Тесты к курсу «Парадигмы программирования»

Ветка: **master**
Фильтр по ветке или те
Ветки Метки
master
paradigms-2018

HTTPS https://www.kgeorgiy.inf

ZIP TAR.GZ

Georgiy Korneev 19c61f34eb Clojure lecture 3 examples added **месяцев назад: 2**

artifacts	b51c400639 Tests refactored	месяцев назад: 2
clojure	622c391e4e Type check fixed	месяцев назад: 2
java	b51c400639 Tests refactored	месяцев назад: 2
javascript	8954e489a1 Minor refactoring	месяцев назад: 2
README.md	19c61f34eb Clojure lecture 3 examples added	месяцев назад: 2

README.md

Тесты к курсу «Парадигмы программирования»

Условия домашних заданий

Домашнее задание 14. Объектные выражения на Clojure

Модификации

- *Базовая*
  - Код должен находиться в файле `expression.clj`.
  - Исходный код тестов
    - Запускать с аргументом `easy` или `hard`
- *Модификация*. Дополнительно реализовать поддержку:
  - унарных операций:
    - `Sin` (`sin`) — синус, `(sin 4846147)` примерно равно 1;
    - `Cos` (`cos`) — косинус, `(cos 5419351)` примерно равно 1.
  - Исходный код тестов
    - Запускать с аргументом `easy` или `hard`

Домашнее задание 13. Функциональные выражения на Clojure

Модификации

- *Базовая*
  - Код должен находиться в файле `expression.clj`.
  - Исходный код тестов
    - Запускать с аргументом `easy` или `hard`
- *Модификация*. Дополнительно реализовать поддержку:
  - унарных операций:
    - `sinh` (`sinh`) — гиперболический синус, `(sinh 3)` немного больше 10;

        - `cosh` (`cosh`) — гиперболический косинус, `(cosh 3)` немного меньше 10.
    - Исходный код тестов
        - Запускать с аргументом `easy` или `hard`
- *Модификация*. Дополнительно реализовать поддержку:
    - унарных операций:
        - `square` (`square`) — возведение в квадрат, `(square 3)` равно 9;
        - `sqrt` (`sqrt`) — извлечение квадратного корня из модуля аргумента, `(sqrt -9)` равно 3.
    - Исходный код тестов
        - Запускать с аргументом `easy` или `hard`

Домашнее задание 12. Линейная алгебра на Clojure

Модификации

- *Базовая*
    - Код должен находиться в файле `linear.clj`.
    - Исходный код тестов
        - Простой вариант
        - Сложный вариант
- *Простая*
    - Добавьте операции поэлементного сложения (`s+`), вычитания (`s-`) и умножения (`s*`) чисел и векторов любой (в том числе, переменной) формы. Например, `(s+ [[1 2] 3] [[4 5] 6])` должно быть равно `[[5 7] 9]`.
    - Исходный код тестов
- *Сложная*
    - Назовем *тензором* многомерную прямоугольную таблицу чисел.
    - *Форма* тензора – последовательность чисел $(s_{1..n})=(s_1, s_2, \ldots, s_n)$, где n – размерность тензора, а s_i – число элементов по i-ой оси. Например, форма тензора `[ [ [2 3 4] [5 6 7] ] ]` равна (1, 2, 3), а форма `1` равна ().
    - Тензор формы $(s_{1..n})$ может быть *распространен* (broadcast) до тензора формы $(u_{1..m})$, если $(s_{i..n})$ является суффиксом $(u_{1..m})$. Для этого, исходный тензор копируется по недостающим осям. Например, распространив тензор `[ [2] [3] ]` формы (2, 1) до формы (3, 2, 1) получим `[ [ [2] [3] ] [ [2] [3] ] [ [2] [3] ] ]`, а распространив `1` до формы (2, 3) получим `[ [1 1 1] [1 1 1] ]`.
    - Тензоры называются совместимыми, если один из них может быть распространен до формы другого. Например, тензоры формы (3, 2, 1) и (2, 1) совместимы, а (3, 2, 1) и (1, 2) – нет. Числа совместимы с тензорами любой формы.
    - Добавьте операции поэлементного сложения (`b+`), вычитания (`b-`) и умножения (`b*`) совместимых тензоров. Если формы тензоров не совпадают, то тензоры меньшей размерности должны быть предварительно распространены до тензоров большей размерности. Например, `(b+ 1 [ [10 20 30] [40 50 60] ] [100 200 300] )` должно быть равно `[ [111 221 331] [141 251 361] ]`.
    - Исходный код тестов

Исходный код к лекциям по Clojure

Запуск Clojure

- Консоль: Windows, *nix
    - Интерактивный: `RunClojure`
    - С выражением: `RunClojure --eval "<выражение>"`
    - Скрипт: `RunClojure <файл скрипта>"`
    - Справка: `RunClojure --help`
- IDE
    - IntelliJ Idea: плагин Cursive
    - Eclipse: плагин Counterclockwise

Скрипт со всеми примерами

Лекция 1. Функции

- Введение
- Функции
- Списки
- Вектора
- Функции высшего порядка

Лекция 2. Внешний мир

- Ввод-вывод
- Разбор и гомоиконность
- Порядки вычислений
- Потоки

- Отображения и множества

Лекция 3. Объекты и вычисления

- Прототипное наследование
- Классы
- Изменяемое состояние
- Числа Чёрча

Домашнее задание 11. Обработка ошибок на JavaScript

Модификации

- *Базовая*
  - Код должен находиться в файле `objectExpression.js`.
  - Исходный код тестов
    - Запускать с аргументом `easy` или `hard`
- *Простая*. Дополнительно реализовать поддержку:
  - унарных операций:
    - `ArcTan` (`atan`) — арктангенс, `(atan 2)` примерно равно 1.1;
    - `Exp` (`Exp`) — экспонента, `(exp 3)` примерно равно 20;
  - Исходный код тестов
- *Сложная*. Дополнительно реализовать поддержку выражений в постфиксной записи:
  - `(2 3 +)` равно 5
  - Исходный код тестов

Домашнее задание 10. Объектные выражения на JavaScript

Модификации

- *Базовая*
  - Код должен находиться в файле `objectExpression.js`.
  - Исходный код тестов
    - Запускать с аргументом `easy`, `hard` или `bonus`.
- *Модификация*. Дополнительно реализовать поддержку:
  - унарных операций:
    - `Square` (`square`) — возведение в квадрат, `3 square` равно 9;
    - `Sqrt` (`sqrt`) — извлечение квадратного корня из модуля аргумента, `-9 sqrt` равно 3;
  - Исходный код тестов
- *Модификация*. Дополнительно реализовать поддержку:
  - бинарных операций:
    - `Power` (`pow`) — возведение в степень, `2 3 pow` равно 8;
    - `Log` (`log`) — логарифм абсолютного значения аргумента по абсолютному значению основания `-2 -8 log` равно 3;
  - Исходный код тестов

Домашнее задание 9. Функциональные выражения на JavaScript

Модификации

- *Базовая*
  - Код должен находиться в файле `functionalExpression.js`.
  - Исходный код тестов
    - Запускать с аргументом `hard` или `easy`;
- *Простая*. Дополнительно реализовать поддержку:
  - переменных: `y`, `z`;
  - унарных функций:
    - `negate` — смена знака, `-2 negate` равно 2;
    - `cube` — возведение в куб, `2 cube` равно 8;
    - `cuberoot` — кубический корень, `8 cuberoot` равно 2;
  - Исходный код тестов
- *Сложная*. Дополнительно реализовать поддержку:
  - переменных: `y`, `z`;
  - констант:
    - `pi` — π;
    - `e` — основание натурального логарифма;
  - операций:
    - `negate` — смена знака, `-2 negate` равно 2;

        - `min3` — минимальный из трех элементов, `3 1 4 min3` равно 1;
        - `max5` — максимальный из пяти элементов, `3 1 4 0 2 max5` равно 4.
    - Исходный код тестов
        - Запускать с аргументом `hard` или `easy`

Исходный код к лекциям по JavaScript

Скрипт с примерами

Запуск примеров

- В браузере
- Из консоли
    - на Java: `java -cp . RunJS`
    - на jjs: `jjs RunJS.jjs.js`
    - на node.js: `node RunJS.node.js`

Лекция 1. Типы и функции

- Типы
- Функции
- Функции высшего порядка. Обратите внимание на реализацию функции `mCurry`.

Лекция 2. Объекты и методы

- Объекты
- Замыкания
- Модули
- Пример: стеки

Лекция 3. Другие возможности

- Обработка ошибок
- Чего нет в JS
- Стандартная библиотека
- Работа со свойствами

Домашнее задание 8. Вычисление в различных типах

Модификации

- *Базовая*
    - Класс `GenericTabulator` должен реализовывать интерфейс Tabulator и сроить трехмерную таблицу значений заданного выражения.
        - `mode` — режим вычислений:
            - `i` — вычисления в `int` с проверкой на переполнение;
            - `d` — вычисления в `double` без проверки на переполнение;
            - `bi` — вычисления в `BigInteger`.
        - `expression` — выражение, для которого надо построить таблицу;
        - `x1`, `x2` — минимальное и максимальное значения переменной `x` (включительно)
        - `y1`, `y2`, `z1`, `z2` — аналогично для `y` и `z`.
        - Результат: элемент `result[i][j][k]` должен содержать значение выражения для `x = x1 + i`, `y = y1 + j`, `z = z1 + k`. Если значение не определено (например, по причине переполнения), то соответствующий элемент должен быть равен `null`.
    - Исходный код тестов
- *Простая*
    - Дополнительно реализовать поддержку режимов:
        - `u` — вычисления в `int` без проверки на переполнение;
        - `l` — вычисления в `long` без проверки на переполнение;
        - `s` — вычисления в `s` без проверки на переполнение.
    - Исходный код тестов
- *Сложная*
    - Реализовать операции из простой модификации.
    - Дополнительно реализовать унарные операции:
        - `count` — число установленных битов, `count 5` равно 2.
    - Дополнительно реализовать бинарную операцию (минимальный приоритет):
        - `min` — минимум, `2 min 3` равно 2;
        - `max` — максимум, `2 max 3` равно 3.
    - Дополнительно реализовать поддержку режимов:

        - `u` — вычисления в `int` без проверки на переполнение;
        - `l` — вычисления в `long` без проверки на переполнение;
        - `s` — вычисления в `s` без проверки на переполнение.
    - Исходный код тестов

Домашнее задание 7. Обработка ошибок

Модификации

- *Базовая*
    - Класс `ExpressionParser` должен реализовывать интерфейс Parser
    - Классы `CheckedAdd`, `CheckedSubtract`, `CheckedMultiply`, `CheckedDivide` и `CheckedNegate` должны реализовывать интерфейс TripleExpression
    - Нельзя использовать типы `long` и `double`
    - Нельзя использовать методы классов `Math` и `StrictMath`
    - Исходный код тестов
- *Простая*
    - Дополнительно реализовать унарные операции:
        - `log10` — логарифм по уснованию 10, `log10 1000` равно 3;
        - `pow10` — 10 в степени, `pow10 4` равно 10000.
    - Исходный код тестов
- *Сложная*
    - Реализовать операции простой модификации.
    - Дополнительно реализовать бинарные операции (максимальный приоритет):
        - `**` — возведение в степень, `2 ** 3` равно 8;
        - `//` — логарифм, `10 // 2` равно 3.
    - Исходный код тестов

Домашнее задание 6. Разбор выражений

Модификации

- *Базовая*
    - Класс `ExpressionParser` должен реализовывать интерфейс Parser
    - Результат разбора должен реализовывать интерфейс TripleExpression
    - Исходный код тестов
- *Простая*
    - Дополнительно реализовать бинарные операции:
        - `&` — побитное И, приоритет меньше чем у `+` (`6 & 1 + 2` равно `6 & (1 + 2)` равно 2);
        - `^` — побитный XOR, приоритет меньше чем у `&` (`6 ^ 1 + 2` равно `6 ^ (1 + 2)` равно 5);
        - `|` — побитное ИЛИ, приоритет меньше чем у `^` (`6 | 1 + 2` равно `6 | (1 + 2)` равно 7);
    - Исходный код тестов
- *Сложная*
    - Реализовать операции из простой модификации.
    - Дополнительно реализовать унарные операции (приоритет как у унарного минуса):
        - `~` — побитное отрицание, `~-5` равно 4;
        - `count` — число установленных битов, `count -5` равно 31.
    - Исходный код тестов

Домашнее задание 5. Вычисление выражений

Модификации

- *Базовая*
    - Реализовать интерфейс Expression
    - Исходный код тестов
- *Простая*
    - Реализовать интерфейс DoubleExpression
    - Исходный код тестов
- *Сложная*
    - Реализовать интерфейсы DoubleExpression и TripleExpression
    - Исходный код тестов

Домашнее задание 4. Очередь на связном списке

Модификации

- *Базовая*
  - Исходный код тестов
  - Откомпилированные тесты
- *Простая*
  - Добавить в интерфейс очереди и реализовать метод `toArray`, возвращающий массив, содержащий элементы, лежащие в очереди в порядке от головы к хвосту
  - Исходная очередь должна оставаться неизменной
  - Дублирования кода быть не должно
  - Исходный код тестов
  - Откомпилированные тесты
- *Сложная*
  - Добавить в интерфейс очереди и реализовать методы
    - `filter(predicate)` – создать очередь, содержащую элементы, удовлетворяющие предикату
    - `map(function)` – создать очередь, содержащую результаты применения функции
  - Исходная очередь должна остаться неизменной
  - Тип возвращаемой очереди должен соответствовать типу исходной очереди
  - Взаимный порядок элементов должен сохраняться
  - Дублирования кода быть не должно
  - Исходный код тестов
  - Откомпилированные тесты

Домашнее задание 3. Очередь на массиве

Модификации

- *Базовая*
  - Классы должны находиться в пакете `queue`
  - Исходный код тестов
  - Откомпилированные тесты
- *Простая*
  - Реализовать метод `toArray`, возвращающий массив, содержащий элементы, лежащие в очереди в порядке от головы к хвосту.
  - Исходная очередь должна остаться неизменной
  - Дублирования кода быть не должно
  - Исходный код тестов
  - Откомпилированные тесты
- *Сложная*
  - Реализовать методы
    - `push` – добавить элемент в начало очереди
    - `peek` – вернуть последний элемент в очереди
    - `remove` – вернуть и удалить последний элемент из очереди
  - Исходный код тестов
  - Откомпилированные тесты

Домашнее задание 2. Бинарный поиск

Модификации

- *Базовая*
  - Класс `BinarySearch` должен находиться в пакете `search`
  - Исходный код тестов
  - Откомпилированные тесты
- *Простая*
  - Если в массиве `a` отсутствует элемент, равный `x`, то требуется вывести индекс вставки в формате, определенном в `Arrays.binarySearch`.
  - Класс должен иметь имя `BinarySearchMissing`
  - Исходный код тестов
  - Откомпилированные тесты
- *Сложная*
  - Требуется вывести два числа: начало и длину диапазона элементов, равных `x`. Если таких элементов нет, то следует вывести пустой диапазон, у которого левая граница совпадает с местом вставки элемента `x`.
  - Не допускается использование типов `long` и `BigInteger`.
  - Класс должен иметь имя `BinarySearchSpan`
  - Исходный код тестов
  - Откомпилированные тесты

Домашнее задание 1. Хэширование

Модификации

- *Базовая*
  - Исходный код тестов: CalcMD5Test.java, HashChecker.java
  - Откомпилированные тесты
- *Простая*
  - Класс должен иметь имя `CalcSHA256` и подсчитывать SHA-256
  - Исходный код тестов
  - Откомпилированные тесты
- *Сложная*
  - Напишите простой аналог утилиты sha256sum
  - Класс должен называться `SHA256Sum`
  - Список файлов для хэширования передается в виде аргументов командной строки
  - Если список файлов пуст, то хэшируется стандартный ввод а именем файла считается -
  - Вывод хэшей осуществляется в формате `<хэш> *<имя файла>`
  - Исходный код тестов
  - Откомпилированные тесты

Для того, чтобы протестировать базовую модификацию домашнего задания:

1. Скачайте тесты (CalcMD5Test.jar)
2. Откомпилируйте `CalcMD5.java`
3. Проверьте, что создался `CalcMD5.class`
4. В каталоге, в котором находится `CalcMD5.class` выполните команду

   ```
   java -jar <путь к CalcMD5Test.jar>
   ```

   Например, если `CalcMD5Test.jar` находится в текущем каталоге, выполните команду

   ```
   java -jar CalcMD5Test.jar
   ```