Language Assignment #2: Smalltalk

Issued: Thursday, February 28
Due: Thursday, March 14

Purpose

This assignment asks you to begin using an object-oriented imperative programming language named Smalltalk, which is more object oriented than Java or C++. In Smalltalk, everything is an object. Smalltalk was designed by Alan Kay, Dan Ingalls, and Adele Goldberg, at Xerox PARC, in 1972.

Documentation

Smalltalk lecture slides are at:

```
1 pub/slides/slides-smalltalk.pdf
```

Smalltalk is not described, in an introductory way, in our textbook.

The onyx cluster has a Smalltalk interpreter, which is well documented:

```
1 $ info smalltalk
2 $ man gst
3 /usr/share/gnu-smalltalk
```

and demonstrated by:

```
1 pub/sum/smalltalk
```

There is also a whole book, but you probably don't want to look at it:

```
1 pub/doc/Bluebook.pdf
```

Assignment

Port the simple banking application at:

```
1 pub/la2
```

from Java to Smalltalk.

Try to model your Smalltalk solution on the Java solution. Thus, you will have multiple Smalltalk classes. Order is important: translate them like this:

```
1 gst Customer.st Account.st CheckingAccount.st \
2     SavingsAccount.st Bank.st
```

Hints and Advice

- Smalltalk has multiple "versions" of syntax, all of which are rather Neanderthal. Work from my `sum.st` example. Section 1.3 of the `info` documentation, *Syntax of GNU Smalltalk* might be useful.
- Like Java, Smalltalk classes have constructors, which are just static (i.e., class) methods. You can name your constructors whatever you want: they don't have to be named `new`, but that's the convention. You can define your own constructor, with initialization parameters, but it needs to call the parameterless `Object` constructor `new` to construct an object. Your constructor can then invoke an instance method on the object to initialize it.
- A method name can be the same as an instance-variable name. A formal-parameter name cannot be the same as an instance-variable name.
- Numbers are objects. Arithmetic uses message passing.
- A number can return a string representation of itself, with the `asString` method.
- A string can return its concatenation with another string, with the `,` (comma) method, like this:

```
1   s:=s , (account toString)
2        , (Character nl asString)
```

- An abstract class/method can be approximated like this:

```
1  accrue: rate [
2      ^self subclassResponsibility
3  ]
```