CS1530, Lecture 14: Tactical Software Design

Bill Laboon

ACCIDENTAL VS ESSENTIAL COMPLEXITY

- Software is complex - hopefully we all argree on that
- Some is necessary - doesn’t mean that complexity is inherent or good in and of itself

ACCIDENTAL VS ESSENTIAL COMPLEXITY

- **Essential complexity** - essential to the problem being solved

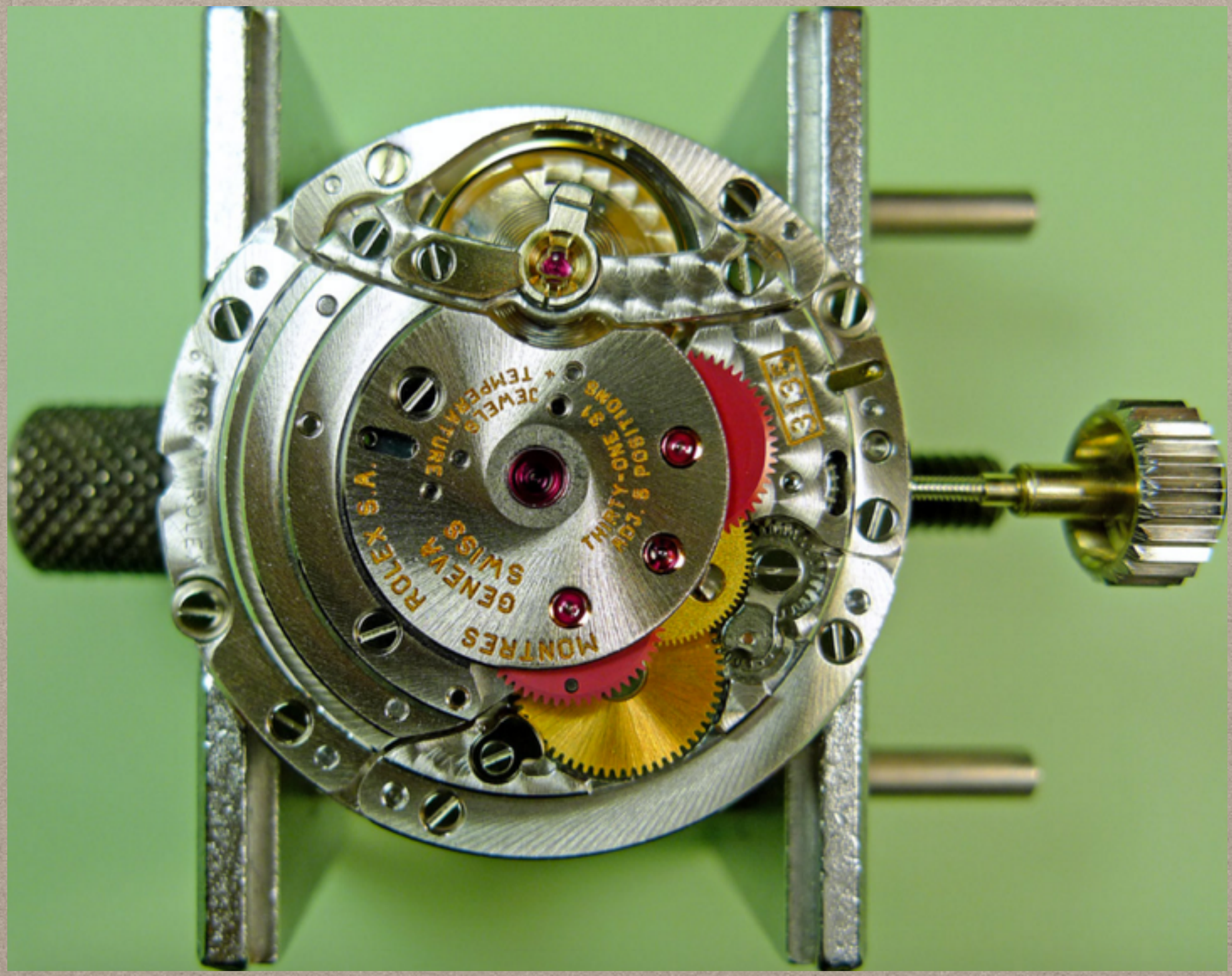
  EXAMPLE: Writing accounting software means implementing complex business logic to handle complex taxes.

ACCIDENTAL VS ESSENTIAL COMPLEXITY

- Accidental complexity - complexity caused by implementation

  EXAMPLE: Writing accounting software in Assembly language and using base-3 (trinary) for all internal calculations

A WATCH AS GUIDE FOR OBJECT-ORIENTED STYLE



From *"Rolex Caliber 3135 – Still worthy of the crown after all these years?"* http://www.chronometrie.com/rolex3135/rolex3135.html

A BEAUTIFUL EXAMPLE OF OBJECT-ORIENTED DESIGN!

From ABlogToWatch, ***Rolex Submariner Review: 114060 & 116610*** http://www.ablogtowatch.com/rolex-submariner-114060-116610-watch-review/

COMPLEXITY DIFFERENTIAL

- Note how complex the internals are compared to the input (crown) / output (hands)
- An ordinary user of the watch does not need to adjust mainspring vs balance spring, balance wheel, BPM, etc. - only turn crown or press/depress it
- Keep your classes like this!
- Hide complexity - keep “intellectual humility”

INFORMATION HIDING AND ENCAPSULATION

- Unless you are a watchmaker (or a watch nerd), you'll never even know these things exist
- Allows you to work at a higher abstraction level - "what time is it?" instead of confirming how many times the balance wheel has rotated compared to a baseline
- **Information hiding** - no need for knowledge of internals
- **Encapsulation** - can only interact with system via well-defined interface

POSTEL'S PRINCIPLE (A/K/A THE ROBUSTNESS PRINCIPLE)

- "Be conservative in what you send, but liberal in what you accept."
- A paraphrase of what Jon Postel wrote in the specification for TCP
- In other words, code as closely to the specification as you can, but don't assume that others will do so.
- Be prepared for invalid, outdated, or incorrect data

NOT ALWAYS A GOOD IDEA

- ON ERROR RESUME NEXT - A VB command saying that if an error occurs in a line, just go to the next one

- The VB compiler is being very liberal in what it accepts… but is that helping to build better code?

- What if received data is malformed? Should we try to guess what the user meant? Where is the line?

ABSTRACTIONS

- Systems are built on abstractions
- You write classes, which are an abstraction of Java code
- Java code is an abstration of JVM bytecode
- JVM bytecode runs on the JVM, which is an abstraction of the actual hardware
- etc.

ABSTRACTIONS ARE GOOD!

- You can think of the history of programming as piling up abstractions on top of each other
- I couldn’t get anything done if I worried about what x86 instructions were executing with every keystroke

LEAKY ABSTRACTIONS

- Abstractions should ideally abstract away all unnecessary implementation details of what they are hiding
- If they do not, this is called a “leaky abstraction”
- A leaky abstraction “leaks” details of its implementation / underlying details which those using the abstraction do not need to know

EXAMPLES OF LEAKY ABSTRACTIONS

- SQL: Performance characteristics can vary dramatically based on which columns are accessed, in which order joins occur, etc. Not obvious from SQL code!
- Accessing a file on a network share drive: many errors which can occur here (network failure, external server power failure, etc.) which would never happen for a local file.
- TCP over IP - “guarantees” a reliable connection, but you don’t always actually get one!

LEAKY ABSTRACTION CLASS

```
// Dog should be able to move and bite
// Implementation details (muscles) should
// not be visible!

public class Dog {
    public void bite(int strength) { … }
    public void move(Location l) { … }
    public void tenseMuscle(int muscleId){
      …
    }
    public void relaxMuscle(int muscleId){
      …
    }
}
```

SPOLSKY'S LAW OF LEAKY ABSTRACTIONS

"All non-trivial abstractions are, in some sense, leaky."

-Joel Spolsky

http://www.joelonsoftware.com/articles/LeakyAbstractions.html

WHY?

- You can never fully abstract everything away
- The abstraction is, at heart, implemented somehow
- For edge cases or performance, sometimes you will need to know how data is being abstracted

SPECIFIC JAVA TIPS

- Chapters 10 - 12 cover various method programming tips, but from a generic perspective (e.g., Java does not have implicit variable declaration)
- The following slides are a selection of techniques which are both relevant to Java and I have found useful (granted the latter part is a bit subjective - but nothing is stopping you from reading the whole chapters)

INITIALIZE YOUR VARIABLES AT DECLARATION TIME

```
int numAnimals = -1;
if (catsAvailable) {
    numAnimals = getKittens() + getCats();
} else {
    numAnimals = 0;
}
```

IF VARIABLES DON'T VARY, DON'T LET THEM!

```
// "final" is your friend

final int WHEELS_PER_CAR = 4;
```

AVOID MAGIC NUMBERS, VARIABLES SHOULD DESCRIBE

```
// Bad - what does the number/vars mean?
float m = 2.71828 ^ (r * t);

// Better
final float E = 2.71828;
float money = E ^ (interestRate * timeInYears)
```

DON'T IGNORE WARNINGS

The Java compiler is trying to help you!

Warnings mean that there is a possible error

See Warning.java

MINIMIZE VARIABLE SCOPE; ELIMINATE VARIABLE RE-USE

See VarReuse.java

PARTITION SIDE EFFECTS; MAKE CODE PURE

```
// Pure function (input -> output)
// No side effects (printing, writing to
// file, writing to global vars)
// Behavior only depends on args
public static int foo(int x, int y) {
    x = x + 5;
    int z = x + y;
    return z;
}
```

SIDE-EFFECTFUL METHODS

```
// Behavior depends on non-args
// Writes to disk, reads from database
// Also does calculation!
// Segregate calculation from side effects
public void bar() {
    int f = _a + _b - _c;
    String name = Database.getName(_name);
    DefaultFile.writeLine(name + ":" + f);
}
```