2IPC0 Programming Methods

From Small to Large Programs

Loek Cleophas
Eindhoven University of Technology
Department of Mathematics & Computer Science
Software Engineering & Technology Group

`http://canvas.tue.nl/courses/473`

Java Program Code: What Is Your Opinion?

```java
public boolean isSecure(int[][][] keys)
{
    for (int k1 = 0; k1 != keys.length; ++ k1) {
        for (int k2 = 0; k2 != keys.length; ++ k2) {
            if (k1 != k2) {
                boolean convertible; // can key k1 be converted into key
                convertible = true; // anticipated

                for (int r = 0; r != N_ROWS; ++ r) {
                    if (keys[k1][r].length != keys[k2][r].length) {
                        convertible = false;
                        break;
                    }

                    for (int i = 0; i != keys[k1].length; ++ i) {
                        if (keys[k1][r][i] > keys[k2][r][i]) {
                            convertible = false;
                            break;
```

Java Program Code: What Is Your Opinion?

```
                            }
                        } // end for i

                        if (! convertible) {
                            break;
                        }
                    } // end for r

                    if (convertible) {
                        return false;
                    }
                }
            } // end for k2
        } // end for k1

        return true;
    }
```

It Does Not Work

- This is everything, apart from a `main` method with some I/O
- Find the defect within 5 seconds

It Still Does Not Work

- The purpose is not described in a comment
- Find another defect within 5 seconds

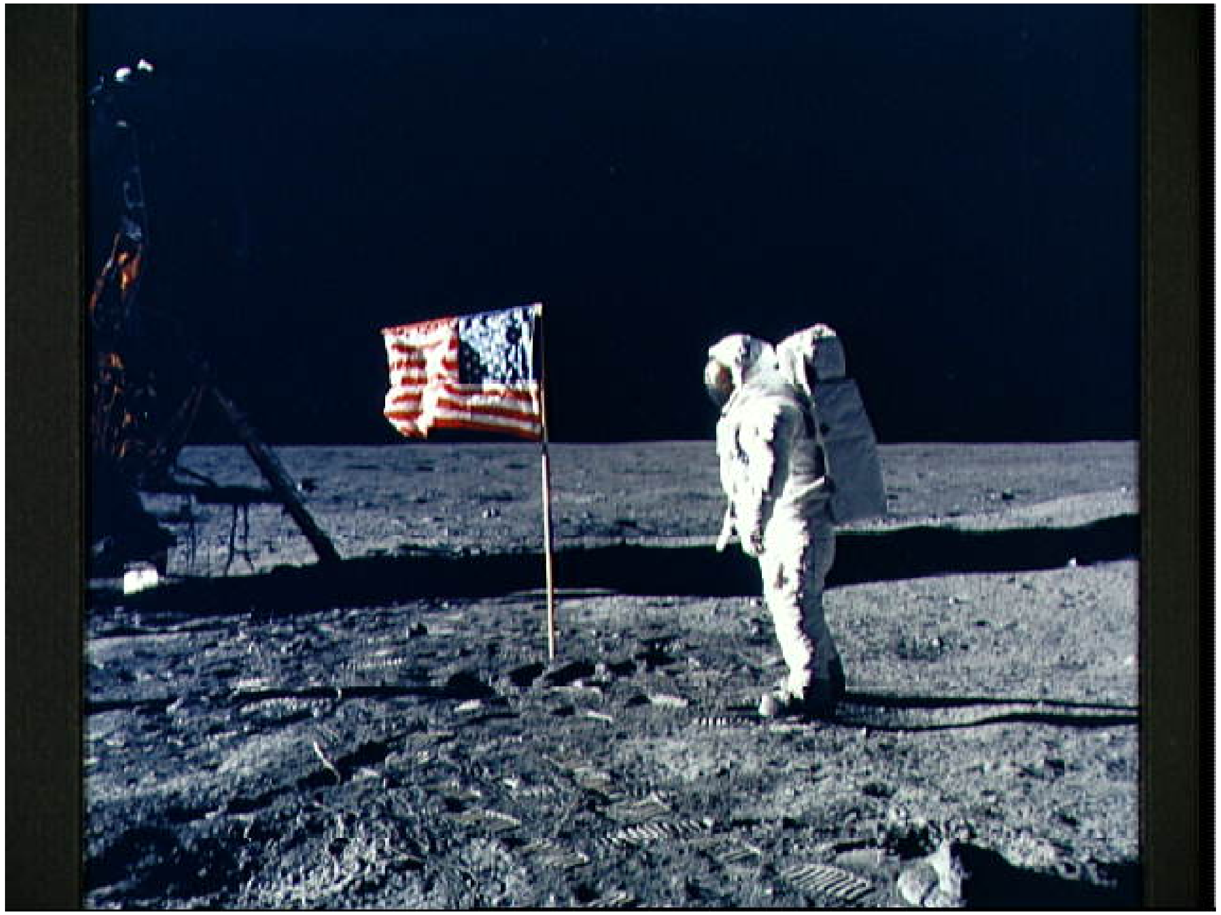
And It Still Does Not Work

- The method is too complex, lacking clear structure
- Structure should be introduced earlier, not when debugging
- Find yet another defect within 5 seconds

And It Still Does Not Work

- There is no unit test code
- Now, fix it (homework)

What Was the (Real) Goal?



Look beyond Planting the Flag

- `en.wikipedia.org/wiki/Edmund_Hillary` (1953)
  `en.wikipedia.org/wiki/Neil_Armstrong` (1969)
- Planting the flag is highly visible, but not the ultimate goal.

  Though the media would often like you to believe it is.
- Planting the flag brings you halfway, at best.
- The cost after planting the flag exceeds that of before.
- You must prepare for the second half before you depart.
- In software, planting the flag is the yell "it works".

  After that comes maintenance (and usually much more).

Overview

- Organizational issues, the big picture
- Motivation, background
- Procedural abstraction
- Functional decomposition
- Test-Driven Development

Compared to Last Year

Bachelor College: Registration $\rightarrow$ Participation $\rightarrow$ Examination

- Again 2 graded programming homework assignments (2IPC3)
- Again 2 graded intermediate written tests (2IPC2), but in week **3** and 6
- 12 compulsory non-graded programming homework assignments
- May miss 2 non-graded compulsory assignments

Weekly Schedule

Day	Time	What
(Monday	11:00–12:00	Staff meeting)
Wednesday	13:45–15:30	Instruction
Wednesday	15:45–17:30	Lecture (Aud. 2 + 9)
Friday	08:45–10:30	Lab Session
Friday	10:45–12:30	Lecture
Tuesday*	23:59	Homework DL (*this Friday: Mercurial Demo)

For details, see 2IPC0 (2016–2017) in Canvas or Oase

Who

Group	Wednesday Instructor
1	Loek Cleophas
2	Önder Babur
3	Serguei Roubtsov
4	Gerard Zwaan
5	Wieger Wesselink

Group	Friday Lab Assistants
1/2/3	Jimmy, Jari, Aleksandr
4/5	Geert, Niels, Mitchel

Schedule for Homework

Announced	**What**	**Due**
1	1+2 Compulsory PAs	Fri 10 / Tue 14 Feb 2017
2	2 Compulsory PAs	Tue 21 Feb 2017
3	1 Compulsory PA	Tue 7 Mar 2017 (d/t Carnaval)
	Graded programming assignment	Sun 12 Mar 2017 (d/t Carnaval)
4	2 Compulsory PAs	Tue 14 Mar 2017
5	2 Compulsory PAs	Tue 21 Mar 2017
6	1 Compulsory PA	Tue 28 Mar 2017
	Graded programming assignment	Sun 2 Apr 2017
7	1 Compulsory PA	Tue 4 Apr 2017

Schedule for Written Tests

Week	What	When
3	First Intermediate Test	Fri 24 Feb 2017
6	Second Intermediate Test	Fri 24 Mar 2017
10	Final Exam	Fri 21 Apr 2017

Why and How of Peer Review

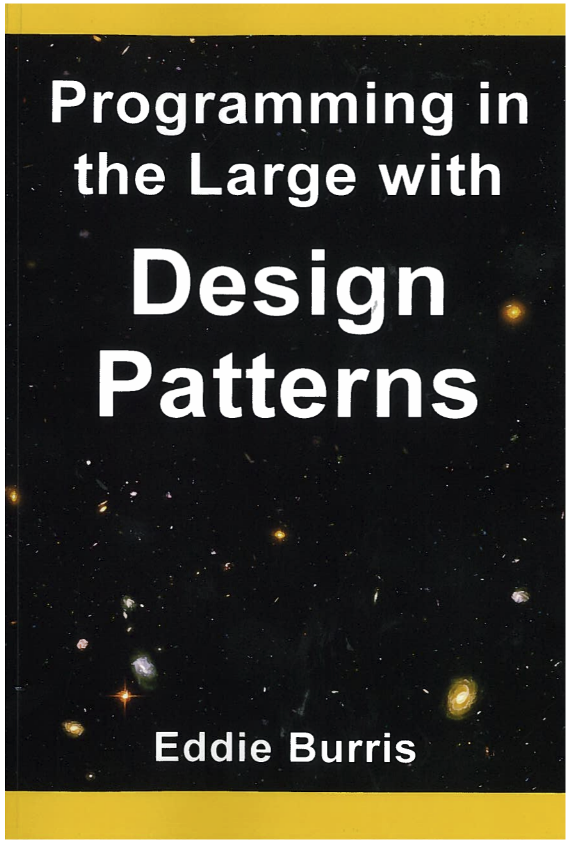
- Ability to review software designs and code is a learning objective
- Act of reviewing supports the learning process
- Carried out in a double-blind fashion

Study Material

- *Programming in the Large with Design Patterns* by Eddie Burris, Pretty Print Press, 2012.

- Slides, handouts, sample code

- Miscellaneous (web) resources



Tools

JDK8	`www.oracle.com/technetwork/java/javase/downloads`
NetBeans	`www.netbeans.org`
Javadoc	`docs.oracle.com/javase/8/docs/technotes/guides/javadoc/`
JUnit 4	`www.junit.org`
Mercurial	`mercurial.selenic.com`
DrJava	`www.drjava.org`
Profiler	integrated into NetBeans
Java Logging	part of the Java language: `java.util.logging`
UML	`www.uml-diagrams.org`

Motivation

- Assumption: You have some Java programming experience.

  Syntax, semantics, pragmatics (conventions)

  Types, variables, expressions, statements, input/output

- Goal: Systematic design of larger object-oriented Java programs

- Design: a blueprint; also: the activity leading to a blueprint

- Concepts, terminology, notation

  How to communicate about programs *and* about programming

- Systematic, rational design

  How to reason about programs, motivate design decisions

Product, Process, and Documentation

Product: machine-executable ‘working’ program or component

Product documentation: artifacts to support product (design)

Process: the way persons work (individually, or as a team)

Guidelines, step-by-step check lists, design methods, . . .

Process documentation: describes/prescribes a process

Focus on: methodical process, producing documented product

Not on product itself, not on creation of process documentation

How to create something big and complex?

- Engineering, technology
- Science
- How to be in control?
- How to improve probability of success?

How to create something big and complex?

“There are two ways of constructing a software design:

one way is
to make it so simple that there are
obviously no deficiencies,

and the other is
to make it so complicated that there are
no *obvious* deficiencies.”

C. A. R. Hoare (winner of Turing Award 1980)

The Essence of Engineering

"[I]t is the essence of modern engineering not only to be able to check one's own work, but also to have one's work checked and to be able to check the work of others.

In order for this to be done, the work must follow certain conventions, conform to certain standards, and be an understandable piece of technical communication."

H. Petroski.
To Engineer is Human: The Role of Failure in Successful Design.
Vintage Books, 1992.

How to Write Readable Java Source Code

(Background material)

- *The Way We Program* by Diomidis Spinellis
- *Code Conventions for the Java Programming Language*
- *Java Coding Standards* by ESA BSSC (2005)
- Concerns: layout, naming, commenting, structure

Coding Standard for 2IPC0

Layout

- NetBeans > Source > Format : default settings do a good job

  But: it may also mess up things!

  It is better if you write well-formatted code from the start

Comments

- Each **`public`** class, constructor, method, field has doc comment
- Each non-public class, constructor, method, field, local variable has (possibly non-doc) comment, but javadoc is preferred

Recap of Java

See ebook by David Eck: *Introduction to Programming Using Java*

Install DrJava and play with it; do some Interactions

```
> 2+2
4
> import javax.swing.JFrame;
> JFrame jf = new JFrame(); // no window visible
> jf.show(); // now locate the window on your computer screen
> jf.getWidth()
128
> jf.setSize(640, 480);
> jf.setTitle("Hello, here I am!");
```

Monolithic Programs

The imperative core of Java, consisting of

types, values, variables, expressions, assignment, selection, repetition

allows you to express every computation!

However, in that case,

programs are monolithic, without explicit structure, unmanageable

Procedural Abstraction

See Chapter 4 (Subroutines) of David Eck's book

Especially §4.1 through §4.4

Manage Complexity

Technique: ‘Divide and Conquer’ (and Rule)

General problem solving technique:

1. Split problem into subproblems (Decomposition)
2. Solve subproblems independently (Recursion possible)
3. Combine subproblem solutions into total solution (Integration)

Problems and Solutions in Java

Wrapped-up solutions in Java can take the form of

- methods
- classes
- packages
- . . .

Divide, Conquer, and Rule with Java Methods

- Specify a subproblem $\boxed{\textit{Divide}}$

  Syntax: method name, (formal) parameters, return type

  Semantics: Contract with pre, post, return

- Create (design and implement) subproblem solution $\boxed{\textit{Conquer}}$

  Method body

- Use subproblem solutions $\boxed{\textit{and Rule}}$

  Method invocations with arguments (actual parameters)

Hierarchical (recursive) subdivision:

  Create subproblem solution by using subsubproblem solutions

Method Contract Format

Precondition: Predicate in terms of global variables and parameters

Postcondition: Predicate in terms of global variables and parameters

`\old(v)` denotes value of `v` *on entry*

Typically *not* used for functions,
but can involve pseudo-variable `\result`

Returns: Function of global variables and parameters

'**Returns:** `F`' is equivalent to '**Postcondition:** `\result == F`'

Typically used for functions

Method Contract: Example Clauses

Precondition:

```
0 <= n

(\exists int i; 0 <= i < a.length; 0 < a[i])
```

Postcondition:

```
score.size() == 0

(\forall i, j; 0 <= i < j < a.length; a[i] <= a[j])

(\forall int v; ; C(a, v) == C(\old(a), v))
    where C(a, v) == (\num_of i; a.has(i); a[i] == v)
          a.has(i) == 0 <= i < a.length

0 <= \result <= 10 || \result == 100
```

Returns:

```
x ^ n

r such that 0 <= r < a.length && 0 < a[r]
```

Contract: concerns and benefits

		Two-sided Contract		
		Precondition		**Postcondition**
	User	concern		benefit
Party		$\downarrow$		$\uparrow$
	Provider	benefit	$\rightarrow$	concern

User takes care that precondition holds.

Provider exploits this precondition and takes care of postcondition.

User exploits this postcondition.

Specification of methods: two-sided contract

Contract serves as **interface** between

- **user** of solution (customer, client, caller), and
- **provider** of solution (supplier, implementor, callee)

Contract specifies *which* problem is solved,
by **precondition** and **postcondition** .

User needs not know *how* the problem was solved.

Provider needs not know *what* solution is used for.

Specification Plays Central Role

$$\begin{tikzcd}
 & \text{contract} \arrow[d, dash, double] & \\
\text{invocation} \arrow[ur] & {} & \text{implementation} \arrow[ul]
\end{tikzcd}$$

- Relate invocation (call) to contract and

  relate implementation to contract.

- *Never* relate invocation and implementation directly.

  That way, ‘divide’ fails.

  This leads to complexity and errors, en hence not to ‘conquer’.

Java Methods

- procedure : return type **void**, used as *statement*

  (a.k.a. command)

- function : return type not **void**, used as *expression*

  (a.k.a. query, inspector)

- parameters : name, type, **final** or not, modified or not

  order of parameters

- variables : local versus global

- aliasing : different names referring to the same entity

Test-Driven Development (TDD)

1. Gather and analyze requirements
2. Select requirement to develop
3. Specify class/methods informally: javadoc summary sentences
4. Specify formally: model with invariants, headers and contracts

   Class has no data representation and empty method bodies
5. Implement rigorous tests, in unit test class
6. Choose data representation and implement class methods
7. Test the implementation, and fix defects
8. . . . (see `test-driven-development.pdf`)

For larger classes, steps 5, 6, and 7 can be interleaved.

Assignments Series 1

- Read handouts (see Downloads)
- Practice with JUnit4 (Candy assignment)
- List some benefits and dangers of ‘Divide & Conquer’
- Apply procedural abstraction to Secure Key Collection
- Make sure all your code adheres to 2IPC0 coding standard

Technical Summary

- Coding conventions
- Manage complexity through structure
- Procedural Abstraction
- Functional Decomposition
- Techniques: Divide & Conquer, Test-Driven Development