Compléments de la POO
Cours 3

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

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

Rappel dernières séances

- Cours
  - Contrats et autres spec
  - Types et sous-types
  - Programmation défensive
    - Problème général
    - Problème d’alyasing
    - Solutions: encapsulation, copie défensive, utilisation des immuables et énums, interdiction d’´héritage
  - Héritage et co
- TD (ce qu’on a vu en plus)
  - Encore plus de rappels
  - Limites de l’héritage
  - Patron décorateur

Héritage et ses problèmes

Encore un peu là-dessus

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 et limites de l’héritage (voir exemple)

- Aliasing bien sûr
- Parfois ça boucle ou ça fait n’importe quoi
- Parfois ça casse le parent
- Parfois ça casse l’héritier

- Héritage multiple interdit

Règle de base

Quand on veut des héritiers on s'y prépare (contrats bien documentés, classe bien encapsulée, etc)

Sinon on se protège

- Final?
- Sealed?
- Private constructor?

Remplacer l'héritage par composition

- Avantage : plus de flexibilité, plus de sûreté
- Inconvénient : parfois plus lourd…
- Comment faire:
  - Patron Decorator : voir TD4, vous connaissez aussi:
    pr=**new PushbackReader**(**new BufferedReader**(**new FileReader**(**new File**("test.txt"))));
  - Patron Adapter, Delegation , renseignez-vous
- Conseils
  - sous-typage simple, peu d'options => héritez;
  - sous-typage multiple ou avec des options, ou pas vraiment sous-typage => composez
  - en tout cas les interfaces sont très utiles

Java Fonctionnel

POO+programmation fonctionnelle – ça existe et c’est utile

Vous l’avez vu un tout petit peu…

Pourquoi et comment

- Paradigme fonctionnel est aussi très ancien (LISP, ML, Haskell,…)
- Aujourd'hui fait partie des nombreux langages
- En Java existe depuis Java 8:
  - Une autre sorte de modularité/réutilisation de code
  - Allège le code, p.ex. des IG, des programmes multithread
  - Essentiel pour les API très importants Stream+Collectors (pour traitement massif de données, CompletableFuture (programmation concurrente et asynchrone sans risque)
- Et en plus c'est ~~amusant~~ intellectuellement rafraichissant

Fonctions de première classe & fonctions d'ordre supérieur

- On voudrait écrire par exemple:

```
void repeter5(??? f){//fonction paramètre???
for (int i=0;i<5;i++)
        f()}
```

Et puis l'appeler sur

```
void g(){System.out.println("la")}
```

Comme ceci

```
repeter5(g)
```

Mais comment???

- Ça se fait en C (avec les ptrs):

```
void repeter5(void (*f)()){//pointeur vers fonction
        for (int i=0;i<5;i++)
                f();
}
void g(){ puts("la");}

int main(){ repeter5(g); }
```

Il nous faut FPC & FOS

- Des FPC (fonctions de première classe – *functions as first class citizens*) des valeurs comme des autres.
  - peuvent être affectée à une variable,
  - être paramètre d'une fonction, ou bien sa valeur de retour
- On donnera une FPC pour gérer un bouton , pour lancer dans un Thread, pour appliquer 1000 fois (ou 0), pour trouver ses racines…
- Et on aura des FOS (fonctions d'ordre supérieur) qui utilisent tels paramètres, ou renvoient des FPC comme résultats
- Par ailleurs les structures de contrôle while, if…else etc ressemblent aux FOS, voir exemple repeat5 plus haut

Parenthèse: FPC existent en maths, on aimerait programmer

- « Types » Espaces
  $C^2[0,1]$, $\mathrm{L}_2(\mathrm{R})$
- « FOS »

Soit $M = \{x_1, \dots, x_n\} \subset N$,
$f: N \to R$, alors on aura $\mathrm{I}MAGE(f, M) \subset R$
définie comme $\{f(x_1), \dots f(x_n)\}$

- Algos numériques:
  Soit $f: [0,1] \to R$ telle que $f(0) < 0, f(1) > 0$. Alors pour résoudre l'équation $f(x) = 0$ avec précision ε on fera:

```
dichotomy(f,epsilon)
        left=0; right=1
        while (right-left>epsilon)
                middle=(left+right)/2
                if(f(middle)>0)
                        right=middle
                else
                        left=middle
        return (left+right)/2
```

Requis pour FPC

Trois choses requises:

1. Système de types pour écrire les FPC
2. Représentation en mémoire des FPC
3. Syntaxe commode pour définir vos propres FPC

Réalisé en Java 8+

1. Interfaces fonctionnelles
2. Objets
3. Plusieurs possibilités

- instanciation « traditionnelle »
- λ-expressions
- ... dont références au méthodes

Type de FPC: interface fonctionnelle

- Toute interface avec unique méthode abstraite convient.
- Annotation conseillée

```
@FunctionalInterface
interface MaFPC{
String func(int x);
}
```

- Comparable, Comparator, Runnable,Callable, ActionListener aussi…
- Excellente bibliothèque de types java.util.functions

FPC en mémoire

- La FPC corresspond à une instance d'une classe implémentant une interface fonctionnelle (par exemple Runnable)
- La fonction implémente la seule méthode abstraite* (par ex. run())

* Il faut absolument connaitre son nom

Représentation syntaxique de f en FPC (ancienne/ennuyeuse) –voir Eclipse

Technique 1 (très lourde)

- Choisir (ou faire) une interface fonctionnelle
- Créer une classe qui l'implémente avec f réalisant la méthode abstraite.
- L'instancier

Technique 2 (un peu moins lourde)

- Choisir (ou faire) une interface fonctionnelle
- Instancier un objet de classe anonyme avec f réalisant la méthode abstraite.

Représentation nouvelle des FPC: lambda

- lambda abstraction
  - x -> x+1
  - () -> 3+math.random() //expression
  - (x,y) -> {z= Math.sin(x); return x+y+z} //liste d'instructions
- Références
  - Math::sin // équivalent à x -> Math.sin(x)
  - String::length //équivalent à String s ->s.length()
  - s::length //équivalent à ()->s.length()
  - Personne::new //équivalent à (nom,prenom)->new Personne (nom,prenom)

Exemples de code – FirstClass.java sur moodle

FOS

- repeter5
- ifThenElse
- image
- dichotomie

FPC tout style

- Objet de classe
- Objet de classe anonyme
- Lambda-expression (par lambda-abstraction)
- … par référence à une méthode

API disponibles : types pour vos FPC

- Pour le type ()->() utiliser Runnable la doc (cliquez)
- Pour plein d'autres java.util.function la doc (cliquez)
- Tout type existe en générique et en double/int etc
- Il y a des méthodes sympas, regardez…
- Les types les plus intéressants:

Interface de java.util.function	Type	Exemple de lambda-expression
UnaryOperator	A->A	x -> 10*x+3
Function	A->B	s -> s.length()+3
Predicate	A->{true,false}	x -> x>33
Supplier	()->A	Math::random
Consumer	A->()	x -> System.out.println(x)

- Il faut connaitre les noms de méthodes (apply, test, accept) pour programmer vos FOS
- Si besoin, faites vos propres types (génériques?) de FPC

Exemples jouets encore (SecondOrder.java sur moodle)

- Objet de ype Supplier
- Objet de type Predicate
- Notre propre Interface fonctionnelle: relation ternaire
- FOS ajouter: prend deux fonctions, renvoit la somme

- Les vrais exemples plus tard…

Remarques savantes

FPC: évaluation paresseuse

- transformées,composées avant d'être évaluées;
- évaluées plus tard,0,1, 533 fois, en fonction de critères programmables ;
- Exécutées dans un autre contexte(p.ex. autre *thread*).

Comment Java type les lambdas?

- À la compilation
- En fonction du contexte
- Type de x->x ou de x->() ou même x-> 2*x dépend énormément du contexte!