Code

"Tiny Tiny" QuickSort in Java

Intentions

To create a quick sort procedure for whole numbers with a naïve pivot that is as declarative as possible without typical implicit steps in Java.

Design

Used totallylazy (http://code.google.com/p/totallylazy/wiki/Sequence) sequence to build a declarative expression that adds a minimal amount of implicit expressions.

Code

```
public static Sequence<Integer> qsort(Sequence<Integer> nums) {
	if (nums.size() > 1) {
		final int pivot = nums.head();
		Sequence<Integer> rest = nums.tail();
		return qsort(rest.filter(lessThanOrEqualTo(pivot)))
				.add(pivot)
					.join(qsort(rest.filter(greaterThan(pivot))));
	} else if (nums.size() == 1){
		return sequence(nums.head());
	}
	return sequence();
}
```

Code Dependencies: totallylazy

Definition Language

Intentions

To create a compact definition language that can be easily processed and is sufficiently malleable to express simple metadata on JVM.

Design

Used Clojure's homoiconic features to encapsulate all metadata specifications with maps. Capable of defining metadata modules

All shown code is designed and written by C Kim

Code

```
(ns definitionlang)

(def *def-module* "definition module")
(def *definition* "definition")
(def *value* "value")
(def *imported-definition* "imported-definition")

(defmacro definition-module [name & exprs]
  `(proc-map
          (list ~@exprs)
          (create-module-map (str '~name))))

(defn create-module-map [name]
  {:type *def-module*
   :name name
   :imported-definitions ()
   :definitions ()})

(defmacro imported-definition [definition short-name]
  `{:type *imported-definition*
     :definition (str '~definition)
     :pseudonym (str '~short-name)})

(defmacro definition [name & values]
  `{:type *definition*
    :name ~name
    :values (list ~@values)})

(defmacro value [name value-type cardinality]
      `{:type *value*
   :name ~name
   :value-type (str '~value-type)
   :cardinality ~cardinality})

(defn get-type [map]
     (condp =  (:type map)
       *imported-definition* [:imported-definition :imported-definitions]
       *definition* [:definition :definitions]
       nil))

(defn do-list [map  mod-map]
  (let [map-type (get-type map)]
    (when-not (= map-type nil)
          (let [newlist (cons  map ((first map-type) mod-map))]
             (assoc mod-map (second map-type) newlist)))))

(defn proc-map [lst m]
         (when (first lst)
           (let [newmap (do-list (first lst) m)]
           (if-not (second lst)
                newmap
       (recur (rest lst) newmap)))))
```

All shown code is designed and written by C Kim

Example

```
(definition-module Lorem
        (imported-definition org.data data)
        (imported-definition org.numbers numbers)

        (definition "Definition-A"
          (value "Value-A" "Type-A" "0..n")
          (value "Value-Z" "Type-B" "1..1"))

        (definition "Definition-B"
          (value "Value-B1" numbers "0..n")
          (value "Value-B2" "Type-B" "1..1")))
```