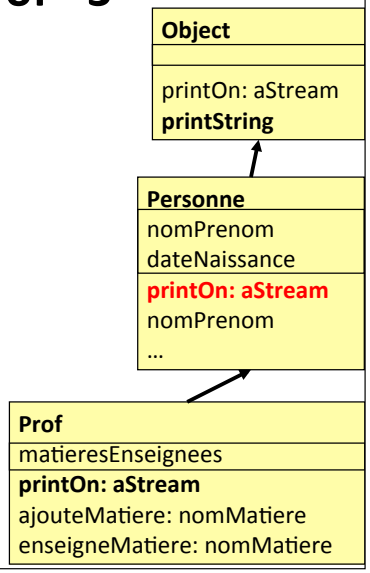
printOn: aStream
  super printOn: aStream.
  aStream nextPutAll: ', Prof'



Envoi du message printString à une instance de Prof - 3

- super printOn: aStream
  - Dans une méthode de Prof
- Début de la recherche dans Personne
  - méthode printOn: trouvée
- Exécution de la méthode trouvée

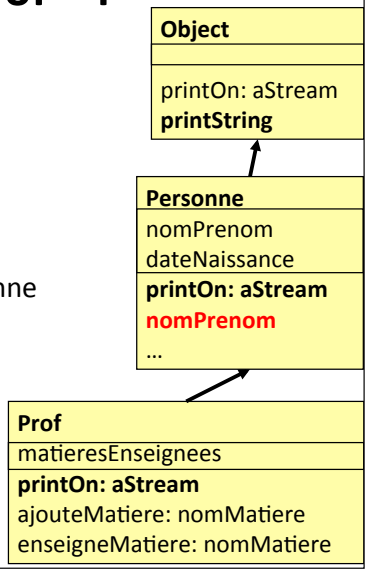
printOn: aStream
  aStream nextPutAll: self nomPrenom



Envoi du message printString à une instance de Prof - 4

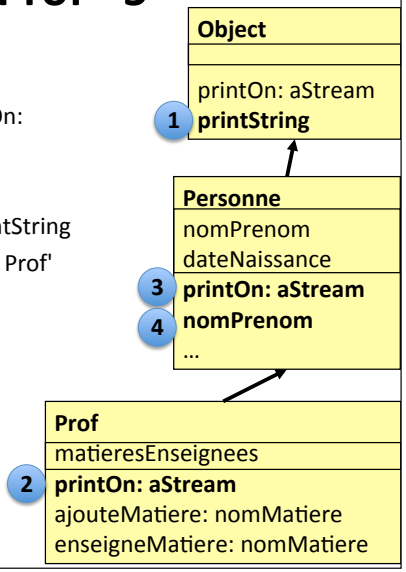
- self nomPrenom
  - Dans une méthode de Personne
- Début de la recherche dans Prof
  - pas de méthode nomPrenom
- Poursuite de la recherche dans Personne
  - méthode nomPrenom trouvée
- Exécution de la méthode trouvée

nomPrenom
  ^ nomPrenom



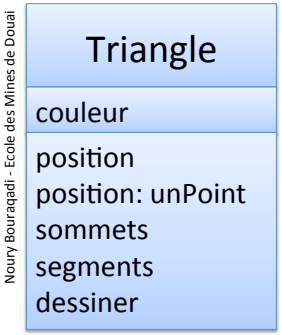
Envoi du message printString à une instance de Prof - 5

- Terminaison de Personne>>printOn:
- Suite et fin de l'exécution de Prof>>printOn:
  - aStream nextPutAll: ', Prof'
- Suite et fin de l'exécution de Object>>printString
  - Retour de la chaîne : 'Michel Vaillant, Prof'

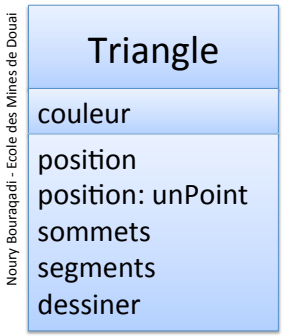


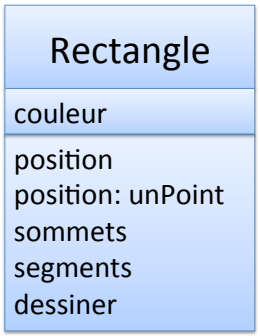
Classes Abstraites

Figures Géométriques

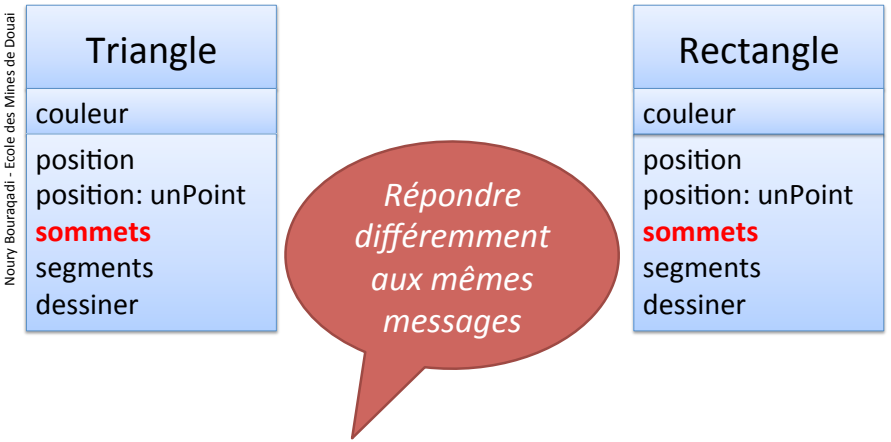


Figures Géométriques

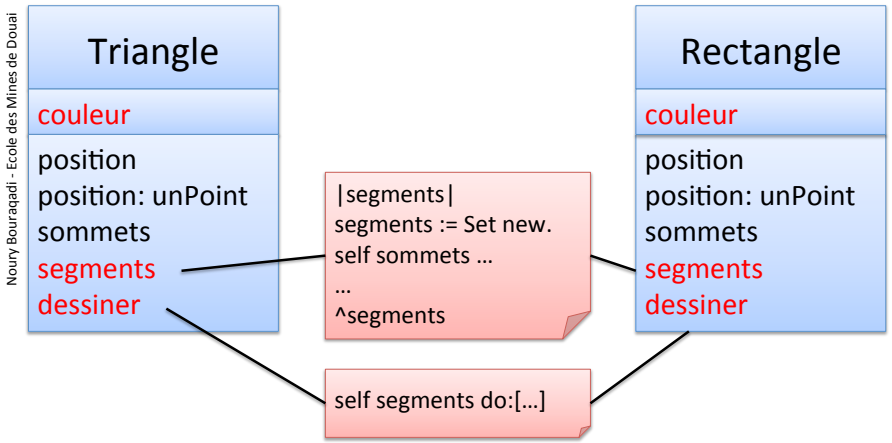




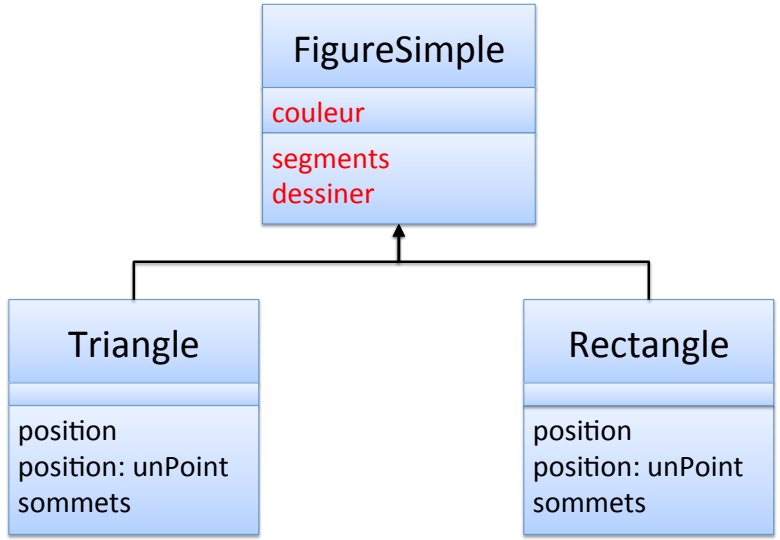
Polymorphisme



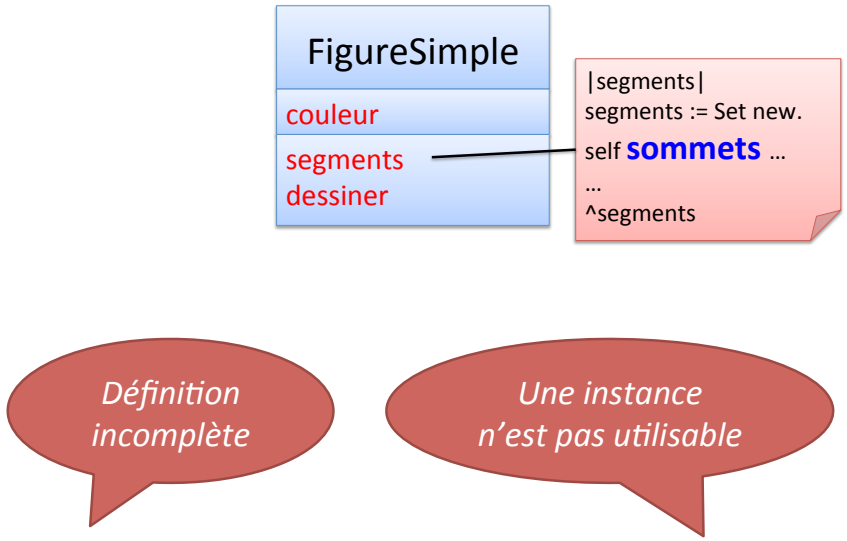
Redondances !



L'héritage pour factoriser



Classe abstraite



Méthodes abstraites