Langages objets

Redéfinition des méthodes dynamiques
(bloc 11)

M2 Pro CCI, Informatique
Emmanuel Waller, LRI, Orsay

résumé des épisodes précédents

- généralités
  - découverte Java, prise en main environnement, débugage
- Java sans objets (exactement comme C)
  - types primitifs, tableaux, objets (struct + pointeur), imbrication
  - opérateurs et expressions, instructions de contrôle, fonctions (statiques)
- concepts objets
  - méthodes dynamiques, héritage

exemple : complément cours précédent

- Rappel : rationnels : num, dén, afficher, modifier
- on veut maintenant à la fois des rationnels, et en plus gérer des rationnels avec une couleur, « colorés » (scolaire, mais clair) :
  - champs : num, den, couleur
  - méthodes : afficher (num, den, couleur), modifier, repeindre
- (rem : on remplace simplement les points par les rationnels de l'avant-dernier cours)

- Sans tableau : Ex0.java (correspond exactement à cours 10, Ex3.java)
- Avec tableau, mais on n'affichera pas la couleur des rationnels colorés : Ex0bis.java (correspond exactement à rajouter en plus cours 9, Ex2.java : la gestion du tableau de rationnels)

redéfinition des méthodes dynamiques

- but, principe, fonctionnement, vocabulaire
- un détail
- redéfinition méthodes et dérivations successives
- parcours d'un ensemble hétérogène
- remarque (méthodes dynamiques)
- redéfinition d'un champ statique
- factorisation, réutilisation
- choix méthode statique ou dynamique

but

- On dispose pendant l'exécution, dans une case de type Rationnel, d'un objet
- Il est de la classe Rationnel ou de RationnelColoré, mais on ne sait pas laquelle
- Dans tous les cas on veut l'afficher avec toutes ses spécificités :
  - Si c'est un Rationnel : num, dén
  - Si c'est un RationnelColoré : num, dén, couleur

- Ex : r = new ... // Rationnel ou RationnelColoré
- Problème :
  - r.afficher() : incorrect : n'affiche pas la couleur
  - r.afficherRationnelColore() : incorrect : ne compile pas (rappel)

intuition

- Intuitivement on voudrait :

```
if (r est de la classe Rationnel)
    r.afficher() // num, dén
else // il est donc de la classe RationnelColoré
    r.afficherRationnelColoré() // num, dén, couleur
```

- Comment faire ?

rappel : notion de surcharge

- un symbole de fonction est dit *surchargé* s'il est associé à plusieurs codes
- ex :
  - 4 + 5 et 4.0 + 5.0
  - Compte.affiche et Client.affiche
    - même nom de fonction dans deux classes
- possible de même pour méthodes dynamiques :

```
Compte c1; Client c2;
c1.affiche(); c2.affiche();
```

rappel : comment choisit Java ?

- Il faut choisir un code pour le nom de focntion surchargé
- résolution de la surcharge
- +, méthodes statiques : compilateur devine à partir du contexte
- méthodes dynamiques : r.afficher() : pendant l'exécution la JVM demande à l'objet sa classe, et elle exécute le code de cette classe

intuition (suite)

- Donc une méthode dynamique fait le test :

  if (r est de la classe Rationnel)

principe

- on va simplement utiliser la surcharge
- On va simplement remplacer le nom afficherRationnelColoré par afficher
- Pendant l'exécution, la JVM demande à l'objet sa classe, et elle exécute le code de cette classe :

```
if (r est de la classe Rationnel)
    r.afficher() // code de Rationnel : num, dén
else // il est donc de la classe RationnelColoré
    r.afficher() //code de RationnelColoré :  num, dén, couleur
```

exemple

- afficher rationnels et rationnel colorés avec leurs spécificités
- Ex1.java
- démonstration
- déroulement mémoire

redéfinition des méthodes dynamiques

- but, principe, fonctionnement, vocabulaire
- un détail
- redéfinition méthodes et dérivations successives
- parcours d'un ensemble hétérogène
- remarque (méthodes dynamiques)
- redéfinition d'un champ statique
- factorisation, réutilisation
- choix méthode statique ou dynamique

Fonctionnement, vocabulaire

- on dit que la méthode afficher de RationnelColoré *redéfinit* celle de sa classe de base Rationnel
- remarques :
  - les types des paramètres et la valeur de retour de la méthode afficher redéfinie doivent être ceux de la méthode afficher de la classe de base (il existe cas plus général : pas vu en CCI)
  - aucune contrainte sur son code

- comment a lieu le choix entre les deux codes de méthode (= résolution de la surcharge) ?
  - Ce n'est pas le compilateur qui choisit :
    - Impossible car on ne sait pas en général quel objet il y aura dans la variable
    - En effet, à un moment il peut y avoir un objet d'une classe, puis plus tard un objet d'une autre classe (ex : tableau hétérogène ci-dessous)
  - c'est pendant l'exécution que, en fonction de la classe effective de l'objet receveur (et non de celle de la variable qui contient sa référence), que la JVM choisit le code correspondant

- c'est la *liaison dynamique*
- elle donne son nom aux méthodes dynamiques
- *c'est leur raison d'être*

- en résumé : on utilise donc pour cela à la fois:
  - héritage
  - redéfinition (via surcharge)
  - liaison dynamique (résolution dynamique de la surcharge)

redéfinition des méthodes dynamiques

- but, principe, fonctionnement, vocabulaire
- un détail
- redéfinition méthodes et dérivations successives
- parcours d'un ensemble hétérogène
- remarque (méthodes dynamiques)
- redéfinition d'un champ statique
- factorisation, réutilisation
- choix méthode statique ou dynamique

Remarque de détail

- Considérons le code :

```
class RationnelColore extends Rationnel { ...
  public void afficher() {
    this.afficher(); // au lieu de super.afficher()
    System.out.println("et ma couleur est "+couleur); }
```

- on a un appel récursif : pas ce qu'on veut
- préciser qu'on veut affiche de la classe de base :

  super.affiche() (cf Ex1.java)

- (rem : dans certains cas, pas nécessaire utiliser super)

redéfinition des méthodes dynamiques

- but, principe, fonctionnement, vocabulaire
- un détail
- redéfinition méthodes et dérivations successives
- parcours d'un ensemble hétérogène
- remarque (méthodes dynamiques)
- redéfinition d'un champ statique
- factorisation, réutilisation
- choix méthode statique ou dynamique

redéfinition de méthode et dérivations successives

- une méthode est héritée dans toutes ses descendantes, sauf celles qui la redéfinissent
- de même pour la redéfinition d'une méthode

redéfinition des méthodes dynamiques

- but, principe, fonctionnement, vocabulaire
- un détail
- redéfinition méthodes et dérivations successives
- parcours d'un ensemble hétérogène
- remarque (méthodes dynamiques)
- redéfinition d'un champ statique
- factorisation, réutilisation
- choix méthode statique ou dynamique

parcours d'un ensemble hétérogène

- Considérons un tableau qui contient des Rationnels et RationnelColorés
- déclaration, remplissage, parcours
- Ex2.java : code de principe
- Démonstration
- Déroulement mémoire : exercice
- Code complet : Ex3.java

redéfinition des méthodes dynamiques

- but, principe, fonctionnement, vocabulaire
- un détail
- redéfinition méthodes et dérivations successives
- parcours d'un ensemble hétérogène
- remarque (méthodes dynamiques)
- redéfinition d'un champ statique
- factorisation, réutilisation
- choix méthode statique ou dynamique

méthodes statiques : remarques

- rappel : une méthode statique peut prendre en paramètre un objet d'une sous-classe de son paramètre prévu
- une méthode statique peut être en apparence « redéfinie », mais ne marche pas (voir exemple ci-dessous)
- rappel : jamais de liaison dynamique, car par définition le choix du code est fixé lors de la compilation (statique)
  - En particulier : le nom de la classe apparaît dans le code ; ex : Rationnel.afficher(r)

exemple

- Redéfinition de méthode statique
- Comportement bizarre, mais intellectuellement satisfaisant
  - le compilateur refuse d'appliquer la redéfinition et prend le code de la classe de la variable
- Ex4.java

redéfinition des méthodes dynamiques

- but, principe, fonctionnement, vocabulaire
- un détail
- redéfinition méthodes et dérivations successives
- parcours d'un ensemble hétérogène
- remarque (méthodes dynamiques)
- redéfinition d'un champ statique
- factorisation, réutilisation
- choix méthode statique ou dynamique

en résumé

- Rationnel est la classe de base
- RationnelColoré est la classe dérivée de Rationnel :
  - elle hérite de Rationnel tous ses membres (méthodes dynamiques et champs dynamiques)
  - elle en redéfinit certains (méthodes et/ou champs)
  - elle en possède en propre
- intuition : un rationnel coloré est un « cas particulier » de rationnel, avec des spécificités au niveau de la structure et du comportement

Redéfinition d'un champ dynamique

- Un champ dynamique peut être redéfini dans une sous-classe
- Sa classe doit être une sous-classe de celui de la super-classe

exemple

- Une personne a un nom et un nombre fétiche, qui est un rationnel
- Un peintre a un nom, style de peinture, et un nombre fétiche qui est un rationnel coloré
- Ex5.java

redéfinition des méthodes dynamiques

- but, principe, fonctionnement, vocabulaire
- un détail
- redéfinition méthodes et dérivations successives
- parcours d'un ensemble hétérogène
- remarque (méthodes dynamiques)
- redéfinition d'un champ statique
- factorisation, réutilisation
- choix méthode statique ou dynamique

Factorisation, réutilisation

- Revenons aux rationnels et rationnels colorés (sans les personnes)
- on a factorisé dans les membres de Rationnel tout ce qui était commun aux deux classes
  - (rem : en général seul le nom est commun, ex : afficher)
- Réutilisation : si Rationnel existait avant RationnelColoré
  - il est inutile de créer, coder et vérifier pour RationnelColoré num, dén, modifer, afficher
  - on les réutilise en créant RationnelColoré comme dérivée de Rationnel

- Important : on a factorisé aussi le rangement et son parcours, puisqu'on n'a utilisé qu'un seul tableau, hétérogène

absence de duplication

- C'est un critère fondamental de qualité de code
- C'est la conséquence de la factorisation
- regardons le code de notre application (Ex3.java) : aucune duplication de code ni de structure
- grâce à :
  - héritage
  - redéfinition et liaison dynamique (ne servent à rien l'un sans l'autre)
  - super

- se verra encore mieux en TD, où on aura programmé auparavant sans héritage ni redéfinition
- Ce n'était pas le cas dans les exemples et TD précédents : exercice

pertinence de la conjonction héritage et redéfinition

- Redisons-le d'une troisième manière
- exercice :
  - coder la gestion des rationnel et rationnels colorés ci-dessus sans ces concepts
  - Comparer aux exemples des cours précédents

- on s'aperçoit des redondances : champs, code, rangement : c'est mal
  - ex : ajout d'un champ ou modification format d'affichage
    - il faut modifier en plusieurs endroits
    - donc risque d'oubli ou erreur : incohérence
- on ne peut pas factoriser
- exemple : TD aujourd'hui mieux que TD précédent, et encore mieux que ceux d'avant

statique et dynamique en Java : résumé

- rappel vocabulaire :
  - statique : connu (resp. existe) lors de la compilation
  - dynamique :
    - inconnu (resp. n'existe pas) lors de la compilation
    - connu pendant l'exécution
- champs d'un objet
  - dynamique :
    - n'existera que si objet créé lors exécution
    - Existera pour chaque objet créé
  - statique :
    - un exemplaire unique dans la classe
    - connu lors compilation

- méthodes
  - statique :
    - fournit fonctions et procédures usuelles
    - code de f pour un appel f(x) : connu lors compilation
  - dynamique
    - permet redéfinition (héritage, surcharge, laison dynamique)
    - code de f pour un appel x.f() :
      - inconnu lors compilation
      - dépend classe effetcive objet contenu dans x lors exécution
  - remarque : on peut avoir dans même classe fonctions statiques et fonctions dynamiques ; elles peuvent s'appeler mutuellement, etc.

redéfinition des méthodes dynamiques

- but, principe, fonctionnement, vocabulaire
- un détail
- redéfinition méthodes et dérivations successives
- parcours d'un ensemble hétérogène
- remarque (méthodes dynamiques)
- redéfinition d'un champ statique
- factorisation, réutilisation
- choix méthode statique ou dynamique

rappel : choix entre méthode statique et dynamique

- critère 1 : il n'y a pas d'objet en jeu : statique (« obligatoire ») ; ex : calculer la factorielle
- critère 1bis : méthode dynamique fait traitements locaux à un objet (ex : déplace, affiche) ; donc pas sur tableau
- critère 2 : vu ensuite

choix entre méthode dynamique et méthode statique (suite et fin)

- critère 2 :
  - s'il y a en jeu de l'héritage et de la redéfinition (et donc de la liaison dynamique) : dynamique (obligatoire)
  - sinon il y a le choix
    - certains préfère le « style dynamique » même quand ce n'est pas nécessaire
    - CCI-LO : statique obligatoire

Remarque : tableau et objet

- Rappel : « un tableau est un objet » (référence + espace)
- Mais :
  - Champs même type, même nom, indicés
  - classe tableau :
    - dérive de Object
    - pas dérivable
  - pas de méthodes dynamiques

redéfinition des méthodes dynamiques

- but, principe, fonctionnement, vocabulaire
- un détail
- redéfinition méthodes et dérivations successives
- parcours d'un ensemble hétérogène
- remarque (méthodes dynamiques)
- redéfinition d'un champ statique
- factorisation, réutilisation
- choix méthode statique ou dynamique

redéfinition des méthodes dynamiques

- Delannoy chap. 8.4, 8.5 (?)