Langages objets

Méthodes dynamiques
(bloc 9)

M2 Pro CCI, Informatique
Emmanuel Waller, LRI, Orsay

résumé des épisodes précédents

- découverte Java, prise en main environnement
- types primitifs, opérateurs et expressions, instructions de contrôle, débugage
- Tableaux, fonctions, objets (sans et avec fonctions)
- Imbrication des constructeurs tableau et objet

méthodes dynamiques

- but
- principe
- Qu'est-ce qui est différent ?
- Fonctionnement, vocabulaire
- déroulement mémoire
- Cas de deux paramètres symétriques
- Méthodologie : écriture, choix statique/dynamique
- notion de surcharge

Java : sans programmation objet ?

- jusqu'à présent dans le module : pas fait du tout de programmation objet
  - types, opérateurs et expressions, contrôle, tableaux : comme C
  - fonctions : comme C avec syntaxe « bibliothèque » Personne.afficher(p), comme Math.cos(x)
  - objet = struct + pointeur : comme C
  - champs de classes : variables globales
- on va voir le concept objet : méthodes dynamiques

but

- en Java : 2 sortes de fonctions :
  - statiques : déjà vu
  - dynamiques
- certains concepts objet (héritage, redéfinition, polymorphisme, etc., ainsi que des bibliothèques) utilisent les fonctions dynamiques

principe

- aujourd'hui :
  - une méthode dynamique :
    - est une fonction
    - comme méthode statique, mais avec autre syntaxe + subtilités
  - que peut-on faire de plus avec ? rien
- différences et utilisation puissante : dès prochain cours

méthodes dynamiques

- but
- principe
- Qu'est-ce qui est différent ?
- Fonctionnement, vocabulaire
- déroulement mémoire
- Cas de deux paramètres symétriques
- Méthodologie : écriture, choix statique/dynamique
- notion de surcharge

exemple

- Rationnels : afficher
- rappel : avec fonction statique (cours objets et fonctions : Ex4.java)
- avec fonction dynamique : Ex1.java
- démonstrations

qu'est-ce qui est différent ?

- Syntaxe :
  - Déclaration : signature et corps (Ex1.java)
  - Appel (Ex1.java)
- Une fonction dynamique s'appelle obligatoirement sur un objet de sa classe
  - Éventuellement avec d'autres paramètres quelconques
  - Cet objet « paramètre principal » s'appelle le *receveur de la méthode dynamique*
    - on dit parfois que cet objet « reçoit un message » (un nom de fonction), d'où son nom de receveur

Dans l'appel : où est l'objet paramètre ?

- c'est bien r (ex : pour l'appel r.afficher())
- c'est bien un objet, de la classe Rationnel

Dans la déclaration : où est l'objet paramètre ?

- dans la déclaration de la fonction :
  - il faut une variable du type classe correspondant
  - pour manipuler l'objet paramètre : accès aux champs, calculs, appels d'autres fonctions dessus, stockage dans tableau, etc.
- une telle variable existe automatiquement sans être déclarée : c'est *this*
- this contient donc l'adresse de l'objet receveur

- ex :
  - this.num est le champ num de r (de même pour den)
  - lesRationnels[0] = this;
- rem : this est spéciale :
  - ne peut pas apparaître à gauche d'une affectation
  - ex : this = new Rationnel(4, 5) : n'a pas de sens
- rem : attention, on peut omettre this :
  - ex : num signifie this.num ; h() signifie this.h()
  - CCI : ne jamais le faire
- rem : exactement comme dans constructeur

- Bref, tout est exactement pareil que dans une fonction statique
  - sauf que this déclaré implicitement au lieu de explicitement
  - Revoir Ex1.java

fonctionnement

- Passage et modification des paramètres (y compris receveur), allocation dynamique, etc. : exactement comme fonction statique

vocabulaire

- Rappel : receveur
- on appelle *membres d'une classe* l'ensemble de ses champs et méthodes dynamiques

exemple

- Tableau de rationnels : afficher, modifier : Ex2.java

méthodes dynamiques

- but
- principe
- Qu'est-ce qui est différent ?
- Fonctionnement, vocabulaire
- déroulement mémoire
- Cas de deux paramètres symétriques
- Méthodologie : écriture, choix statique/dynamique
- notion de surcharge

déroulement mémoire

- exactement comme fonction statique
- dans le modèle mémoire, on ajoute la case this dans les variables locales (exactement comme constructeur)
- exemple déroulement mémoire complet : Ex2.java

cases

méthodes dynamiques

- but
- principe
- Qu'est-ce qui est différent ?
- Fonctionnement, vocabulaire
- déroulement mémoire
- Cas de deux paramètres symétriques
- Méthodologie : écriture, choix statique/dynamique
- notion de surcharge

exemple

- indiquer si deux rationnels sont identiques (mêmes nulmérateur et dénominateur) : Ex3.java
- rem :
  - 2 paramètres
  - syntaxe fonctions dynamiques :
    - adaptée à 1 receveur
    - tortueuse pour 2, mais fonctionne parfaitement
    - la symétrie en a et b n'est pas explicite dans a.coincide(b) : une des lacunes des méthodes dynamiques : tant pis
    - vocabulaire : dans a.coincide(b) : a est le receveur, et b un paramètre usuel

méthodes dynamiques

- but
- principe
- Qu'est-ce qui est différent ?
- Fonctionnement, vocabulaire
- déroulement mémoire
- Cas de deux paramètres symétriques
- Méthodologie : écriture, choix statique/dynamique
- notion de surcharge

méthodologie : comment écrire une méthode dynamique ?

- écrire fonction statique
- retirer static
- retirer paramètre principal (ex : p) de l'en-tête
- remplacer dans code p par this
- appel de cette fonction : suivre syntaxe
- Relire Ex1.java

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 : afficher, modifier) ; donc pas sur tableau
- critère 2 : vu ensuite

remarques

- appel méthode :
  - dynamique : sur un objet : o.f()
  - statique : sur un nom de classe : Client.f(o)
- on peut appeler :
  - une méthode dynamique dans le code d'une méthode dynamique
  - méth. dynamique dans méth. statique
  - méth. statique dans méth. dynamique
  - méth. statique dans méth. statique : déjà vu

méthodes dynamiques

- but
- principe
- Qu'est-ce qui est différent ?
- Fonctionnement, vocabulaire
- déroulement mémoire
- Cas de deux paramètres symétriques
- Méthodologie : écriture, choix statique/dynamique
- notion de surcharge

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();
```

comment choisit Java ?

- résolution de la surcharge
- +, méthodes statiques : compilateur devine à partir du contexte

- méthodes dynamiques : c.affiche() : à l'exécution la JVM demande à l'objet sa classe, et elle exécute le code de cette classe (détails vus ultérieurement) ; intérêt : ex :
  - Un tableau contient des objets de différentes classes (comptes simples, comptes négociés)
  - Unique code de traitement : for (i...) t[i].affiche()
    - Impossible choisir code affiche à compilation
    - Choix lors exécution (dynamique : d'où nom méthodes)

méthodes dynamiques : récapitulatif (6)

- but
- principe
- Qu'est-ce qui est différent ?
- Fonctionnement, vocabulaire
- déroulement mémoire
- Cas de deux paramètres symétriques
- Méthodologie : écriture, choix statique/dynamique
- notion de surcharge

- remarque : le chapitre 6 du Delannoy contient tout ce qu'on a vu jusqu'à maintenant sur les objets, y compris manipulations mémoire, fonctions, etc. (et détails hors programme)
  - ne pas lire (hors programme) : surdéfinition (6.8), récursivité (6.10), classes internes (6.12), paquetages (6.13)