Good Smalltalk

- The objects modeled by your code represent the objects in the real world

- The methods on the objects represent the actions that take place in the real world

- Make it clear to another person how to *use* your code
- Unit tests are a useful tool here
  - each test should illustrate a use case, with its outcome

- Classes have appropriate superclasses
  - use the “is-a” test

- “Class-side” methods for:
  - operations on the collection of all instances
  - creation of instances
  - reading input, converting to instances
  - for modeling operations on the class itself

Once and only once

- Everything that your program needs to say about the problem is said *Once,* e.g.,
  - *meaningful abstractions are represented as classes*
  - *meaningful operations are represented as methods*
- Everything that your program needs to say about the problem is said *Once only,* e.g.,
  - Common code is factored-out and reused, not copied
  - Constants are represented as named methods

Naming

- Classes and methods are named in a way that is meaningful to their clients.
  - Names follow Smalltalk's conventions
    - capitalization
    - accessors
      - instance variable '*position*' has a getter *position* and a setter *position:*
- *Skublics* chapter 1 is all about naming

Use Descriptive Names

☛ **Guideline 1**

Choose names that are descriptive.

Example

✔ `timeOfDay`

✘ `tod`

✔ `milliseconds`

✘ `millis`

✔ `editMenu`

✘ `eMenu`

Capitalization

Guideline 3

Begin class names, global variables, pool dictionaries, and class variables with an upper case letter. If a compound word is used, each word should begin with an upper case letter.

Example

`Behavior`	"class"
`Display`	"global variable"
`CharacterConstants`	"pool dictionary"
`CurrentUser`	"class variable in a class called User"

Naming Classes

- Class names are singular
- Chose a name that *classifies* objects

☛ **Guideline 5**

Choose a name indicative of a classification of objects. Select the least restrictive name possible for a reusable class.

Example

✔	`ProblemReport`	
✘	`Application`	"too generic"
✔	`TreeWalker`	
✘	`TreeWalkerForBinaryTrees`	"too specific"

- Use a name that describes the *rôle*, not the *implementation*

☛ **Guideline 7**

Avoid naming a class that implies anything about its implementation structure.

Example

"A database for Problem Reports that uses a Dictionary. There is no need to tell the user the implementation."

✔ `PRDatabase`

✘ `PRDictionary`

"A proper name that is stored as a String."

✔ `ProperName`

✘ `ProperNameString`

"This class is not implemented with a Set; it is a specialized Set."

✔ `SortedSet`

Namespaces

- Pharo doesn't have namespaces

Guideline 6

To avoid name space collisions, add a prefix indicative of the project to the name of the class.

Example

✔	`PRFormat`	"PR abbreviation for ProblemReport"
✔	`PublisherFormat`	"...for an on-line publisher project"
✔	`NASASpaceShip`	"part of the NASA project"

Naming Variables

- Two conventions: *semantic* and *typed*
- *Semantic Names*
  - the name describes the rôle that the variable plays
  - e.g., newSizeOfArray, name, newPosition
    - *not* anInteger, aString, aPoint

- *Typed Names*
  - the name describes the kind of object that will be referred to by the variable
    - e.g., aNumber, aCollection, aString
  - Don’t be overly-specific
    - e.g, don’t say aSmallInteger, anArray, aByteString unless the object really must be of that class.
  - Usually, type names refer to a protocol, not to a specific class
  - don’t lie
    - don’t name a parameter aCollection if **nil** is a valid argument

- It can be OK to mix semantic and typed names
  - examples from the base classes:

```
Collection » inject: initialValue into: aBinaryBlock
SequenceableCollection » copyFrom: start to: stop
SequenceableCollection » findFirst: aBlock ifNone: errorBlock
String » padded: leftOrRight to: length with: char
Boolean » ifTrue: trueBlock ifFalse: falseBlock
```

Naming State variables

- name state variables using natural language words or phrases

Example
"Class PhoneBook"

✔ `phoneNumber`
✘ `number`

✔ `name`
✘ `labelForPerson`

- use a plural noun for a collection

"Class VideoGame"

✔ `player`
✘ `boardMan`

✔ `enemies`
✘ `badGuyList`

✔ `score`
✘ `value`

Boolean state variables

- Use predicate clauses:

"In class Face..."

✔ `eyesOpen` "true if eyes are open"

✘ `isHappy` "true if face shows a happy expression"

"isHappy implies a binary state limiting the use of this variable. Instead of storing whether or not the face is happy, the variable **expression,** from the example for Guideline 10 representing a tristate such as happy, sad and mellow, would be used in a method called **#isHappy** returning (expression = #happy)."

"In class Vehicle..."

✔ `fourWheelDrive`

✔ `motorRunning`

"In class AlarmClock..."

✔ `alarmEnabled`

Non-boolean State Variables

- Use common nouns and noun-phrases:

	Example
	"In class Face..."
✔	`nose`
✔	`expression`
✔	`numberOfFreckles`
	"In class Vehicle..."
✔	`numberOfTires`
✔	`numberOfDoors`
	"In class AlarmClock..."
✔	`time`
✔	`alarmTime`
	"In class TypeSetter..."
✔	`page`
✔	`font`
✔	`outputDevice`

Method Names

- The name of a method should tell a reader *what* the method does.
  - should not have to look at the implementation
  - should not even have to look at the method comment

☛ **Guideline 12**

Choose method names so that someone reading the statement containing the method can read the statement as if it were a sentence.

Example

✔
```
FileDescriptor seekTo: word from: self position
```
✘
```
FileDescriptor lseek: word whence: self position
```

Guideline 13

Use imperative verbs and phrases for methods which perform an action.[2]

Example

✔
```
Dog
    sit;
    lieDown;
    playDead.
```

✔
```
aReadStream peekWord
```

✘
```
aReadStream word
```

✔
```
aFace lookSurprised
```

✘
```
aFace surprised
```

✔
```
anAuctionBlock add: itemUpForSale
```

✔
```
Inspector openOn: anObject
```

✔
```
record deleteFieldAt: index
```

☛ **Guideline 14**

Use a phrase beginning with a verb, such as *is* or *has*, for methods that answer a Boolean when interrogating the state of an object.

Example

"A method to test if an object is a String"

✔ `isString`

"A method to test if a Person is hungry"

✔ `aPerson isHungry`

✘ `aPerson hungry`

"A method to check if a Vehicle has four wheels"

✔ `aVehicle hasFourWheels`

✘ `aVehicle fourWheels`

☛ **Guideline 15**

Use common nouns for methods which answer a specific object.

Example

"Answer the next item on the auction block."

✔ `anAuctionBlock nextItem`

"This could be the current or the next item on the auction block."

✘ `anAuctionBlock item`

✔ `aFace expression`

☛ **Guideline 16**

Avoid the parameter type or name in the method name if you are using typed parameter names.

Example

✔ `fileSystem at: aKey put: aFile`

✘ `fileSystem atKey: aKey putFile: aFile`

"for semantic-based parameter names"

✔ `fileSystem atKey: index putFile: pathName`

"useful when your class has several #at:put: methods"

✔ `fileSystem definitionAt: aKey put: definition`

✔ `aFace changeTo: expression`

✘ `aFace changeExpressionTo: expression`

The rule is: make *uses* of the message read well.
The method template is secondary.

- How to do this? test first!

☛ **Guideline 17**

Use a verb with a preposition for methods that specify objects. Use the preposition **on:** when a method operates on another object.

Example

✔ `at: key put: anObject`

✔ `changeField: anInteger to: anObject`

✔ `ReadWriteStream on: aCollection.`

✘ `ReadWriteStream for: aCollection.`

~~✔ `File openOn: stream`~~

~~✘ `File with: stream`~~

✔ `display: anObject on: aMedium`

✘ `display: anObject using: aMedium`