Compléments de la POO
Cours 6

2021-2022 Université de Paris – Campus Grands Moulins

Licence 3 d'Informatique

Eugène Asarin

On corrige le second test

il était plus facile que le premier

Question 1 - culture générale

- Dans quel aspect les langages Simula et Smalltalk sont-ils

  Ce sont les tous premiers langages orientés objet

- Simula 67 (c'était la 2ème version) a introduit les classes en 1967
- Smalltalk (1969-1972-1980!) « Tout est objet » + IDE graphique
- Réponses fausses: en programmation fonctionnelle.
- Réponse incomplète : en POO
- Réponse bizarre : réflexif et dynamiquement typé.

Corrigendum

- Simula est un langage de programmation connu par sa bibliothèque de simulation des systèmes aux évènements discrets (et non des systèmes physiques)

Question 2 – révision

- Quelles sont les deux différences principales entre les interfaces List et Set ?

1. Set n'a pas de doublons
2. Les éléments d'une List sont ordonnés, chacun a son indice

- Je n'aime pas quand vous parlez de LinkedList, la question était sur les interfaces.
- Je n'aime pas la réponse: par complexité, l'ajout se fait en O(1) pour List et en O(n) pour Set. En réalité ça dépend de la classe, c'est presque O(1) pour HashSet, O(log n) pour TreeSet etc…

Question 3 – expliquer une classe

```
class Enigme{
   private static Enigme instance=null;
   private String s=null;
   private Enigme( ){
    … télécharger un article de Wikipedia
dans s
     instance=this ;
  }
   public static Enigme getInstance(){
        if (instance !=null)
             return instance;
        return new Enigme();
   }
   public String getContent(){return s;}
}
```

*C'est une classe…*singleton

Son unique instance peut être obtenue grâce à la fabrique statique getInstance.

A l'instanciation un article est téléchargé, chaque appel à getContent renvoie cet article.

Ce n'était pas demandé mais on l'utilise comme ceci

```
Enigme x=Enigme.getInstance() ;
String article=x.getContent();
```

2 niveaux de réponse: comment est-il fait? que fait-il concrètement?

Question 4 – Qu'affichera le code suivant ?

```
Function<String,Integer> f=String::length;
IntPredicate p= (n -> n>8);
System.out.println(p.test(f.apply("bonjour"));
```

false

- C'est du fonctionnel (lambda-expressions)
- f.apply(s)= s.length()
- p.test(n)= (n>8) (un Booléen)

Question 5 – Programmez une fonction (d'ordre supérieur) qui donne le produit de deux fonctions entières

Rappel : la méthode abstraite de la classe IntUnaryOperator est int applyAsInt(int operand)

- Solution 1 (avec lambda – ma préférée)

```
public static IntUnaryOperator product(IntUnaryOperator f,
IntUnaryOperator g){
      return x -> f.applyAsInt(x)*g.applyAsInt(x);
 }
```

- Solution 2 (avec une classe anonyme – tout à fait correcte)

```
public static IntUnaryOperator product(IntUnaryOperator f, IntUnaryOperator g){
   IntUnaryOperator res= new IntUnaryOperator (){
     @Override
     public int applyAsInt(int x){ return f.applyAsInt(x)*g.applyAsInt(x);}
   };
  return res;
}
```

Projet

- Tout est en ligne (ou presque)
  - Sujet
  - Choix de binôme (tous inscrits? 90 binômes)
  - Remise (jusqu'au 23/12 à 23:59, mais faites-le avant les vacances!)
  - Forum questions-réponses (j'ai mis 2 de vos questions, n'hésitez pas!)
- Aujourd'hui on parlera du parallélisme
- Autres questions?

Concurrence en Java

Très petite introduction – suite et fin

Déjà vu

- Notions de concurrence et parallélisme
- Threads, comment les créer, lancer, endormir, attendre la fin, interrompre
- Mémoire partagée: très utile , mais danger de data race et résultats faux
- Verrous/moniteur/synchronized – comment les mettre pour éviter les erreurs de concurrence. Mais danger de blocage

Aujourd'hui – 2 études de cas

- Concurrence: Producteur génère des messages, les mets dans une boite, le consommateur les ramasse et les affiche. Version simplifiée de serveur de *chat* etc…
  - Solution 1: avec synchronisation et attente par wait
  - Solution 2: avec structure de données thread-safe BlockingQueue
- Parallélisme: Tableau des nombres premiers
  - ~~Solution 0: à la main avec les Threads~~
  - Solution 1: avec un Executor et un pool de Threads
  - Solution 2: avec ForkJoin
  - Solution 3 :avec parallel Streams

Voir exemples Java!!!

Leçons étude de cas 1

- On a écrit les gros Runnable sans lambda…
- Solution1
  - Déjà vu: mémoire partagée protégée par synchronized
  - Comment 2 threads s'attendent poliment avec wait-notify
- Solution 2
  - Avec une structure de données thread-safe on n'a plus rien à faire
  - Regardez attentivement BlockingQueue, ConcurrentMap et leurs implémentations

Leçons étude de cas 2

- Executor et autre WorkStealingPool permettent de lancer facilement autant de threads de calcul que l'ont veut, l'API gère le niveau de parllélisme
- ForkJoin est adapté aux algorithmes parallèles « diviser-pour-régner »
- Si vous savez programmer en Stream, ça vaut la peine d'essayer de paralléliser (en ajoutant une opération intermédiaire parallel()).
  Remarque: en aval il faut utiliser les collections thread-safe dans ce cas.