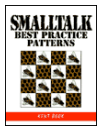
Naming Smalltalk Patterns

Stéphane Ducasse
Stephane.Ducasse@univ-savoie.fr
http://www.iam.unibe.ch/~ducasse/

Coding Standards

- Mainly from Smalltalk Best Practice Patterns by K. Beck
- Excellent
- Must read!



Coding Standards

- Standards
  - improve communication
  - let code be the design
  - make code more habitable
  - change

Coding Standards for Smalltalk

- Variables have no types
- Names can be any length
- Operations named with keywords
- Pretty printer

Names

- Names should mean something.
- Standard protocols
  - Object (printOn:, =)
  - Collection (do:, add:, at:put:, size)
- Standard naming conventions

Intention Revealing Selector

- Readability of message send is more important than readability of method
- Name should specify what method does, not how.

- aDoor open
- and not
- aDoor putPressureOnHandleThenPullWithRotation

Examples

ParagraphEditor>>***highlight***: aRectangle
    self reverse: aRectangle
If you would replace highlight: by reverse: , the system will run in the same way but you would reveal the implementation of the method.

Examples

If we choose to name after HOW it accomplished its task
    Array>>linearSearchFor:,
    Set>>hashedSearchFor:,
    BTree>>treeSearchFor:

These names are not good because you have to know the type of the objects.
Collection>>searchFor:
even better
    Collection>>includes:

Name your Method Well

Instead of:

```
setTypeList: aList
  "add the aList elt to the Set of type taken by the variable"

  typeList add: aList.
```

Write:

```
addTypeList: aList
  "add the aList elt to the Set of type taken by the variable"
  typeList add: aList.
```

Name Well your Methods

```
setType: aVal
  "compute and store the variable type"
   self addTypeList: (ArrayType with: aVal).
   currentType :=  (currentType computeTypes: (ArrayType with: aVal))
```

Not precise, not good

```
computeAndStoreType: aVal
    "compute and store the variable type"
    self addTypeList: (ArrayType with: aVal).
    currentType :=  (currentType computeTypes: (ArrayType with: aVal))
```

Precise, give to the reader a good idea of the functionality and not about the implementation

Method Names

Method Names

- If there is already a ***standard*** name, use it otherwise follow these rules.
- Three kinds of methods
  - change state of receiver
  - change state of argument
  - return value from receiver

Change State of Receiver

- method name is verb phrase
  - translateBy:
  - add:

Change State of Argument

- Verb phrase ending with preposition like ***on*** or ***to***.
  - displayOn:
  - addTo:
  - printOn:

Return Value from Receiver

- Method name is noun phrase or adjective, a description rather than a command
  - translatedBy:
  - size
  - topLeft

Method Names

- Specialized names for specialized purposes.
  - Double-dispatching methods
  - Accessing methods
  - Query methods
  - Boolean property setting
  - Converter methods

Accessing Methods

- Many instance variables have accessing methods, methods for reading and writing them.
- Same name than the instance variables
- Accessing methods come in pairs.
  - name, name:
  - width, width:
  - x, x:

When to use Accessing Methods

- Two opinions:
  - Always, including an object's own instance variable
    - lazy initialization, subclassing is easier
  - Only when you need to use it.
    - better information hiding
    - With the refactoring browser it is easy to transform the class using or not accessing

Query Method

- Methods that return a value often describe the type of the value because they are noun phrases.
- Query methods are not noun phrases, but are predicates.
- How can we make the return type clear?

- Provide a method that returns a Boolean in the "testing" protocol. Name it by prefacing the property name with a form of "be" or "has"- is, was, will, has

Testing Methods

- Prefix every testing method with "is".
  - isNil
  - isControlWanted
  - isEmpty
  - hasBorder

Converting Method

- Often you want to return the receiver in a new format.
- Prepend "as" to the name of the class of object returned.
  - asSet (in Collection)
  - asFloat (in Number)
  - asComposedText (in Text)

Classes

Simple Superclass Name

- What should we call the root of a hierarchy?
  - Complex name conveys full meaning.
  - Simple name is easy to say, type, extend.
  - But need to show that subclasses are related.

Simple Superclass Name

- Give superclasses simple names: two or (preferably) one word
  - Number
  - Collection
  - VisualComponent

Qualified Subclass Name

- What should you call a subclass that plays a role similar to its superclass?
  - Unique name conveys most information
  - Derived name communicates relationship to superclass

Qualified Subclass Name

- Use names with obvious meaning. Otherwise, prepend an adjective to most important superclass.
  - OrderedCollection
  - UndefinedObject
  - CloneFigureCommand, CompositeCommand, ConnectionCommand

Variables: Roles vs. Types

- Types are specified by classes
  - aRectangle
  - aCollection
  - aView
- Roles - how an object is used
  - location
  - employees
  - topView

Role Suggesting Instance Variable

- What should you name an instance variable?
  - Type is important for understanding implementation. But class comment can describe type.
  - Role communicates intent, and this harder to understand than type.

Role Suggesting Instance Variable

- Name instance variables for the role they play. Make the name plural if the variable is a collection.
  - Point: x, y
  - Interval: start, stop, step
  - Polyline: vertices

Type Suggesting Parameter Name

- Name of variable can either communicate type or role.
- Keywords communicate their parameter's role, so name of variable should give new information.

Type Suggesting Parameter Name

- Name parameters according to their most general expected class, preceded by "a" or "an". If there is more than one parameter with the same expected class, precede the class with a descriptive word.

Temporaries

- Name temporaries after role they play.
- Use temporaries to:
  - collect intermediate results
  - reuse result of an expression
  - name result of an expression
- Methods are simpler when they don't use temporaries!

Conclusion

Names are important
Programming is about
  communication
  intention
  …
Read the book:
  Smalltalk Best Practice Patterns
Even if you will program in Java or C#!

When the program compiles this is the start not the end…