Why Learn Smalltalk?

- A pure object-oriented language
  - All values are objects
  - All operations are message sends (the Smalltalk term for method calls)
- Historical context
  - One of the first object-oriented languages
  - Java was originally designed to provide “Smalltalk semantics in a C-like language”.
  - Still used today

Smalltalk: A Pure Object-Oriented Language

- All values are objects:
  - 3, ‘hello world’, true, etc...
- All operations (except assignment) are message sends:
  - 3 + 4, 20 negated, etc...
- Even control structures are sends:
  - 1 to: 5 do: [ :i | x := x+i ], etc...
- Classes are objects too!
  - Point new sends the new message to the Point class object

Some History

- 1964: Kristen Nygaard and Ole-Johan Dahl develop Simula—the first object-oriented language.
- 1966: Alan Kay starts grad school at the University of Utah, and learns object-oriented programming from a pile of Simula code left on his desk.
- 1971: Alan Kay develops Smalltalk-71 as a programming language for the KiddiKomp.

Some History, continued ...

- 1972: Smalltalk rewritten from scratch in response to a bet that Kay could define “the most powerful language in the world” in “a page of code”.
- 1976: Smalltalk-76 developed.
- 1980: Smalltalk-80 (modern Smalltalk) released by Xerox.

Some History, continued ...

- 1979-80: Apple bases Lisa user interface on Xerox’s SmallTalk Development environment. Lisa eventually evolved into the Macintosh.
- 1991: Bridge Systems develops a C-like language with SmallTalk semantics for Sun. This later evolves into Java.
- 1996: Kay and colleagues release Squeak - an open-source dialect of Smalltalk-80.
  - Try it out! Available at:
    - http://www.squeak.org/

Smalltalk: A Pure Object-Oriented Language

- All values are objects:
  - 3, ‘hello world’, true, etc...
- All operations (except assignment) are message sends:
  - 3 + 4, 20 negated, etc...
- Even control structures are sends:
  - 1 to: 5 do: [ :i | x := x+i ], etc...
- Classes are objects too!
  - Point new sends the new message to the Point class object

Smalltalk: A Pure Object-Oriented Language

- All values are objects:
  - 3, ‘hello world’, true, etc...
- All operations (except assignment) are message sends:
  - 3 + 4, 20 negated, etc...
- Even control structures are sends:
  - 1 to: 5 do: [ :i | x := x+i ], etc...
- Classes are objects too!
  - Point new sends the new message to the Point class object

Smalltalk:

A Pure Object-Oriented Language

- All values are objects:
  - 3, ‘hello world’, true, etc...
- All operations (except assignment) are message sends:
  - 3 + 4, 20 negated, etc...
- Even control structures are sends:
  - 1 to: 5 do: [ :i | x := x+i ], etc...
- Classes are objects too!
  - Point new sends the new message to the Point class object
Basic Smalltalk Syntax

<table>
<thead>
<tr>
<th>Comment</th>
<th>“Your comment”</th>
</tr>
</thead>
<tbody>
<tr>
<td>String literal</td>
<td>‘hello world’</td>
</tr>
<tr>
<td>Character literal</td>
<td>$a</td>
</tr>
<tr>
<td>Booleans</td>
<td>true, false</td>
</tr>
<tr>
<td>Arrays</td>
<td>#(2 3 5 10)</td>
</tr>
<tr>
<td>Assignment</td>
<td>x := 5</td>
</tr>
<tr>
<td>Return</td>
<td>^ resultValue</td>
</tr>
<tr>
<td>Statement Separator</td>
<td>stmt1 . stmt2</td>
</tr>
</tbody>
</table>

Messages

- All non-assignment operations in Smalltalk are message sends.
  - Like method calls in Java.
- Messages are sent to a receiver object
  - In Java, the receiver of x.foo(y,z) is x.
- Three types of messages: unary, infix binary, and keyword.
  - The main difference is how arguments are passed.

Unary Messages

- No arguments
- Syntax is:
  
```receiverObject methodName```
- Example:
  - 20 negated
  - myQueue front
  - Point new
  - Date today

Infix Binary Messages

- Consist of one or two non-alphabetic characters, like + or &&.
- Syntax is:
  
```receiver <binOp> argument```
- Examples:
  - 3 + 4
  - (x < y) & (3 <= 4)

Keyword Messages

- Take one or more arguments separated by keywords.
- Syntax:
  
```rcvr keyword1: arg1 keyw2: arg2```
- Examples:
  - x at: 5 put: $a
  - Array new: 10
  - 4 printOn: Transcript base: 8
  - Point3D x: 4 y: 5 z: 10 negated

Precedence of Messages

- Precedence of messages:
  - First, send unary messages
  - Then, infix binary messages.
  - Finally, send keyword messages.
- Multiple messages of the same type are sent in left to right order.
  - 5 negated squared
- Only one keyword message is allowed per statement, unless we use parenthesis.
  - x foo: 5 bar: 6
  - (x foo: 5) bar: 6.
Class Exercise:

How do the following expressions evaluate?

- $5 + 3 \times 2$
  - Answer: $7 + 9$ negated
  - Answer: $(5 \text{ multBy: (4 multBy: 3)})$ negated
  - Answer:

- $16$
  - Answer: 16
  - Answer: $7 + 9$ negated
  - Answer: $(5 \text{ multBy: (4 multBy: 3)})$ negated
  - Answer:

- $16$
  - Answer: 16
  - Answer: $7 + (-3)$ negated
  - Answer: $(5 \text{ multBy: (4 multBy: 3)})$ negated
  - Answer:

- $16$
  - Answer: 16
  - Answer: $-2$
  - Answer: $(5 \text{ multBy: (4 multBy: 3)})$ negated
  - Answer:
Class Exercise:

- How do the following expressions evaluate?
  - 16
  - Answer: 16
  - -2
  - Answer: -2
  - 6 multipliedBy: 4
  - Answer:
  - (5 multBy: (4 multBy: 3)) negated
  - Answer:

Class Exercise:

- How do the following expressions evaluate?
  - 16
  - Answer: 16
  - -2
  - Answer: -2
  - 24
  - Answer: 24
  - (5 multBy: (4 multBy: 3)) negated
  - Answer:

Class Exercise:

- How do the following expressions evaluate?
  - 16
  - Answer: 16
  - -2
  - Answer: -2
  - 24
  - Answer: 24
  - (5 multBy: (4 multBy: 3)) negated
  - Answer:

Class Exercise:

- How do the following expressions evaluate?
  - 16
  - Answer: 16
  - -2
  - Answer: -2
  - 24
  - Answer: 24
  - (5 multBy: (4 multBy: 3)) negated
  - Answer:

Class Exercise:

- How do the following expressions evaluate?
  - 16
  - Answer: 16
  - -2
  - Answer: -2
  - 24
  - Answer: 24
  - (5 multBy: (4 multBy: 3)) negated
  - Answer:

Class Exercise:

- How do the following expressions evaluate?
  - 16
  - Answer: 16
  - -2
  - Answer: -2
  - 24
  - Answer: 24
  - (5 multBy: (4 multBy: 3)) negated
  - Answer:

Defining a class

- In Smalltalk, every class has a superclass (except the Object class).
- Recall, classes are objects—thus, we can send them messages.
- To define a new class, we simply send a subclass keyword message to its superclass.
- All methods and variables of the superclass are inherited by the new subclass.
The Subclass Message

The subclass message takes the following arguments:

- **subclass**: The name of the new class.
- **instanceVariableNames**: Whitespace-separated string listing the fields of the new class.
- **classVariableNames**: List of variables that are shared by all instances (objects) of the new class.
- **poolDictionaries**: List of dictionaries that this class has access to.
- **category**: No semantic significance; helps the programmer organize classes.

Example - Point Class

Object subclass: #Point
instanceVariableNames: ‘x y’
classVariableNames: ‘OriginX OriginY’
poolDictionaries: ‘’
category: ‘CSE 413-Point Examples’

Defining Instance Methods

Once we have defined a class, we can define the messages types that an object of that class can receive.

- These are the “instance methods” of the class.
- If an incorrect message type is sent to an object, a runtime error is generated.
- Instance methods are entered by selecting the class, and clicking “Instance”.
- Method declarations consist of three parts: the header line, the local variables declarations, and the method body.

Example - Point Methods

An infix binary method:

```smalltalk
+ anotherPoint
| result |
result := Point new.
result x: x + anotherPoint x.
result y: y + anotherPoint y.
^ result
```

Shift the point:

```smalltalk
xShift: xs yShift: ys
x := x + xs.
y := y + ys
```
Class Exercise: scaleBy

Define a `scaleBy` method that multiplies all coordinates by a fixed factor.

Answer that returns a new point:

```smalltalk
scaleBy: factor |
result := Point new.
result x: x * factor.
result y: y * factor.
^ result
```

Answer that modifies the receiver:

```smalltalk
scaleBy: factor
x := x * factor.
y := y * factor
```

Defining Class Methods

- Recall that classes are objects too.
- Thus, we can also define the message types that a class object can receive.
  - These are the class methods.
- Common uses:
  - Constructors
  - Methods that have nothing to do with a specific instance (object) of the class.
  - To enter a class method, select the class and click on “Class”.

Example - Point Constructor

```smalltalk
x: xCoord y: yCoord |
p |  p := self new.
p x: xCoord.
p y: yCoord.
^ p
```

Example: Changing the Origin

```smalltalk
originX: newX y: newY
OriginX := newX.
OriginY := newY
```

I’ll explain why you should use `self new` rather than just `new` when we discuss dynamic dispatch.
Access Protection

- All messages/methods are public - anyone can send them.
- All variables are private - only methods of the class may access them.
- In fact, an object's variables are only added to the environment when a message to the object is evaluated.

Next Time

- How control structures are implemented in Smalltalk.
  - Hint: everything in Smalltalk is a message send!
- Self, Super, Inheritance and Dynamic Dispatch.
- A case study in object oriented design.