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?
Interaction model

```
Actor1

startStop

startStop

getElapsedTime

Object1: Top Package::Stopwatch
```
State model of Stopwatch
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.
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
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.
public interface IStopWatch {
    public void start();
    public void stop();
    public void reset();
    public long getTime();
}

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;
    }
}
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.
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
JUnit Architecture
A Simplified Workflow of Applying JUnit
public void testStart_01() {
    _sw.start();
    long f = System.currentTimeMillis();
    try(Thread.sleep(2000));
    catch(Exception e){}
    long f1 = System.currentTimeMillis();
    TestCase.assertEquals(f1-f, _sw.getTime());
}

public void testAll_01() {
    _sw.start();
    try(Thread.sleep(3000));
    catch(Exception e){}
    _sw.stop();
    _sw.reset();
    _sw.start();
    long f1 = System.currentTimeMillis();
    try(Thread.sleep(2000));
    catch(Exception e){}
    long f2 = System.currentTimeMillis();
    TestCase.assertEquals(f2-f1, _sw.getTime());
}
public void testAll_02() {
    long f = System.currentTimeMillis();
    long f1=0, f2=0;
    _sv.start();
    try{
        Thread.sleep(1000);
        f1 = System.currentTimeMillis();
        _sv.stop();

        Thread.sleep(1000);
        f2 = System.currentTimeMillis();

        _sv.start();
        Thread.sleep(1000);
    } catch(Exception e){}

    long f3 = System.currentTimeMillis();

    testCase.assertEquals((f1-f)+(f3-f2), _sv.getTime());
}
Using JUnit on Eclipse, By Example
New JUnit Test Case
New JUnit Test Case Wizard

1. Specifying the package and class name.
2. Selecting test methods to be created.
Write Test Methods

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

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

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

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

...
Quick Run the test case
JUnit Test Result
If the test case failed...
Import a project
Import Wizard

1. Select "Existing Projects into Workspace" from the import source list.

2. In the "Import Project From File System" dialog, enter the project name as "StopWatch" and browse to the location "C:\eclipse\workspace\StopWatch".
Add JUnit Reference to Eclipse Build Path – 1/4
Add JUnit Reference to Eclipse Build Path – 2/4
Add JUnit Reference to Eclipse Build Path – 3/4
Add JUnit Reference to Eclipse Build Path – 4/4