Language Assignment #2: Smalltalk

Issued: Thursday, February 9
Due: Thursday, March 9

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.

Translator

In our lab, `onyx` is the home-directory file server for its nodes (e.g., `onyxnode01`). There is also a shared directory for "apps" at `/usr/local/apps`. Nodes share a translator for Smalltalk, named `gst`, which is installed below `/usr/local/apps`, which is a non-standard location.

Due to network constraints, `onyx` can be reached from the public Internet, but a node can only be reached from `onyx`. So, you can SSH and login to `onyx`, then SSH and login to a node.

An easy way to use `gst`, from a node, is to permanently add a line to the end of your `.bashrc` file. To do so, login to a random node, from `onyx`, by executing the script:

```
pub/bin/sshnode
```

Then, execute the script:

```
pub/bin/bashrc
```

Don't change your `$PATH`; just execute the script. Then, logout from the node and login to a node.

Documentation

Smalltalk lecture slides are at:

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

Smalltalk is demonstrated by:

```
pub/sum/smalltalk
```

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

Links to programming-language documentation can be found at:

```
http://csweb.boisestate.edu/~buff/pl.html
```

Assignment

Port the simple banking application at:

```
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:

```
gst Customer.st Account.st CheckingAccount.st \
    SavingAccount.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:

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

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

```
accrue: rate [
    ^self subclassResponsibility
]
```