CS 535 Object-Oriented Programming & Design
Spring Semester, 2003
Doc 11 Some OO Terms

References

Ralph Johnson Lecture notes, Lecture 3 Data Abstraction and Encapsulation, http://st-www.cs.uiuc.edu/users/cs497/lectures.html

Object-Oriented Design Heuristics, Riel, Chapter 2

Some OO Terms

- Abstraction
- Encapsulation
- Information Hiding
- Coupling
- Cohesion
- Polymorphism

Abstraction

"Extracting the essential details about an item or group of items, while ignoring the unessential details."

Edward Berard

"The process of identifying common patterns that have systematic variations; an abstraction represents the common pattern and provides a means for specifying which variation to use."

Richard Gabriel

Example

Pattern: Priority queue

Essential Details: length
items in queue
operations to add/remove/find item

Variation: link list vs. array implementation
stack, queue

Heuristic 2.8

A class should capture one and only one key abstraction

Key abstraction

- Main entity in a domain model

Look at nouns in requirements specification system description

Encapsulation

Enclosing all parts of an abstraction within a container

Class contains

- Variables
- All the code that accesses the variables

Heuristic 2.9

Keep related data and behavior in one place

Code that uses a lot of accessing methods of an object should be used to that object

(aPoint x squared + aPoint y squared) sqrt

verses

aPoint r

Information Hiding

An object should hide design decisions from its users

Hide

- What is stored & what is computed
- Classes used

How does Point story its data?

How does OrderedCollection hold elements?

We use the classes without knowing

Heuristic 2.1

All data should be hidden within it class

Smalltalk instance variables in can be accessed in:

- Instance methods of Class where they are defined
- Instance methods of subclasses of the Class where they are defined

Most languages have a construct for global access to data

- Smalltalk has shared variables
- Use sparingly
- Use for constants
- What is a constant?

Engineering Heuristics, Absolutes & Beginners

All design decisions involve trade offs

Heuristics are design decisions that are nearly always true

No heuristic is correct all the time

Beginners violate heuristics because

- They don’t understand the trade offs involved
- Don’t know about alternatives
- Habit

Smalltalk and Private Methods

Private method

- Used for some internal computation
- Not to be called from outside of the class

All instance methods in Smalltalk are publicly accessible

Put private methods in “private” protocol

Smalltalk programmers know not to use such methods

Two View of a Class: Inside & Outside

Users of a class care about

- Public methods
- English description
- Examples
- Tests

Users don’t need to know implementation details

Coupling

Strength of interaction between objects in system

How tangled together the classes are

Cohesion

Degree to which the tasks performed by a single module are functionally related

Ralph Johnson's Suggestions for Finding Abstractions

- Do one thing
- Eliminate duplication
- Keep rate of change similar
- Decrease coupling, increase cohesion
- Minimize interfaces
- Minimize size of abstractions
- Minimize number of abstractions

Do One Thing

Method should do on thing

- Method name should tell what it does

  findString:startingAt:
  asNumber
  asUppercase
  dropFinalVowels

Class should be what its name says

  String
  OrderedCollection
  Array
  ReadStream

Break complex classes/methods into simpler ones

Eliminate Duplication

(self asInteger - $a asInteger + anInteger) \\ 26 – (self asInteger - $a asInteger)

(self alphabetValue + anInteger) \\ 26 - self alphabetValue.

Keep rate of change similar

- Separate initial conditions from algorithm's temporary variables
- Separate tax tables from employee data from time cards

Minimize interfaces

Use the smallest interface you can

- Use Number instead of Float

Avoid embedding classes in names

- add: instead of addNumber:

Don’t check the class of an object

Minimize size of abstractions

Methods should be small

- Median size is 3 lines
- 10 lines is starting to smell

Classes should be small

- 7 variables is starting to smell
- 40 methods is starting to smell

VW 7.0 Base System

	Average	Mean
Variables / class	2.1	1
Methods / class	16.7	9
Carriage returns/method	7.6	5.0

Code used to generate Numbers

Variables Per Class

```
classes :=Smalltalk allClasses reject: [:each | each isMeta]
variablesInClass :=classes collect: [:each | each instVarNames size].
average :=((variablesInClass fold: [:sum :each | sum + each] )/
                variablesInClass size) asFloat.
mean := variablesInClass asSortedCollection at: variablesInClass size // 2.
max := variablesInClass fold: [:partialMax :each | partialMax max: each]
```

Methods Per Class

```
classes :=Smalltalk allClasses reject: [:each | each isMeta]
methodsInClass :=classes collect: [:each | each selectors size].
average :=((methodsInClass fold: [:sum :each | sum + each] )/
                methodsInClass size) asFloat.
mean := methodsInClass asSortedCollection at: methodsInClass size // 2.
max := methodsInClass fold: [:partialMax :each | partialMax max: each]
```

Minimize number of abstractions

A class hierarchy 6-7 levels deep is hard to learn

Break large system into subsystems, so people only have to learn part of the system at a time

Polymorphism

Objects with the same interface can be substituted for each other at run-time

Variables take on many classes of object

Objects will behave according to their type

Code can work with any object that has the right set of methods

In C++ polymorphism requires
- Inheritance
- Pointers
- Virtual functions

In Java polymorphism requires
- Inheritance or
- Interfaces

In Smalltalk polymorphism does not require inheritance

Example

```
Counter>>printOn: aStream
   aStream
      nextPutAll: ‘Counter(‘;
      nextPutAll: count printString;
      nextPutAll: ‘)’
```

aStream can be any object that implements nextPutAll:

Note we do not write:

```
Counter>>printOn: aStream
   aStream class = FileStream ifTrue:[ write to file ].
   aStream class = WriteStream ifTrue: [write to write stream]
   aStream class = TextCollector ifTrue: [write to Transcript]
```

Avoid Case Statements

Smalltalk has no case statement

OO programers send a message to object instead

Each type of object handles the message according to its type

Case statements make it harder to add new cases

Simplistic Example

Bank offers various types of accounts:

- Checking
- Savings
- CD
- Junior savings accounts

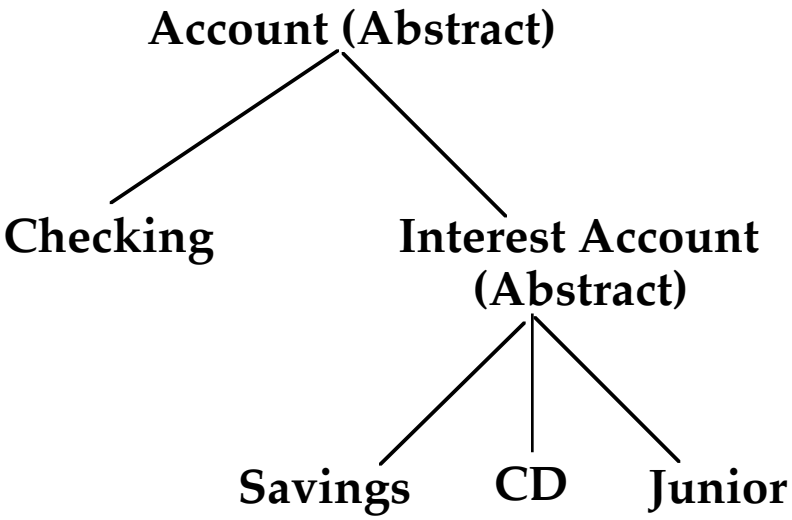
Each type has different rules for processing a transaction

Banking Classes

Customer

Transaction

Currency



Processing a Transaction

Using Case Statment

```
newCustomer := Bank createNewAccount: type.

Etc.

newCustomer class = Checking ifTrue:[ …]
newCustomer class = Savings ifTrue:[ …]
newCustomer class = CD ifTrue:[ …]
newCustomer class = Jonior ifTrue:[ …]
```

Polymorphism

newCustomer := Bank createNewAccount: type.
newCustomer.processTransaction: amount

Which processTransaction is called?

Adding new types of accounts to program requires:

- Adding new subclasses
- Changing code that creates objects

Avoid checking the class of an object