Clean Code Workshop

first day

Naming

name and conquer

Tim Ottinger

- Ottinger's Rules for Variable and Class Naming
- 1997
- Most downloaded OO-paper

Reveal Intent

Reveal Intent

- **int r;** // remaining days in month
- **int y;** // elapsed years since World War ||
- **void copy(int[] a1, int[] a2);** //Copy numbers from a1 to a2

Reveal Intent

- **int remainDaysInMonth;**
- **int elapsedYearsSinceWorldWar2;**
- **void copy(int[] source, int[] destination);**

Reveal Intent

```
def ss(aa)
  rr = 0
  uu = Math::sqrt(aa).floor
  if aa == 2 or aa == 1
    return True
  end

  1.upto(uu) do |i|
    if aa % i == 0
      return False
    end
  end

  return True
end
```

Avoid Disinformation

Avoid disinformation

- Avoid meaning collision
  - **XXXList**
  - **YYYTree**
  - **ZZZMap**
- Avoid visually similar names in same namespace
  - **thisIsAMethod**
  - **thisIsAMethod**

```
 66
 67 /**
 68  *  An abstract class that defines our requirements for manipulating
dates,
 69  *  without tying down a particular implementation.
 70  *  <P>
 71  *  Requirement 1 : match at least what Excel does for dates;
 72  *  Requirement 2 : class is immutable;
 73  *  <P>
 74  *  Why not just use java.util.Date?  We will, when it makes sense. At
times,
 75  *  java.util.Date can be *too* precise - it represents an instant in
time,
 76  *  accurate to 1/1000th of a second (with the date itself depending on
the
 77  *  time-zone).  Sometimes we just want to represent a particular day
(e.g. 21
 78  *  January 2015) without concerning ourselves about the time of day,
or the
 79  *  time-zone, or anything else.  That's what we've defined SerialDate
for.
 80  *  <P>
 81  *  You can call getInstance() to get a concrete subclass of
SerialDate,
 82  *  without worrying about the exact implementation.
 83  *
 84  * @author David Gilbert
 85  */
 86 public abstract class SerialDate implements Comparable,
 87                                              Serializable,
 88                                              MonthConstants {
 89
 90     /** For serialization. */
 91     private static final long serialVersionUID = -293716040467423637L;
 92
 93     /** Date format symbols. */
```

Use Problem Domain Names

Use Problem Domain Names

- int off_first_off_second = 0;
- int off_first_on_second = 1;
- int on_first_off_second = 2;
- int on_first_on_second = 3;

Use Problem Domain Names

- int open_interval = 0;
- int left_open_interval = 1;
- int right_open_interval = 2;
- int closed_interval= 3;

Use Problem Domain Names

- **enum {open_interval, left_open_interval, right_open_interval, closed_interval}**

Use Problem Domain Names

- **enum {open_interval, left_open_interval, right_open_interval, closed_interval}**
- **Add context**

Class & Methods

- Class
  - Nouns
  - Noun Phrases
- Methods
  - Verbs
  - Verb Phrases
  - boolean isXXX()

Variable Names in method

- The bigger the scope
  - the longer the name
- The smaller the scope
  - the shorter the name

Parameters names in methods

- The longer the better
  - Only write once
  - Kinda of documentation

Class variable names

- The longer the better
  - Only visible in current class
  - Long scope
  - Kinda of documentation

Method names in class

- The Public API (Bigger scope)
  - The shorter the better
- The Private Methods (Smaller scope)
  - As long as necessary

Naming Wrap Up

- Reveal intent
- Avoid disinformation
- Use Problem Domain Names
- Add Meaningful Context
- Names in methods
- Names in classes

Functions

How long should it be?

- Not longer than screen
- Ideal
  - **4 or 5 lines**
- Bottom Line
  - **<= 10 lines**

Why so short?

- No space for nested indentation
- No space for nested if, while, for
  - You must use separate function
    - with meaningful name
- No space for bugs

A Function works at one abstraction level

- **def check_out():**
  - **amounts = calculate_amounts()**
  - **discounts = calculate_discounts()**
  - **if (amounts > 1000)**
    - **discounts *= 1.2**
    - **shippings = 0**
  - **else**
    - **shippings = amounts * 0.05**
  - **end**
  - **amounts - discounts + shippings**
- **end**

A function does one thing

- Single responsibility
  - One abstraction level

How?

- Extract functions until you can’t
- **Wait for testableHtml refactoring demo**

Function Arguments

- Ask explicitly, do not dig!
  - **para1.getXXX().getYYY().getZZZ()**
  - Ask instead
    - **def function(zzz)**

Function Arguments

- Boolean Arguments
  - Is it two functions?
    - Maybe you should write two functions
    - **void checkOut(boolean isVIP)**

Function Arguments

- Different types whenever possible
  - **String encodeDate(int year, int month, int day)**
  - **String encodeDate(Year year, Month month, Day day)**

Function Arguments

- Number of arguments
  - <= 3
- What about more?
  - object?
  - Too much responsibilities?

Command-Query

- Command
  - side-effect
  - never return value
- Query
  - No side-effect
  - Return values

Output Argument

- void appendLastName(StringBuffer name) {
  - name.append(“Last name”);
- }

Exception Handling

- Never mix exception handling with normal execution flow
  - inconsistency
- A function may raise exception must be wholly wrapped

Function Wrap Up

- How long should it be?
- One abstraction level
- Single Responsibility
- Function Arguments?

What’s missing?

- Switch?
- Comments?
- Formatting?

Now, Demo Time!

Comments

- essential
  - API doc
  - warning
  - TODO:
  - why do this

Comments

- redundancy
  - useless code
    - use version control
  - explain
    - use meaningful naming
    - extract functions

Formatting

- use IDE recommended formatting
  - IDEA，RubyMine (**Ctrl + Alt + L**)

Exception

- different level has its own exception type
  - exception translation
- use unchecked exception
- exception vs return value

NULL

- don't use null as return value
  - use special object（eg. empty list）
  - use exception
- don't use null in parameters

Encapsulation

- **can not be changed** and **widely used**
  - third part library API (eg. xml parser)
  - native types (eg. String)
- collections (eg. List, Map, Set)