1 Introduction

You will explore some aspects of and write some simple classes and methods in the µSmalltalk language.

2 Description

- (Adapted from Exercise 12 of Chapter 10 from *Programming Languages: Build, Prove, and Compare* (p. 804).) Implement a class `Rand` having the following protocol.
  - Class method `fromSeed:` creates a new random-number generator in which the argument is the seed (an integer).
  - Class method `new` creates a new random-number generator with a default (but unspecified) seed.
  - Instance method `next` answers the next random number and updates the seed. By default, the sequence of random numbers should be generated as follows:
    \[ s_{i+1} = 9 \times s_i + 5 \mod 1024 \]
    where \( s_0 \) is the initial seed.
  - (extra) Class method `fromSeed:withAPRNG:` creates a new random-generator in which the first argument is the seed (an integer) and the second argument is an applicative pseudorandom number generator (a block of one argument), which determines the sequence of random numbers to be generated.

- (Adapted from Exercise 15 of Chapter 10 from *Programming Languages: Build, Prove, and Compare* (p. 805).) Add to class `Array` a class method `from: aCollection` that makes an array out of the elements of another collection.
  (Note: Simply give the implementation of the class method; there should be no need to otherwise change the implementation of class `Array`.)

The important methods from the class protocol of `Array` are:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>new: anInteger</code></td>
<td>Create and answer an array of size <code>anInteger</code> in which each element is <code>nil</code>.</td>
</tr>
</tbody>
</table>

and the important methods from instance protocols of the `Collection` hierarchy are:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>at:put: key value</code></td>
<td>Modify the receiver by associating <code>key</code> with <code>value</code>. May add a new value of replace an existing value. Answer the receiver. (From the instance protocol for <code>KeyedCollection</code>.)</td>
</tr>
<tr>
<td><code>do: aBlock</code></td>
<td>For each element <code>x</code> of the collection, evaluate <code>(value aBlock x)</code>. (From the instance protocol for <code>Collection</code>.)</td>
</tr>
<tr>
<td><code>size</code></td>
<td>Answer how many element the receiver contains. (From the instance protocol for <code>Collection</code>.)</td>
</tr>
</tbody>
</table>
• Recall the implementations of the instance methods `size` and `addAll:` of the class `Collection`:

```smalltalk
(method size () (locals cnt)
  (set cnt 0)
  (do: self (block (x) (set cnt (+ cnt 1)))
     cnt)
(method addAll: (aCollection)
  (do: aCollection (block (x) (add: self x)))
     self)
```

– Implement the instance method `size` without using a block literal.
(Hint: Although one typically uses a block literal to construct the argument of `do:`, `do:` simply requires that its argument is an object that “walks like a block and swims like a block and quacks like a block”.)

– Implement the instance method `addAll: aCollection` without using a block literal.

– Comment on the feasibility and/or utility of abolishing block literals from µSmalltalk, from the perspective of a language designer/implementer and from the perspective of a language user.

• (Adapted from Exercise 18.5 of Chapter 10 from *Programming Languages: Build, Prove, and Compare* (p. 806).)

Recall the implementation of the instance method `detect:ifNone:` of the class `Collection`:

```smalltalk
(method detect:ifNone: (aBlock exnBlock) (locals answer searching)
  (set searching true)
  (do: self (block (x)
    (ifTrue: (and: searching {(value aBlock x)})
      (set searching false)
        (set answer x)))
    (if searching exnBlock {answer}))
```

Implement the instance method `detect:ifNone:` so as to avoid the final `if`.

3 Requirements and Submission

You may use the reference interpreter (see Appendix A), but there may only be one active laptop in each group.

At the end of class, submit the group’s solutions as hard-copy; be sure to include the names of all group members in the submission.

A Interpreter

A reference µSmalltalk interpreter is available on the CS Department Linux systems (e.g., `glados.cs.rit.edu` and `queeg.cs.rit.edu` and ICLs 1 and 2) at:

```
/usr/local/pub/mtf/plc/bin/usmalltalk
```

Use the reference interpreter to check your code.