CS 683 Emerging Technologies: Embracing Change
Spring Semester, 2001
Doc 7 Coding Patterns
Contents

References

This document is taken from the following two references. All the good ideas here are from Beck and Johnson. Quoted text in this lecture is from Beck's book.

My contribution is just to realize that their ideas are well worth stealing. Johnson lecture on coding style alone is worth the price of his on-line course. Beck's text is a must own for a Smalltalk programmer.

CS 497 Object-Oriented Programming & Design, Lecture notes, Ralph Johnson, Department of Computer Science, UIUC, http://st-www.cs.uiuc.edu/users/cs497/lectures.html

Smalltalk Best Practice Patterns, Kent Beck, Prentice Hall, 1997

Copyright ©, All rights reserved. 2001 SDSU & Roger Whitney, 5500 Campanile Drive, San Diego, CA 92182-7700 USA. OpenContent (http://www.opencontent.org/opl.shtml) license defines the copyright on this document.

Coding Patterns
Kent Beck's Predictors of Good Style

- Once and Only Once

  "In a program written with good style,
  everything is said once and only once"

- Lots of Little Pieces

  "Good code invariable has small methods and small objects"

- Replacing Objects

  "When you can extend a system solely by adding new objects without modifying any existing objects, then you have a system that is flexible and cheap to maintain."

- Moving Objects

  Object should be easily moved to new contexts

- Rates of Change

  "Don't put two rates of change together"

Coding Standards

Consistency makes the code easier to read

A poor standard is better than mixing several good styles

Names

Names have big impact on the readability of code

Names should mean something

at:put:, size, printString

Use standard naming conventions

Class Names

Smalltalk class names:

- Use complete words, no abbreviations

  Names are read 100 to 1000 times more often than typed

  Abbreviations waste more time (reading) than they save

- First character of each word is capitalized

  SmallInteger, LimitedWriteStream, LinkedMessageSet

Simple Superclass Name

Superclass names

- Simple words
- One word preferred, two at maximum
- Convey class purpose in the design

Number
Collection
Magnitude
Model

Qualified Subclass Name

- Unique simple name that conveys class purpose

  If name is in common use

  Array, Number, String

  If the purpose is more important that class hierarchy

- Prepend an adjective to superclass name

  If the class hierarchy is important

  Subclass is conceptually a variation on the superclass

  OrderedCollection, LargeInteger, CompositeCommand

Method Names

If a standard name exists, use of over following these rules

Search the image for similar methods

Intention Revealing Selector

- Name methods after what the method does

  Don't name methods after how it works

  linearSearchFor: indicates how the method works

  searchFor: is better

  includes: is even better

To test a name

  Imagine a second very different implementation

  Will the name work for both implementations?

Types of Methods

General Types

- Modifies receiver
- Modifies argument
- Returns a value

Specialized Types with special rules

- Accessing methods
- Query methods
- Boolean property setting
- Converter methods

Names for Methods that Modify the Receiver

Use a verb phrase for the name

add:
flush
translateBy:
rotate
at:put:

Names for Methods that Modify an Argument

Use a verb phrase ending with preposition like:

- On
- To

displayOn:
displayOn:at:clippingBox:rule:fillColor:
addTo:
printOn:
storeOn:

Names for Methods that Return a value

Use a noun phrase or adjective

Use description not a command

size
capacity
translatedBy:

Accessing State

How does a method access instance variable?

- Directly

```
deposit: aFloat
  balance := balance + aFloat
```

- Indirectly

```
deposit: aFloat
  self balance: (self balance + aFloat)
```

Direct access
- Simpler
- Easier to read
- Better information hiding

Indirect access
- Makes subclassing easier
- Permits lazy evaluation

Accessing Methods

Methods setting and getting the state of an object

Use the name of the variable

x x:
balance balance:

Java prepends set and get to these methods

This is not the Smalltalk convention

Boolean Property Accessing

Don't make accessing methods with just a boolean argument

For setter use two methods starting with "make"

makeVisible, makeInvisible
makeClean, makeDirty

Add a "toggle" method if needed

toggleVisible

For getters use methods starting with "is"

isVisible
isDirty
isClean

Query Methods

Query methods
- Tests a property of an object
- Return a boolean

Prefix the name with a form of "be"
- is, was, will

isNil
isActiveDirectory
isEmpty

Use common English testing words

includes:

If the negation of a query method is common provide an inverse method

notNil
notEmpty

Place query methods in testing category (or protocol)

Converter Methods

Converter methods convert the receiver to another object with the same protocol

Examples
- Converting float to an integer
- Converting a string to a bag

Prepend "as" to the name of the class of the object returned

- asSet
- asString
- asFloat
- asSortedCollection

Converter Constructor Method

Converting an object to another with different protocol

Converting a string to a date object

Provide a class method that
- Takes the object to be converted
- Returns new object
- Place in the class of the new object
- Place the method in "instance creation" category

Name the method by prepending "from" to the class of the original object

Example - in the Date class have:

```
fromString: aString
"Returns a date object represented by aString"
```

Composed Method

Methods should perform one identifiable task

- Method's operations should be at same level of abstraction
- Method should be a few lines long

This minimizes

- Code copying in subclasses
- Number of methods needing changes in subclass

How to Use Composed Method
Top-Down

While writing a method

- Invoke several smaller methods
- The smaller method need not yet exist

Bottom-Up

Factor common code in a single method

Put long loop bodies into separate method

If lines of code need a comment, place the code in a method with Intention Revealing Selector

If you send two or more messages to an object in a method, add a method in that object with Intention Revealing Selector

Variable Names

Key pieces of information about a variable

- Its purpose or role it plays
- Its type

Roles or purpose

- Communicates intent
- Harder to understand than type

Role Suggesting Instance Variable Name

Name instance variables for the role they play

Use a plural name if the variable is a collection

Comment the type in the class comment

Type Suggesting Parameter Name

Keywords indicate the parameter's role

Name parameters after the most general expected type

```
at: anInteger put: anObject
add: aCharacter
```

If multiple parameters have same type, precede the class with descriptive name

Temporary Variable Names

Name temporaries after the role they play

Methods are simpler when they don't use temporaries

Don't avoid using temporaries, try to write methods that don't need them

Behavior & State
Explicit Initialization

How to initialize an instance variable to a default value?

Implement an "initialize" method
- Sets all the variable explicitly
- Call the initialize method from the class method "new"

```
Object subclass: #Timer
   instanceVariableNames: 'count period '
```

Instance Methods

Category: initialize

```
initialize
   count := 0.
   period := self defaultMillisecondPeriod
```

Category: private

```
defaultMillisecondPeriod
   ^1000
```

Class Methods

Category: instance creation

```
new
   ^self basicNew initialize
```

Lazy Initialization

How to initialize an instance variable to a default value?

Lazy initialization uses

- A getter method for all accesses to a variable
- Getter sets the value of the variable with default value method

Explicit initialization favors readability

Lazy initialization favors

- Flexibility
  Subclasses can easily change defaults

- Performance
  Initialization is done only when needed

In timer class remove initialize method and add:

```
period
   period isNil ifTrue: [period := self defaultMillisecondPeriod].
   ^period

count
   count isNil ifTrue: [count := self defaultCount].
   ^count

defaultCount
   ^0
```

Choosing Message

"The long term health of a system is all about managing themes and variations"

Avoid using conditional logic (if statement)

Replace if statement with message sends

Replace:

```
responsibility := (entry isKindOf: Film)
   ifTrue: [anEntry producer]
   ifFalse: [anEntry author]
```

With:

```
Responsibility := anEntry responsibility
```

Where in the Film class we have

```
responsibility
   self producer
```

And in the Entry class we have

```
responsibility
   self author
```

Method Comment

"Communicate important information that is not obvious from the code in a comment at the beginning of the method"

Types of information difficult to convey in code

- Method dependencies

  Sometimes another method must be called before this one

- To do list

- Reasons for change

  If you need to change a method written by someone else comment why

Formatting Methods

Consistency of code layout improves readability

Squeak's system browser formats code, use it

To improve your code layout and understanding of the issues involved read Kent Beck's book: Smalltalk Best Practice Patterns

Exercises

1. Find examples of code in the Squeak image that follow the guidelines given here.

2. Find examples of code in the Squeak image that violate the guidelines given here.

3. Contrast the readability and flexibility of the examples found in 1 & 2.