Compléments de la POO
Cours 5

2021-2022 Université de Paris – Campus Grands Moulins

Licence 3 d'Informatique

Eugène Asarin

Rappel dernières séances

- Cours
  - comparing
  - streams
- TD
  - streams

On corrige le 1er test

Question 1 - culture générale

- On dit que Java a le typage à deux phases. Expliquez

  Phase 1 : à la compilation typage statique des variables et des expressions
  Phase 2 : à l'exécution typage dynamique des objets

- Exemple (variable x est une List<String>, mais elle pointe sur des objets de types différents)

```
List<String> x=null;
x=new ArrayList <String> ();
x=new LinkedList <String> ();
x=List.of( "bon", "jour", "nuit");
int n=x.length();
```

- Je n'aime pas l'expression simpliste « type déclaré ». Parfois on ne déclare rien mais le compilateur se débrouille pour typer:

```
3+4.
var y=5;
x->x+1
```

Question 2 – fabrique

- Remplacez le constructeur public par une fabrique statique et donnez l'exemple de son utilisation

```
public class Point{
   private double x, y ;
   private
  ~~public~~ Point(double x, y ;){
      this.x=x ;
      this.y=y ;
 }
```

```
//codez la fabrique (dans la même classe)
public static Point newInstance (double x,y){ return new Point(x,y);}
}


// utilisez-la (dans un main)
Point p=Point.newInstance(3,6);
```

- *No comment.* Il fallait juste connaitre les fabriques statiques comme alternative aux constructeurs publics.

Question 3 – attaque par aliasing

- Contrat : "chaque jour il ferme **après** l'ouverture". Cochez 2 fragilités. Écrivez des attaques qui cassent le contrat.

☑
```
public class Magasin{
private String nom; //comment s'appelle-t-il
private int[7] ouverture, fermeture ; // heures d'ouverture et fermeture
public  Magasin(String n, int [] o, int []f){//constructeur
        if (o.length !=7 || f.length !=7)
            throw new IllegalArgumentException();
        for (int i=0; i<7; i++)
            if (o[i]>= f[i])
                throw new IllegalArgumentException();
         nom=n ; ouverture=o ; fermeture =f ;
 }
  public String getNom() {return nom;} // accesseurs
  public int[] getOuverture() {return ouverture ;}}
```
☑

```
//on instancie un magasin
String s= "Interfour";
int[] o={8,8,9,9,10,7,9};
int[] f={16,19,20,14,15,15,18};
Magasin shop= new Magasin(s,o,f);
//Attaque 1 – par constructeur
o[3]=23;

//Attaque 2  – par accesseur
int[] ouvre=shop.getOuverture() ;
ouvre[2]=22 ;
```

- Fausse attaque: ouverture[2] =20; impossible pour un champ privé
- Fausse attaque: o ={25,26,24,22,22,22,22}; crée un nouveau tableau, ne change pas le magasin
- Fausse attaque: o2 ={25,26,24,22,22,22,22}; Magasin shop= new Magasin(s,o2,f); lève une exception
- Fausse attaque: String s=shop.getNom(); s= "Marché"; crée une nouvelle chaine, ne change pas le magasin
- Fausse attaque: String s=shop.getNom(); s.setCharAt(2,'z'); impossible, String immuable
- Fausses attaques: par copie, avec heures négatives, etc
- Attaque possible avec héritage

Projet

- Sorry, je finaliserai pendant le week-end
- Faire en binôme (monôme toléré si vraiment besoin).
  - un matheux pour les nombres complexes+un bidouilleur pour l'IG?
  - pourquoi pas, mais chacun doit connaitre tout le projet.
- Date limite: Noël
- Le plagiat sera détecté et sévèrement puni.
- Sujet: faire un logiciel qui construit les fractales (en fichier et en IG)
- Fractales: https://fr.wikipedia.org/wiki/Fractale

Critères fonctionnels

- Remplir les fonctionnalités de base
  - Version ligne de commande + version IG
  - Générer les ensembles de Julia, pouvoir choisir un paramètre
  - Pouvoir choisir le rectangle affiché
- Remplir certaines fonctionnalités avancées
  - Pour Julia, pouvoir choisir un polynôme
  - Pouvoir choisir le « code couleur »
  - Générer l'ensemble de Mandelbrot, autres fractales.
  - Zoom et déplacement confortables

Critères qualité de code

- Respect de règles de style, lisibilité, commentaires
- Bonne architecture objet, éventuellement MVC etc…
- POO, encapsulation, héritage, polymorphisme
- Bonne utilisation des API (on peut utiliser toutes les bibliothèques sauf spécifiques aux fractales)
- Gestion des exceptions, robustesse du code

Critère nouvelles technique
si utile, par exemple

- Lambda pour tous les boutons
- Builder pour configurer votre fractale ou affichage
- FPC comme attribut de Julia
- Thread séparé pour les calculs
- Essayer un thread léger par pixel
- Enum, sealed class, généricité etc... Si utile
- Streams si utile (mais je ne vois pas…)

Ne pas perdre votre temps sur

- Une interface graphique trop belle ou trop poussée
- la programmation défensive (copies défensives etc…)

Concurrence en Java

Très petite introduction

Connaissez-vous la concurrence?

- Sans doute vu un peu dans systèmes, BD, C, Java (surtout IG)
- Un sujet vaste, important et profond
- Indispensable mais plein de dangers
- Il y aura un cours spécifique en M1 : « théorie et pratique de la concurrence »
- Ici : juste quelques notions

Notion de la concurrence

- deux actions, instructions, travaux, tâches, processus, etc. sont concurrents si leurs exécutions sont indépendantes l'une de l'autre
- => deux actions concurrentes peuvent s'exécuter simultanément, si la plateforme d'exécution le permet.
- Programme concurrent: avec certaines portions de code (signalées) indépendantes => la plateforme l'exploite (p.ex. en exécutant simultanément).

Pourquoi, où et quand?

- Naturelle et nécessaire dans des situations variées en programmation
  - Serveur web : un même serveur doit pouvoir servir de nombreux clients indépendamment les uns des autres sans les faire attendre.
  - Interface graphique : gérer les actions de l'utilisateur ET exécuter d'éventuelles tâches de fond
  - Programme qui doit réagir immédiatement à des événements de causes et origines variées et indépendantes.
- Possible et utile pour des algorithmes décomposables en étapes indépendantes (concurrence, parallélisme, accélération)

Parallélisme

- Deux travaux s'exécutent en parallèle ⇔ s’exécutent en même temps
- Degré de parallélisme = nombre de travaux simultanément exécutables.
- CPU modernes: plusieurs cœurs, souvent 2 travaux par cœur (hyperthreading)
- Et donc on a souvent le degré de parallélisme= 8 ou 16 ou…

Point de vue Système

- On a des processus qui tournent indépendamment avec espace mémoire séparé (+un peu de communication)
- Un processus peut avoir plusieurs threads (même tas, une pile pour chacun)
- Le système exécute les threads en parallèle ou en les ordonnonçant
- Typiquement milliers de threads sur 8-16 cœurs…

Processus et threads en Java

- Typiquement un programme Java = un processus
- Un programme Java peut avoir plusieurs threads
  - Le thread principal de main
  - Le thread de l'IG
  - Et les threads que le programmeur lance (avec les API bas ou haut niveau)
  - On n'y pense pas mais il y a d'autres threads (garbage collector,…)
- Important: Chaque thread a sa pile, le tas est partagé

Créer un thread en Java à partir d'un Runnable

```
public class HelloRunnable implements Runnable {
    public void run() { System.out.println("Hello from a thread!");
}
// et on le lance avec
Thread t=new Thread(new HelloRunnable());
t.start();

//Même chose avec lambda est plus légère
new Thread(()-> System.out.println("Hello from a thread lambda!")).start();
```

Autre style, étendre Thread

```
public class HelloThread extends Thread {
        public void run() { System.out.println("Hello from a thread!");
        }

        public static void main(String args[]) {
        (new HelloThread()).start();
        }
}
```

Quelques méthodes (avec un Thread t)

- Thread.sleep(5000) – on dort 5s
- t.join() – on attend qu'il termine; ou t.join(5000) – on l'attend 5s max

- Mais on peut aussi interrompre avec t.interrupt()
- Dans ce cas il est poli de gérer InterruptedException

- Allons vers Eclipse pour expérimenter!

Les threads communiquent par

- Mémoire partagée (le tas est commun) – utile mais risquée
  - On verra les risques et comment les éviter
- Par passage de message – mais il faut l'organiser…
- On revient aux exemples

Anomalies (data races)

- Si au départ x=0, puis deux threads exécutent en même temps, par exemple x++, c'est-à-dire:

  lire x

  calculer x+1

  ecrire la nouvelle valeur de x

Il se peut que T1 lit 0, T2 lit 0, ils calculent 1, ils écrivent 1. Une incrémentation est perdue!!!

- Solution classique – utiliser un verrou. Avant d'accéder à x on prend le verrou, après l'accès on le rend. Ainsi un seul thread peut accéder à x (exclusion mutuelle)

Pour éviter des anomalies (data races)

- Protéger les accès aux données partagées par les verrous (moniteurs)
- Chaque objet java a un verrou intrinsèque
- Pour verrouiller un bloc:

synchronize(obj){faire qch}

- Pour verrouiller une méthode (avec un verrou intrinsèque de this)

synchronized fff(){}

- Il faut le faire très localement (ça annule les avantages de la concurrence, et crée des risques de blocage), seulement quand nécessaire
- Il y a aussi des verrous plus spécifiques, voir TD

Risque de blocage

- Exemple de deadlock – les deux processus ont pris chacun un verrou et ils s’attendent mutuellement (voir exemple)
- Mais existent des livelocks et autres famines, divers scenarios quand un programme concurrent ne peut pas avancer correctement