References

Object-Oriented Design Heuristics, Riel
Abstract Classes

Abstract class
A class that cannot be instantiated

Concrete class
A class that can be instantiated

Why Abstract Classes

- Define an abstraction
- Define a type
- Define interface for subclasses
- Define methods for subclasses
- Hide the existence of concrete subclasses
Defining Abstract Classes

Some languages have special syntax

```java
public abstract class NoObjects {
    public void aFunction() {
        System.out.println( "Hi Mom" );
    }
    public abstract void subClassMustImplement( int foo );
}
```
Defining Abstract Classes - Smalltalk

Mark methods as abstract with “self subclassResponsibility”

```
Collection>>do: aBlock
  self subclassResponsibility
```

Indicate class is abstract in class comment

Include list of abstract methods

Browser will create methods stubs in subclass
What does self subclassResponsibility do?

Informs reader
   Method is abstract
   Concrete subclasses need to implement the method

Raises an exception when executed to indicate
   Subclass did not implement an abstract method
   Created an instance of an abstract class

Informs browser which methods subclasses need to implement
How to Prohibit Instances of Abstract Class

Documentation is normally enough

Implement new so it throws an exception

Stream class>>new
"Provide an error notification that Streams are not created using this message."
self error: ('Streams are created with on: and with:')
How do subclass objects get created?

Stream class>>new
   self error: ('Streams are created with on: and with:')

PositionableStream class>>on: aCollection
   ^super new on: aCollection

What happens when this is done?

PositionableStream on: String new
How do subclass objects get created?

Use basicNew

PositionableStream class>>on: aCollection
  ^self basicNew on: aCollection

basicNew
  Does the same thing as new
  Is used to get around super class’s new method
  Only used in class instance creation methods
  Never implement basicNew
Smalltalk Collections

Object
  \[\uparrow\]
  Collection

Bag  SequenceableCollection  Set

ArrayCollection  Interval  OrderedCollection  Dictionary

Array  CharacterArray  SortedCollection

String  Text

Symbol

*Italic* - Abstract Class

**Bold** - Concrete class
Abstract Classes and Data

Abstract classes commonly do not have instance variables

How can they implement methods?

Identify a core set of abstract operations

Implement other methods using core methods
Collection Class

No instance variables

60 methods

Three abstract methods

add:
remove:ifAbsent:
do:

detect: aBlock ifNone: exceptionBlock

"Evaluate aBlock with each of the receiver's elements as the argument. Answer the first element for which aBlock evaluates to true."

self do: [:each | (aBlock value: each) ifTrue: [^each]].
^exceptionBlock value
Abstract Classes, Types and Hinges

Tagging (declaring) a variable to be an Abstract class instance

Indicates which operations are allowed on the variable

Allows any subclass to be used in the variable

Provides flexibility particularly in languages with static type checking

SomeClass>>foo: aCollection
  ^aCollection fold: [:a :b | a max: b].

public class SomeClass {
  public int foo(Collection a) { blah}
}

public class Restricted {
  public int foo(Array a) { blah}
}
Abstract Classes and Hiding Subclasses

Smalltalk VM on startup informs Filename of the correct concrete class for the current platform

file := 'foo' asFilename.
file class "MacOSXFilename (on my machine)"
Platform Independence Aside

Mac, PC and Unix have different end of line characters

When you read a file:
   Smalltalk converts the platform’s end of line character to cr

When you write a file
   Smalltalk converts cr to the platform’s end of line character

Same code
   Works on all three platforms
   Produces files with the correct end of line character
Hide the existence of concrete subclasses

String is an abstract class

String new
   Does not create a string object
   Creates an instance of a subclass
   Appears to create a String object

String subclasses
   Don’t add new methods
   Provide specific implementations
Strings Continued

| a |
a := String new.
a class. "returns ByteString"

| b |
b := (String with: (Character value: 3585)) "3585 is Thai character".
b class "returns TwoByteString"

| c |
c := String with: $a.
c class. "returns ByteString"
c at: 1 put: (3585 as Character).
c class "returns TwoByteString"

To learn about character encodings read: http://www.joelonsoftware.com/articles/Unicode.html
become: Smalltalk Magic

| c |
c := String with: $a.
c class.                "returns ByteString"
c at: 1 put: (Character value: 3585).
c class                "returns TwoByteString"

How did c change class?

a become: b

   Change all references to ‘a’ to reference ‘b’

   Change all references to ‘b’ to reference ‘a’

   ‘a’ basically becomes ‘b’ and ‘b’ becomes ‘a’
String Class Transformation without become?

Use composition

String has instance variable that holds real string

String forwards messages to the real string

String can replace the real string with a different object
Sample Implementation

Smalltalk.Core defineClass: #'String
  superclass: #{Core.CharacterArray}
  instanceVariableNames: 'realString'

  size
  ^realString size

  at: anInteger
  ^realString at: anInteger

  at: anInteger put: aCharacter
  aCharacter value > 256
  ifTrue: [realString := realString asTwoByteString].
  realString at: anInteger put: aCharacter.
Inheritance

What should I use as a super class?

A has a B
  Indicates that an instance variable of A is an instance of B

A is a B
A is a type of B
  Indicates that A is a subclass of B

A car has an engine, so car object contains an engine object

A BinarySearchTree has nodes, so it has instance variables left and right

A WordStream is a type of ReadStream so it is a subclass of ReadStream
Common Mistakes

Using has-a relation for inheritance

“I need access to engine methods in the car class and now I have it.”
Roles Verses Classes

2.11 Be sure the abstractions you model are classes and not simply the roles objects play.

```plaintext
mother := Mother new.
father := Father new
```

```
mother := Person new.
father := Person new.
```

![Class Diagram]

- Person
- Father
- Mother
- Child