Compléments de la POO
Cours 2

2021-2022 Université de Paris – Campus Grands Moulins
Licence 3 d'Informatique
Eugène Asarin

Merci à Aldric Degorre pour ses supports de cours!!!

Rappel ce qu'on a déjà appris

- Cours
  - Généralités sur la POO et Java
  - Rappels sur Java
  - Rappels sur le style
  - Instanciation des objets
    - Constructeurs
    - Fabriques statiques
    - Builders
  - Design patterns pour l'instanciation
    - Constructeur en téléscope
    - Java Beans
    - Builder
- TD (ce qu'on a vu en plus)
  - Encore plus de rappels
  - Première vue d' aliasing
  - Mutable/immuable
  - Réfléxion (facultatif)

Un peu de ~~théorie~~ (vocabulaire)

Ce qu’on cherche à faire

- Faire des programmes/bibliothèques qui satisfont certaines specification
- Essentiel : faire des classes satisfaisant des contrats….
- … et ceci dans divers scénarios d’utilisation
  - On les instancie, on les manipule, on les étend, on modifie les bibliothèques utilisées
  - On fait des choses stupides (ou méchantes)
  - Et ils doivent toujours respecter le contrats
- Exemples de « bouts de contrat » (il faut préciser les hypothèses)
  - Carre : les méthodes getLongueur() et getLargeur() renvoient le même résultat
  - Planetes : il y en a 9 et aucune autre
  - GenFib : appels successifs de next() donnent la suite de Fibonacci

2 cas de figure

- API/classes publiques : il faut résister à toutes les bêtises/attaques et/ou bien rédiger la spécification/contrat. +++de défense
- Votre implémentation dans les classes non-publiques. Il faut se protéger un peu de ses propres bugs , mais on peut être moins défensifs (en respectant des règles d'utilisation bien établies). + de performance etc

- Trade-of sûreté-performance
- Concrètement : dans 5 minutes

Parlons de types

- ensemble d'éléments représentant des données de forme similaire, traitables de la même façon par un même programme.
- (Ensemble+opérations) très populaire en math : Anneau (Z,0,1,+,*)
- Concrètement la représentation machine et/ou opérations disponibles sont essentielles. Pensez au types Comparable, List, HashSet…
- Divers langages – divers systèmes de types

Et des sous-types

- Le type *A* est sous-type de *B*(*A*<: *B)* si toute entité de type *A* est aussi de type *B*
  - (autrement dit:) «peut remplacer» une entité de type *B*.
  - Faible : A sous-ensemble de B
  - Structurelle :toute instance de *A* sait traiter les messages qu'une instance de *B* sait traiter.
  - **Idéale :** 1 (LSP). Un sous-type doit respecter tous les contrats du supertype.

Types et sous-types en Java

- Type primitifs et types référence
- Chaque var/expr a un type statique (à la compilation)
- Et un type dynamique à l'exécution (classe d'objet référencé)
- (Type dynamique doit *ê*tre sous-type du statique)

- Héritage et implémentation – sous-typage principal dans Java

Encapsulation

Éléments de programmation défensive

Encapsulation – principe (rappel)

- On cache à l'intérieur de la classe tous les détails d'implémentation. On décide quelles méthodes/attributs faut-il donner aux utilisateurs.

Rappel visibilité (hors modules)

- Pour les classes/interfaces etc
  - public class Machin
  - class Bidul
- Ça signifie
  - public : vous le donnez à l'humanité, votre API
  - … : vous ne le donnez à personne, votre implémentation

- public int age
- String genre
- protected double poids
- private Point centre

Attributs publiques de classe publique => danger (rappel)

- L'utilisateur (qui instancie) pourra modifier librement les attributs et « tout casser »
- le contrat ne sera plus respecté! Voir exemple FibGenB
- Solution: faire les attributs privés, fournir des getX et setX si nécessaire, en vérifiant qu'on ne casse rien

Getters et setters et leurs avantages

- contrôle de validité
- Initialisation paresseuse: la valeur de la propriété n'est calculée que lors du premier accès (et non dès la construction de l'objet) ;
- consignation dans un journal, comptage/statistiques.
- observabilité: le setteur notifie les objets observateurs lors des modifications;
- vétoabilité : le setteur n'a d'effet que si aucun objet (dans une liste connue de "véto-eurs") ne s'oppose au changement ;

Aliasing – un danger plus subtil

- Plusieurs références sur le même objet
- Nous : x=new Student ("Jean","Durand",22);
- L'attaquant: y=x; puis y.age=-215; et cela modifie le contenu de x

- Difficile à détecter: voir exemple IntervalleB.java
- A des avantages (économie de mémoire, cohérence, performance)

Comment se protéger d’ aliasing

- Attributs primitifs, immuables (par nature ou finals), inaccessibles, sont hors danger
- si on prend ou on donne un objet mutable on doit faire une copie défensive (profonde, avec le contenu). Voir exemple IntervalleG

Classes immuables (hors danger ou presque)

- L’état d’un objet immuable ne peut pas être modifié
- Exemples: String, Double, BigInteger
- 5 règles
  - Pas de méthodes pour modifier l’état
  - Tout attribut final
  - Tout attribut privé
  - Ne pas prendre/donner références aux objets mutables (faire copie défensive)
  - Interdire les sous-classes (faire final , ou faire constructeurs privés)

Classes finies = enum (encore mieux)

- Une vraie classe avec un nombre fini d'instances (souvent immuables), tous accessibles via constantes.
- Très bien pour les cartes, jours de semaines, etc…
- Il y a des bonne classes pour les manipuler à la EnumSet
- Voir exemple Planets.java

Héritage et ses problèmes

Héritage: idéal et réalité

- Idéal : héritage +> sous-typage
- Carre sous-type de Rectangle?
  - Réponse courte : mutable non, immuable oui
- Mais aussi l’ héritage crée de la fragilité, aliasing etc…
- On peut l’interdire ou l’exiger (classes final, abstract)

Dangers de l’héritage

- On en reparlera

Comment limiter l’héritage

- Final?
- Sealed?
- Private constructor?

Remplacer l’héritage par composition

- Version simple
- Patrons Adapter, Decorator, Delegation
- Voir TD ou cours prochain