PROJET

ÉNONCÉ DES LIVRABLES

PREMIER LIVRABLE

SURVOL

GIT & BUILD

Le but du premier livrable est de préparer la maintenance du programme Java fourni:

1. Créer un repo GitHub privé pour votre équipe;
2. Mettre le code source fourni sous gestion de source.
   S'assurer que le remote « origin » point vers votre répo privé;
3. Utiliser un outil de build Java qui permet de:
   a) compiler le code source Java;
   b) exécuter les tests jUnit; et
   c) générer une archive JAR exécutable.

CODE FOURNI

- Le code source fourni est écrit en Java 11;
- Les tests unitaires utilise la version 5.3 de jUnit;
- Le code applicatif utilise la version 2.9 de la librairie Jackson;
- Les tests unitaires sont contenus dans la classe `minischeme.EvaluatorTests`. Les autres fichiers java correspondent au code applicatif;
- La classe principale est `minischeme.Application`. C'est cette classe qui doit être exécutée lorsqu'on exécute l'archive JAR sur la ligne de commande:
  ```
  $ java -jar minischeme.jar exemples/facto.json
  ```
- Vous pouvez réorganiser les fichiers et les répertoires comme bon vous semble.

ANALYSE DES OUTILS DE BUILD

Vous devez effectuer une courte analyse comparative de 2 outils de build pour Java. Les choix possibles sont: Maven, Gradle, Ant (avec Ivy) et Make.

Le rapport doit être écrit en format Markdown. Le nom du fichier doit être "`analyse-outils-build.md`" et doit être sous gestion de sources, à la racine du repo privé.

MARCHE À SUIVRE

1. Choisir 2 outils à comparer (p.e. « Maven vs Ant (avec Ivy) »;
2. Rédiger une courte description de chaque outil (environ 150 mots par outil)
3. Établir 3 critères d'évaluation;
4. Pour chaque critère:
    a) Rédiger une description du critère;
    b) Indiquer pourquoi ce critère est important;
    c) Indiquer lequel des deux outils est le « meilleur » selon ce critère et, surtout, *pourquoi* c'est le meilleur outil.
    d) (environ 350 mots par critère)

ETC

- Le rapport compte pour **5 points sur 50**;
- Le reste du premier livrable compte pour **5 points sur 50**;
- La version du rapport présente dans le repo le **21 février à la fin de la journée** sera corrigée. Seul le rapport sera corrigé à cette date. Le reste du premier livrable sera corrigé à la fin de la session.

APERÇU DU PROGRAMME

MINI-SCHEME

Le programme que vous devrez instrumentaliser et maintenir est un interpréteur pour un langage de programmation inspiré par les langages Scheme et LISP.

MINI-SCHEME

En mini-scheme, tout est une liste. Le premier élément d'une liste est le nom de la fonction à appeler. Les éléments suivants sont les arguments. Par exemple `["additionner", 2.0, 3.0]` en Java s'écrirait `additionner(2.0, 3.0);`.

Puisque nous sommes habitués aux notations « infixes » (i.e. avec l'opérateur au milieu), la notation « préfixe » peut sembler étrange aux premiers abords:

`(2 × (3 + 4))`

en notation préfixe devient:

`(× 2 (+ 3 4))`

et en mini-scheme devient:

`["*", 2.0, ["+", 3.0, 4.0]]`

4 FONCTIONS SPÉCIALES

- La fonction « `define` » déclare une variable avec sa valeur initiale;
- La fonction « `begin` » permet d'exécuter un bloc d'instructions. Elle exécute tous ses paramètres et retourne la valeur du dernier;
- La fonction « `if` » reçoit 3 paramètres, une valeur booléenne, l'instruction à exécuter si le test est true, et celle à exécuter si le test est false.
- La fonction « `lambda` » crée une fonction. Le premier paramètre est la liste de paramètres attendus, le second est le corps de la fonction.

MINI-SCHEME

Les 2 exemples de code source mini-scheme suivants sont là pour que vous vous familiarisiez avec les bases du langage. Un programme Java (presque) équivalent est donné en contre-partie.

CALCUL DE L'AIRE D'UN CERCLE


```lisp
["begin",

  ["define", "pi", 3.141592],

  ["define", "aire-cercle", ["lambda", ["r"],
    ["*", "pi", "r", "r"]]],

  ["aire-cercle", 10.0]]
```

CALCUL DE L'AIRE D'UN CERCLE


```java
public class Application {
  static double pi = 3.141592;

  static double aireCercle(double r) {
    return pi * r * r;
  }

  public static void main(String... args) {
    aireCercle(10.0);
  }
}
```

CALCUL DE LA FACTORIELLE

λ

```
["begin",

  ["define", "facto", ["lambda", ["n"],
    ["if", ["<", "n", 2.0],
      1.0,
      ["*", "n", ["facto", ["-", "n", 1.0]]]]]],

  ["facto", 5.0]]
```

CALCUL DE LA FACTORIELLE


```
public class Application {

  static double facto(double n) {
    if (n < 2.0) {
      return 1.0;
    } else {
      return n * facto(n - 1);
    }
  }

  public static void main(String... args) {
    facto(5.0);
  }
}
```

SECOND LIVRABLE

TESTS & ANALYSE

Détails à venir…

LIVRABLE FINAL

DÉPLOIE-MENT

Détails à venir…