# Homework #2: Chapters 1–2 & Smalltalk

**Issued:** Tuesday, September 12
**Due:** Tuesday, September 26

There are two parts to this homework assignment. The first part allows you to reinforce your reading, by performing Exercises from the ends of Chapters 1 (Introduction) and 2 (Programming Language Syntax). The second part allows you to program in a language introduced in lecture: Smalltalk.

## Part 1: Textbook Exercises

First, read the question in or textbook. Then, read what this handout has to say about the question.

1.1 This question asks you to give examples of various kinds of errors. Use Java for your examples.

1.6 This question asks you to consider how a parse/syntax tree is traversed differently, during interpretation versus compilation. In our upcoming Translator Assignment, our translator performs interpretation *and* compilation.

1.8 This question asks you to consider the accuracy of `make`. We discussed `make` during the lecture on a typical Unix C/C++ toolchain. Note that plain-old `make` relies on file-system time-of-last-modification timestamps.

2.1 Only do parts a, b, and c.

This question asks you to construct regular expressions for strings in three regular languages: character-string literals in C, comments in Pascal, and numeric literals in C.

Specify the simplified regular languages as described in our textbook, not the real ones from real C and Pascal!

For each of the three parts, start by giving some example strings in the regular language.

2.13 Only do parts a and b.

This question asks you to give a parse tree and a rightmost derivation, of a string, according to a context-free grammar.

Note that we are not covering material past Section 2.1, including scanning algorithms, parsing algorithms, and grammar characteristics (e.g., LL(1)).

2.17 This question asks you to extend the grammar from Figure 2.25.

Note that we are not covering material past Section 2.1, but we can still use the grammar as an example.

In our upcoming Translator Assignment, you will extend a provided grammar, scanner, and parser, in a similar way.

# Part 2: Smalltalk

## Why Smalltalk?

Smalltalk is a foundational object-oriented (OO) and imperative language. It wasn't the first OO language, but it influenced those that followed. Smalltalk is more OO than Java or C++, since almost "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/hw2
```

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
]
```