The Stopwatch tutorial

台北科技大學
資訊工程系
鄭有進 副教授
http://www.ntut.edu.tw/~yccheng
Problem statement

John is conducting a research to improve time performance of step A in an iterative algorithm that looks like

While more iterations to go
  step A;
  step B;
  step C;

Having devised a new method for step A, John wants to know how much improvement is achieved in computing time. Design a timer for this.
Vocabulary

- Timer n.
  - A timepiece, esp. one used for measuring intervals of time

- Stopwatch n.
  - A timepiece that can be instantly started and stopped by pushing a button

- Timepiece n.
  - An instrument that measures, registers, or records time.
A stopwatch
Building a domain model

- Some just know it; others need to see how it works.
  - A conceptual model with and without use cases.
- Aim for the essential concepts
  - Strip (or reduce) dependencies to implementation technology
- Get the model reviewed whenever possible.
Use case: time step A

1. Press the reset button to make sure that stopwatch is correctly initialized.
2. Right before step A, press start button.
3. The stopwatch starts measuring time off a reference clock.
4. Right after step A, press stop button.
5. The stopwatch ends measuring time, and register the accumulated time elapsed.
6. Repeat 2 to 5 until all iterations performed.
7. Read off the accumulated elapsed time.

Note: at anytime, currently accumulated time can be read even if the stopwatch is still on the go.
Domain (conceptual) modeling:

what

Nouns and noun phrases in the use case:

1. Press the **reset button** to make sure that **stopwatch** is correctly initialized.
2. Right before step A, press **start button**.
3. The stopwatch starts measuring **time** off a **reference clock**.
4. Right after step A, press **stop button**.
5. The stopwatch ends measuring time, and register the **accumulated time elapsed**.
6. Repeat 2 to 5 until all iterations performed.
7. Read off the accumulated elapsed time.

Note: at anytime, currently accumulated time can be read even if the stopwatch is still on the go.
Conceptual model: Stopwatch

Too physical?
Review: get the simplest thing that could possibly work

- The start/stop buttons can be combined
  - A toggle button as in the picture
- If used in a program, do we need two buttons for start and stop?
  - Since we can call the stopwatch object’s methods directly
- What happens if we press reset (start) while the stopwatch is on the go?
  - Stopwatch behavior is state-dependent.
State model of Stopwatch
A Design strategy: the simplest thing that could possibly work

- Begin with the straightforward
  - Just to prove that you can do it.
  - In the stopwatch example, you could start thinking procedure-oriented: implementing state transition logic and timing logic for each member function.

- Then ask what-if
  - For example, what if a new state “Broken” is added? In this case, all methods need to be modified.

- Improve the design so that what-if is better answered
  - partly what this course is about: better design
Design Model: solution (how) procedure-based

Enum {Idle, Running, Stopped};
...
void
Stopwatch::start()
{
    if (_state == Idle)
        _state = Running;
    // start measuring time;
    ...
    else if (_state == Running)
    ...
}
Design review

- State transition logic and timing logic are mixed
  - What if we discover a needed state for the Stopwatch later (e.g., Broken)? All methods must be re-examined for state transition logic and timing logic.

- How can object-orientation help?
  - Encapsulate states as objects
  - Delegate logic to state objects
Design Model: Object-oriented using the state pattern
Object orientation is about delaying implementation decisions.

- Problem statement
- Analysis model
- Design model
- Implementation model

procedure  object-based  object-oriented
public interface IStopWatch {
    public void start();
    public void stop();
    public void reset();
    public long getTime();
}

class StopWatch {
    private String _state = "STOPPED";
    private long _startTime = 0;
    private long _diffTime = 0;

    public void start() {
        if (_state == "IDLE" || _state == "STOPPED") {
            _startTime = System.currentTimeMillis();
            _state = "RUNNING";
        }
    }

    public void stop() {
        if (_state == "RUNNING") {
            _diffTime = System.currentTimeMillis() - _startTime;
            _state = "STOPPED";
        }
    }

    public void reset() {
        if (_state == "STOPPED") {
            _state = "IDLE";
            _startTime = 0;
            _diffTime = 0;
        }
    }

    public long getTime() {
        if (_state == "IDLE")
            return 0;
        else if (_state == "STOPPED")
            return _diffTime;
        else if (_state == "RUNNING")
            long t = System.currentTimeMillis();
            return (t - _startTime) + _diffTime;
        return 0;
    }
}
unit testing

- Introduction to JUnit
- Using JUnit on Eclipse, By Example
Introduction to JUnit
What is JUnit?

- JUnit is an open source Java unit testing framework used to write and run repeatable tests. It is an instance of the xUnit architecture for unit testing frameworks.
- Unit testing is white box testing, that is, testing performed by programmers.
White Box Testing, Black Box Testing
A Simplified Workflow of Applying JUnit
Using JUnit on Eclipse, By Example
New JUnit Test Case
New JUnit Test Case Wizard

1. Name: StopWatchTestZ
   Superclass: No superclass specified
   Constructor: No constructor specified
   Public static void main(String[] args)
   Add TestRunner statement for: test
   Class Under Test: StopWatch

2. Available methods:
   - StopWatch
   - stop
   - start
   - reset
   - getTime
   - finalize
   - notify
   - notifyAll

   3 methods selected.
Quick Run the test case
JUnit Test Result
Write Test Methods

- **setUp()**
  - Create the test fixture before each test case runs

- **tearDown()**
  - Destroy the test fixture after each test case run over

- **testXXXX()**
  - Every function which starts with “test” would be executed when the test is running
Test Case `assertXXX` Methods

- `assertEquals(expected, actual)`
- `assertTrue(boolean)`
- `assertNotNull(Object)`
- `assertNull(Object)`
- …

An assert method is silent when its proposition succeeds
But throws an exception if the proposition fails.
public void testInit_01() {
    // confirm the Stopwatch State
    TestCase.assertEquals(IStopWatch.IDLE, _sw.getState());

    // confirm the timing result
    TestCase.assertEquals(0, _sw.getTime());
}

public void testStart_01() {
    _sw.start();
    long f = System.currentTimeMillis();
    this.delay(1000);
    long f1 = System.currentTimeMillis();

    // confirm the Stopwatch State
    TestCase.assertEquals(IStopWatch.RUNNING, _sw.getState());

    // confirm the timing result
    TestCase.assertEquals(f1-f, _sw.getTime());
}

public void testStop_01() {
    // Do some actions
    _sw.start();
    long f = System.currentTimeMillis();
    this.delay(1000);
    long f1 = System.currentTimeMillis();
    _sw.stop();

    // confirm the Stopwatch State
    TestCase.assertEquals(IStopWatch.STOPPED, _sw.getState());

    // confirm the timing result
    TestCase.assertEquals(f1-f, _sw.getTime());
```java
public void testIll_01() {
    // Do some actions
    _sw.start();
    long f = System.currentTimeMillis();
    this.delay(1000);
    long f1 = System.currentTimeMillis();
    _sw.stop();
    this.delay(1000);
    _sw.start();
    long f2 = System.currentTimeMillis();
    this.delay(2000);
    long f3 = System.currentTimeMillis();

    // Confirm the Stopwatch State
    TestCase.assertEquals(StopWatch.RUNNING, _sw.getState());

    // Confirm the timing result
    TestCase.assertEquals(f1-f+f3-f2, _sw.getTime());
}
```
If the test case failed...
JUnit Overview

JUnit features include:

- Assertions for testing expected results
- Test fixtures for sharing common test data
- Test suites for easily organizing and running tests
- Graphical and textual test runners

Test fixture: the set of common resources or data that you need to run one or more tests
JUnit Architecture
Import a project
Import Wizard

1. Select
   - Choose "Existing Project into Workspace"

2. Import Project From File System
   - Project name: "StopWatch"
   - Project location: "Eclipse/workspace/StopWatch"
Add JUnit Reference to Eclipse Build Path – 1/4
Add JUnit Reference to Eclipse Build Path – 2/4

1. **Java Build Path > Libraries**
2. **JUnit**
3. **Add JARs...** → **Add External JARs...** → **JUnit**
4. **Configuring Variables...**
Add JUnit Reference to Eclipse Build Path – 3/4
Add JUnit Reference to Eclipse
Build Path – 4/4