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

SURVOL

TESTS & ANALYSE

Le but du second livrable est de faire des modifications au code source, de les tester avec une suite de tests exhaustifs, de calculer la couverture des tests et de faire une analyse statique du code source. Vous devrez aussi utiliser le mécanisme de pull-request de GitHub pour soumettre vos modifications.

NOUVELLES FONCTIONS

- Les opérateurs d'algèbre booléenne suivants doivent être ajoutés au langage mini-scheme: "`and`" et "`not`".
- L'opérateur "`and`" doit prendre en paramètre une liste d'expressions booléennes. Elle retourne `true` si toutes les expressions sont `true`.
- L'opérateur "`eq`" qui teste l'égalité d'une liste de valeurs numériques doit aussi être ajouté au langage;
- L'opérateur "`eq`" doit retourner `true` si tous les paramètres sont égaux et `false` autrement. Puisque nous traitons avec des Doubles, prévoyez une tolérance de comparaison adéquate.

NOUVELLES FONCTIONS

- Une version modifiée de la classe `Evaluator` est fournie avec ce livrable. La nouvelle version permet de créer des listes;
- Vous devez ajouter les fonctions suivantes: "`count`", qui retourne le nombre d'éléments de la liste; "`head`" qui retourne le premier élément de la liste; "`tail`" qui retourne une sous-liste contenant tous les éléments de la liste sauf le premier élément.

EXIGENCES SUPPLÉMENTAIRES

- Vous devez remettre un programme mini-scheme (JSON) qui montre comment construire les portes logiques suivantes avec uniquement "`and`" et "`not`": NAND, NOR, OR, et XOR. La page wikipedia sur le sujet pourrait vous être utile;
- Tout le code Java doit être entièrement testé avec des tests unitaires JUnit. La qualité des tests fait partie des critères d'évaluation;
- La couverture des tests doit être calculée dans le cadre du processus de build;
- Des outils d'analyse statique doivent être intégrés à l'outil de build. Des rapports de conformité doivent être automatiquement produit lors de la compilation.

EXIGENCES SUPPLÉMEN TAIRES

- Vous devez utiliser le processus de pull-requests de GitHub. Chaque nouvelle fonctionnalité doit être soumise pour approbation sous la forme d'un pull-request;
- Pour être accepté, un pull-request doit contenir le code source applicatif, les tests unitaires et tout le code doit répondre aux critères d'analyse statique choisis.

ANALYSE STATIQUE

Vous devez effectuer une courte analyse comparative de 2 outils d'analyse statique pour Java. Les choix possibles sont: PMD, SpotBugs, Checkstyle et Sonarqube.

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

MARCHE À SUIVRE

1. Choisir 2 outils à comparer;
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 second livrable compte pour **20 points sur 50**;
- La version du rapport présente dans le repo le **28 mars 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.

EXEMPLE DE CALCULS BOOLÉENS


```
["begin",

  ["define", "p", true],
  ["define", "q", false],

  ["and", "p", true, ["not", "q"], ["<", 1.0, 2.0]]]
```

EXEMPLE DE CALCULS BOOLÉENS


```
public class Application {

  public static void main(String... args) {
    final boolean p = true;
    final boolean q = false;

    System.out.println(
        p && true && (!q) && (1.0 < 2.0));
  }
}
```

CALCUL DE LA SOMME DES ÉLÉMENTS D'UNE LISTE

```
["begin",

  ["define", "do-sum", ["lambda", ["xs", "resultat"],
    ["if", ["<", ["count", "xs"], 1.0],
      "resultat",
      ["do-sum", ["tail", "xs"], ["+", "resultat", ["head", "xs"]]]]]],

  ["define", "sum", ["lambda", ["xs"],
    ["do-sum", "xs", 0.0]]],

  ["sum", ["list", 1.0, 2.0, 3.0, 4.0, 5.0]]]
```

CALCUL DE LA SOMME DES ÉLÉMENTS D'UNE LISTE

```java
public class Application {
  private static double do_sum(List<Double> xs, double resultat) {
    if (xs.size() < 1) {
      return resultat;
    } else {
      return do_sum(xs.subList(1, xs.size()), resultat + xs.get(0));
    }
  }

  private static double sum(List<Double> xs) {
    return do_sum(xs, 0.0);
  }

  public static void main(String... args) {
    System.out.println(sum(List.of(1.0, 2.0, 3.0, 4.0, 5.0)));
  }
}
```

LIVRABLE FINAL

SURVOL

INTÉGRATION

Le but du dernier livrable est d'intégrer des modules existants au système et de mettre en place un système d'intégration continue.

NOUVELLE SYNTAXE

- Un compilateur pour une nouvelle syntaxe de Minischeme a été développée. Le projet est disponible sur github: https://github.com/fxg42/minischeme-parser
- Le repo contient les informations et les tests nécessaires pour comprendre son utilisation.
- Les fichiers de configuration Gradle et Maven sont fournis.
- Vous devez remplacer le compilateur actuel (`com.fasterxml.jackson.databind.ObjectMapper`) par le compilateur de ce projet (`minischeme.parser.Parser`);
- Bien entendu, tous les tests existants et les exemples de code source minischeme devront être ajustés…

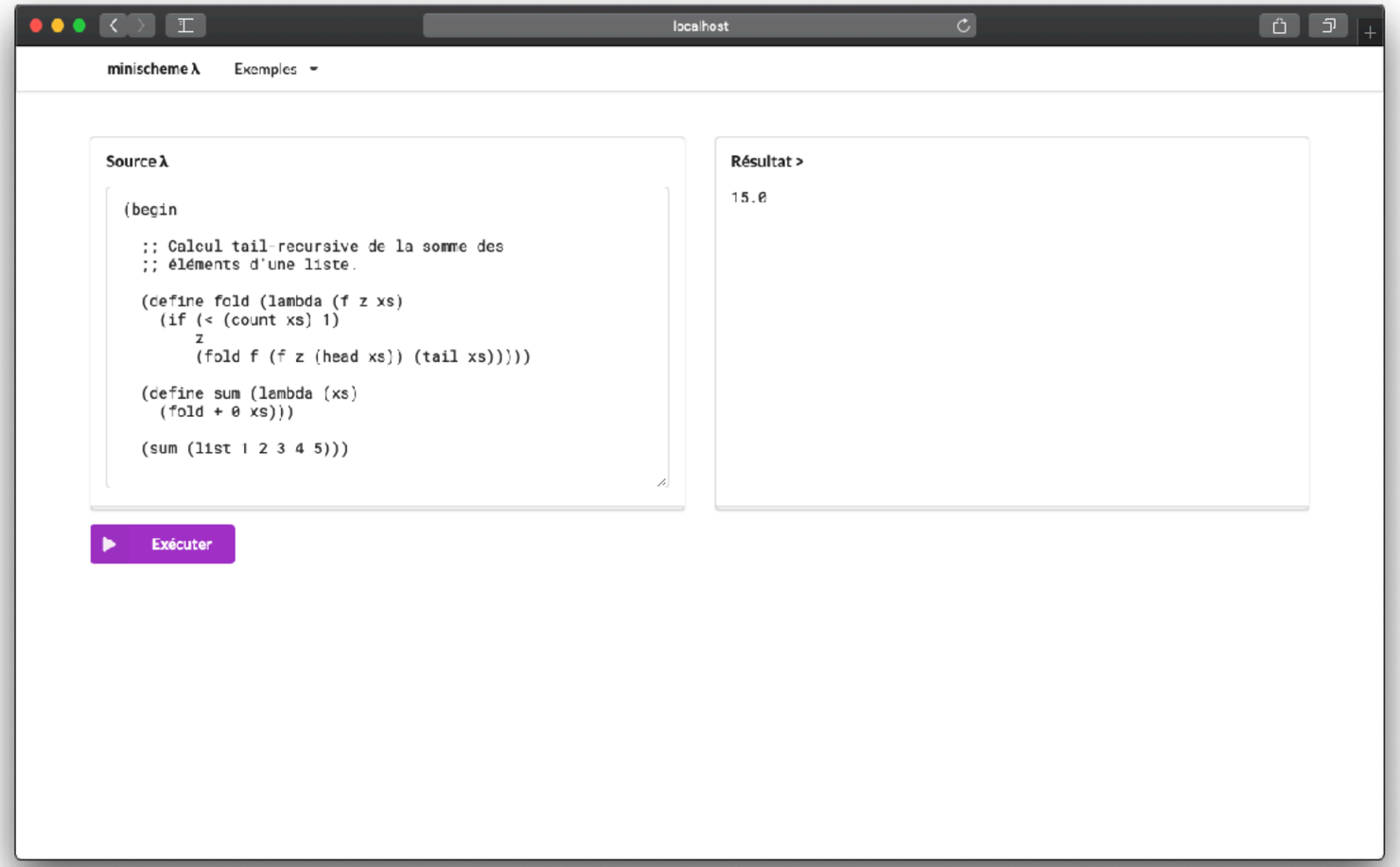
NOUVELLE SYNTAXE

- Les opérateurs et fonctions n'ont plus à être entourés par des "
- Les opérandes n'ont plus à être séparées par des `,`
- Les `[ ]` sont remplacés par des `( )`
- Les mots-clés `true` et `false` sont remplacés par `#t` et `#f` respectivement
- Il est possible d'écrire un nombre sans partie décimale c.-à-d. `10` au lieu de `10.0`
- Les commentaires débutent par `;;` et se terminent à la fin de ligne

NOUVELLE INTERFACE

- Une interface web a été développée pour minischeme. Le projet est disponible sur github: https://github.com/fxg42/minischeme-web
- Le repo contient les informations nécessaires pour comprendre son utilisation.
- Les fichiers de configuration Gradle et Maven sont fournis.
- Vous devez intégrer le projet en cours (avec la nouvelle syntaxe) dans l'interface web fournie. Notez que contrairement au code du compilateur, celui de l'application web devra être modifié.

NOUVELLE INTERFACE



INTÉGRATION

- Vous devez intégrer les 3 modules – l'évaluateur, le compilateur (*parser*) et l'application web – dans votre repo GitHub.
- Les modules « compilateur » et « évaluateur » doivent être intégrer ensemble dans le même fichier de build;
- Une dépendance vers ce module (évaluateur+compilateur) doit être ajoutée dans le module « web ».

CI

- Vous devez utiliser un serveur d'intégration continue (p.e. Jenkins ou autre) pour compiler et tester le code source du module « évaluateur + compilateur » automatiquement;
- L'intégration avec GitHub Classrooms n'étant pas parfaite, vous devrez indiquer au serveur Jenkins (le cas échéant) de surveiller votre repo local. (voir https://stackoverflow.com/a/12185738 pour une explication). Jenkins doit donc être installé en local (sur votre poste) et il surveillera le repo local.

TESTS UI

- Vous devez utiliser un outils de tests d'interfaces web (p.e. Selenium, Cypress, CasperJS ou autre) pour programmer une suite de tests fonctionnels complète pour l'application web;
- Les tests devraient couvrir l'exécution de code source et l'utilisation du menu d'exemples.

ETC

- Ce livrable compte pour **20 points sur 50**;
- Les 3 livrables seront corrigés en présentiel le **25 avril** pendant la période de cours et la période de laboratoire;
- Tout le système doit être fonctionnel sur l'ordinateur portable d'un membre de l'équipe ou sur un ordinateur de laboratoire;
- Des questions pourraient être posées aux membres de l'équipe pendant la correction;
- Des modifications au code source pourraient être demandées pendant la correction;
- La qualité du code source, des tests, de la configuration et de la documentation sera évaluée après le 25 avril.

PROJET

ÉNONCÉ DES LIVRABLES