CS 635 Advanced Object-Oriented Design & Programming
Spring Semester, 2015
Doc 3 Code Smells, Refactoring, Unit Tests
Feb 3, 2015

Review

Object-Oriented Programming is good as it promotes
  Code reuse
  More readable code
  More maintainable code
  Better designs

Basic OO Heuristics

Keep related data and behavior in one place

A class should capture one and only one key abstraction

Beware of classes that have many accessor methods defined in their public interface

OO History

Objects as a formal concept in programming - Simula 67

Smalltalk introduced the term object-oriented programming - 1970s

Became dominant programming methodology
    Early and mid 1990s

4

So Why is Software Still so Bad?

Code Smell

Hint that something has gone wrong somewhere in your code

http://c2.com/cgi/wiki?CodeSmell

Lists of Code Smells

A Taxonomy for "Bad Code Smells"

http://www.soberit.hut.fi/mmantyla/BadCodeSmellsTaxonomy.htm

Coding Horror: Code Smells

http://www.codinghorror.com/blog/2006/05/code-smells.html

Cunningham wiki c2

http://c2.com/cgi/wiki?CodeSmell

Code Smell - Utility Method

Utility methods are a sign that related data and operations are not together

Java & OO

In many situations we can not OO in Java

Can not keep data and operations together in many of Java's existing classes

Ruby, Objective-C & Smalltalk allow you to add to existing classes

9

Result

Can't practice OO in small cases

Develop poor habits

Lose benefits of OO but don't noticce

Code Smell - Vague Identifier

meetsCriteria

flag

This generally happens when the One Responsibility Rule has been violated

One Responsibility Rule

"A class has a single responsibility: it does it all, does it well, and does it only"

Bertrand Meyer

Try to describe a class in 25 words or less, and not to use "and" or "or"

If can not do this you may have more than one class

Tuesday, February 3, 15

Classifying Fowler's Code Smells

Bloaters	Long method Large Class Primitive Obsession Long Parameter List Data Clumps
Object-Orientation Abusers	Switch Statements Temporary Field Refused Bequest Alternative Classes with Different Interfaces
Change Preventers	Divergent Change Shotgun Surgery Parallel Inheritance Hierarchies

http://www.soberit.hut.fi/mmantyla/BadCodeSmellsTaxonomy.htm

Classifying Fowler's Code Smells

Dispensables	Lazy class Data class Duplicate Code Dead Code, Speculative Generality
Couplers	Feature Envy Inappropriate Intimacy Message Chains Middle Man

http://www.soberit.hut.fi/mmantyla/BadCodeSmellsTaxonomy.htm

Duplicate Code

Duplicate Code

Duplicate Code

Duplicate Code

Duplicate Code

Duplicate Code

Duplicate Code

Duplicate Code

Tuesday, February 3, 15

Long Method - Large Class

The average method size should be less than 8 lines of code (LOC) for Smalltalk and 24 LOC for C++

The average number of methods per class should be less than 20

The average number of fields per class should be less than 6.

The class hierarchy nesting level should be less than 6

The average number of comment lines per method should be greater than 1

Long Parameter List

a.foo(12, 2, "cat", "<tr>", 19.6, x, y, classList, cutOffPoint)

Divergent Change

One class is changed in different ways for different reasons

ShotGun Surgery

When you have to make a kind of change you have to make a lot of little changes in different locations

Feature Envy

A method seems more interested in a class other than the one it is in.

Data Clumps

Same three or four data items together in lots of places

Primitive Obsession

Using primitive types instead of creating small classes

Tuesday, February 3, 15

Switch Statements

How do you program without them?

Lazy Class

Class that is not doing enough to pay for itself

Data Class

Class with just fields and setter/getter methods

Data classes are like children.

They are okay as a starting point, but to participate as a grownup object, they need to take some responsibility

25

Inappropriate Intimacy

Classes that spend too much time delving into other classes private parts

Tuesday, February 3, 15

Message Chains

location = rat.getRoom().getMaze().getLocation()

Negative Slope

```
if (foo) {
    if (bar) {
        if (cat = dog) {
            if (rat < 10) {

                ...
```

28

Temporary Field

Field is only used in certain circumstances

Common case
    field is only used by an algorithm
    Don't want to pass around long parameter list
    Make parameter a field

Refused Bequest

Subclass does not want to support all the methods of parent class

Subclass should support the interface of the parent class

Refactoring

Tuesday, February 3, 15

Refactoring

Changing the internal structure of software that changes its observable behavior

Done to make the software easier to understand and easier to modify

32

When to Refactor

Rule of three

Three strikes and you refactor

When to Refactor

When you add a new function

When you need to fix a bug

When you do a code review

When Refactoring is Hard

Databases

Changing published interfaces

Major design issues

When you add a feature to a program

If needed Refactor the program to make it easy to add the feature

Then add the feature

Before you start refactoring

Make sure that you have a solid suite of tests

Test should be self-checking

Do I need tests when I use my IDEs refactoring tools?

Are your IDE refactoring tools bug free?

Eclipse Refactoring

Eclipse Refactoring Menu

Rename...	⌥⌘R
Move...	⌥⌘V
Android	▶
Change Method Signature...	⌥⌘C
Extract Method...	⌥⌘M
Extract Local Variable...	⌥⌘L
Extract Constant...	
Inline...	⌥⌘I
Convert Anonymous Class to Nested...	
Convert Member Type to Top Level...	
Convert Local Variable to Field...	
Extract Superclass...	
Extract Interface...	
Use Supertype Where Possible...	
Push Down...	
Pull Up...	
Extract Class...	
Introduce Parameter Object...	
Introduce Indirection...	
Introduce Factory...	
Introduce Parameter...	
Encapsulate Field...	
Generalize Declared Type...	
Infer Generic Type Arguments...	
Migrate JAR File...	
Create Script...	
Apply Script...	
History...	

40

Rename Class

```
public class Foo {
        public int foo() {
                return 10;
        }
}
```
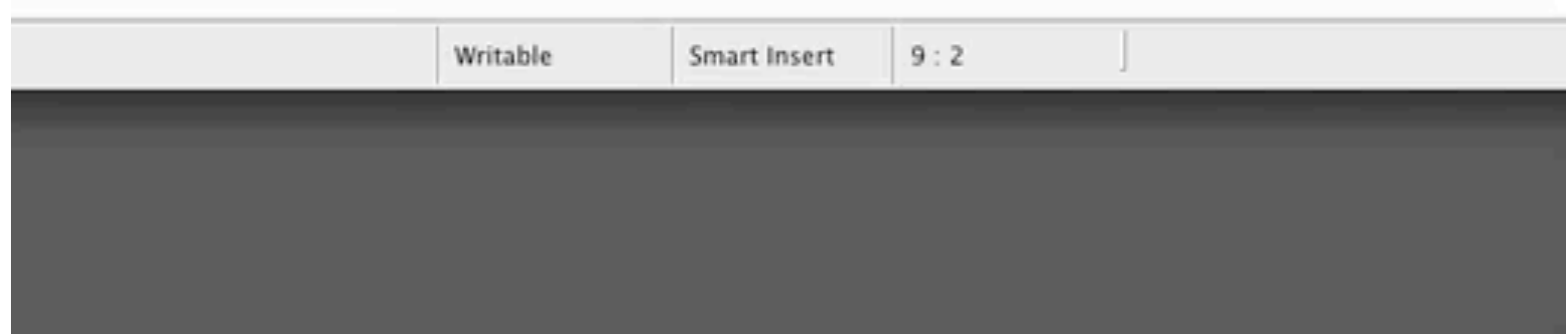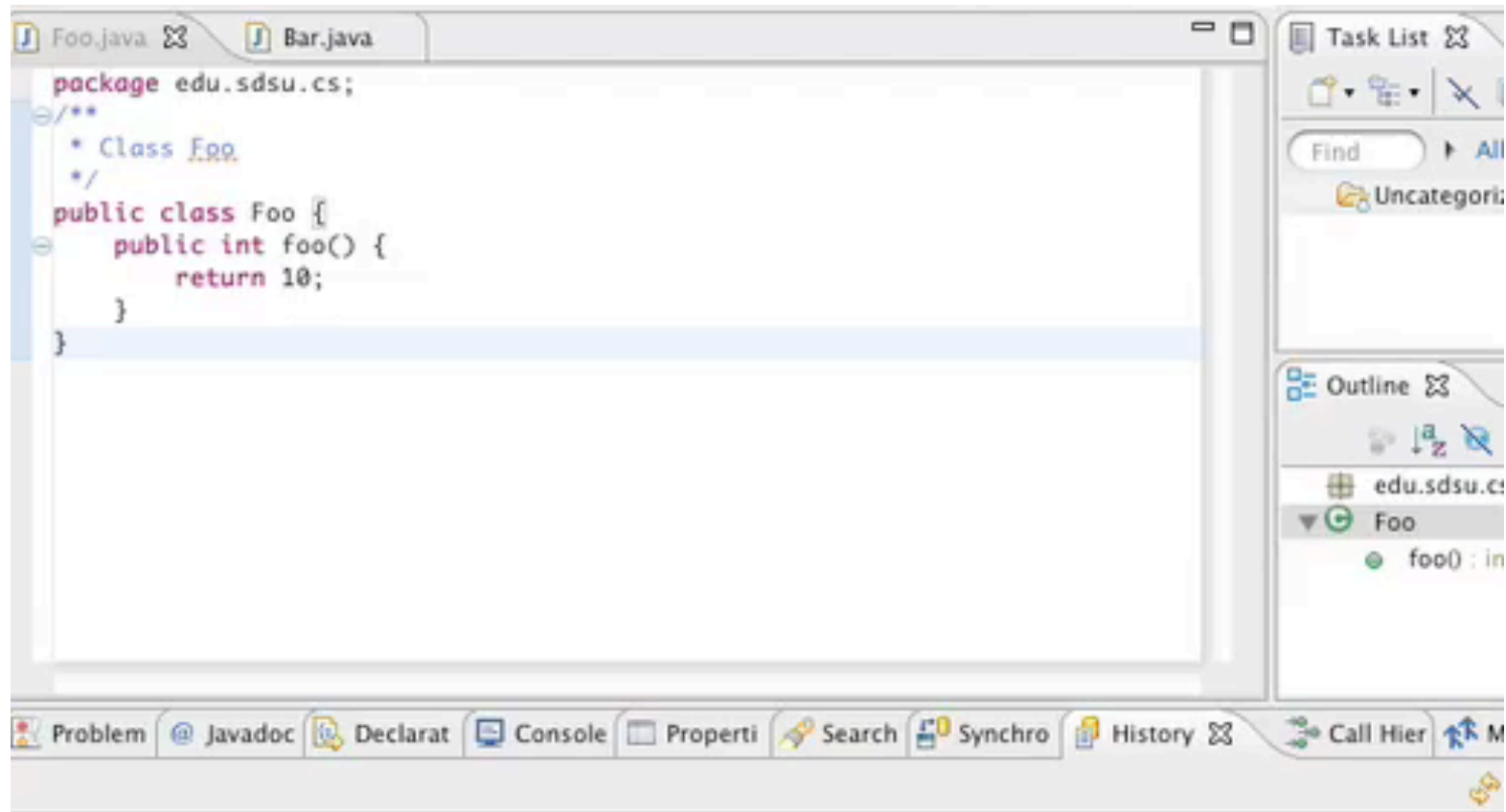
```
public class NewFoo {
        public int foo() {
                return 10;
        }
}
```

```
public class Bar {
    public int bar() {
        Foo test = new Foo();
        return test.foo() + 99;
    }
}
```

```
public class Bar {
    public int bar() {
        NewFoo test = new NewFoo();
        return test.foo() + 99;
    }
}
```

Eclipse Rename


```
Foo.java    Bar.java

package edu.sdsu.cs;
/**
 * Class Foo
 */
public class Foo {
    public int foo() {
        return 10;
    }
}
```

Task List

Find   ▶ All

Uncategori

Outline

edu.sdsu.cs
Foo
foo() : in

Problem Javadoc Declarat Console Properti Search Synchro History Call Hier M

Writable   Smart Insert   9 : 2

Move

```java
public class Bar {
    public int helperMethod(Foo test) {
        return test.foo() + test.fooTwo();
    }

    public int callHelper() {
        Foo data = new Foo();
        return helperMethod(data);
    }
}
```

```java
public class Foo {
    public int foo() { return 10;}

    public int fooTwo() { return 20; }
}
```
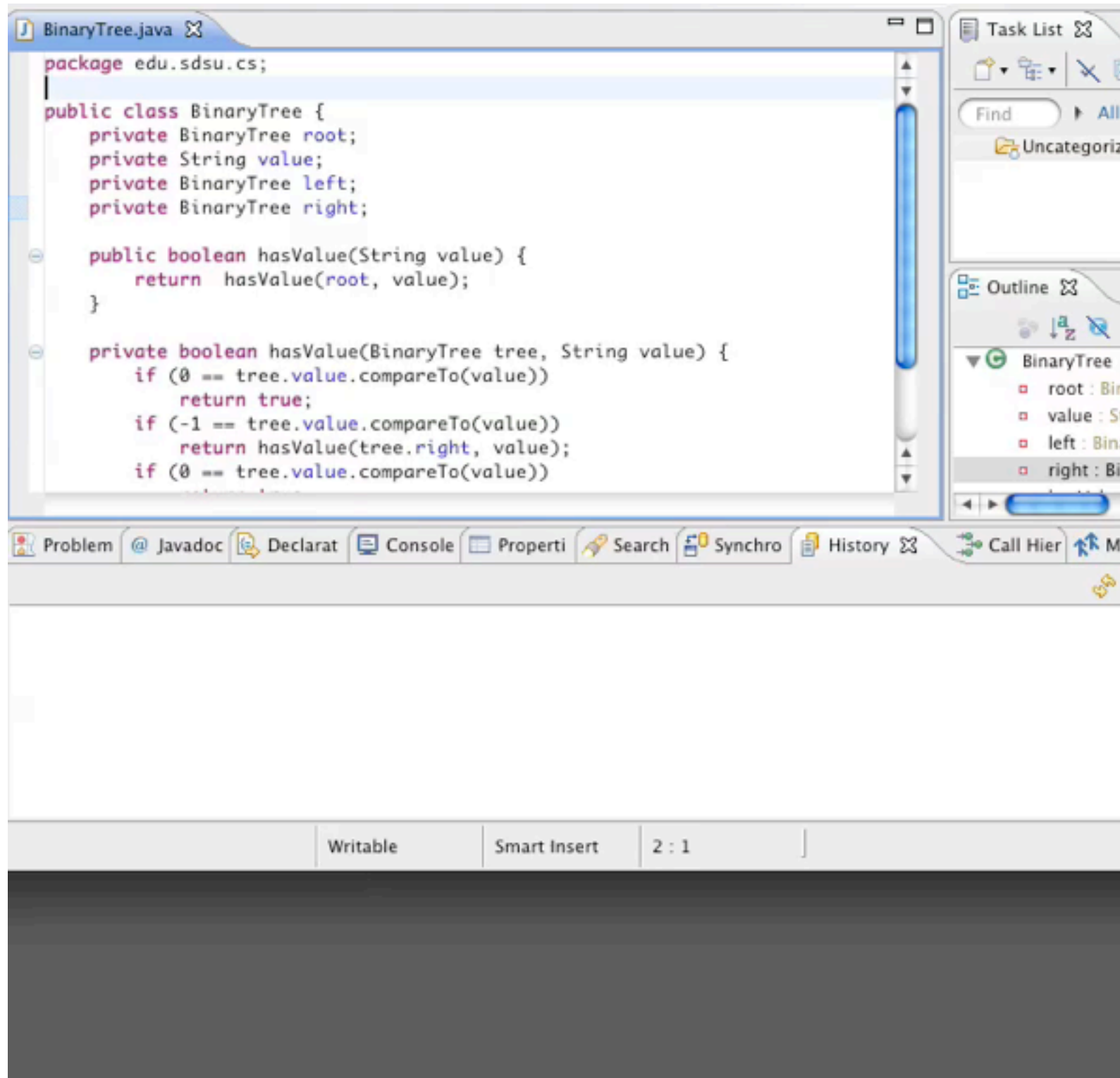
```java
public class Bar {
    public int callHelper() {
        Foo data = new Foo();
        return data.sum();
    }
}
```

```java
public class Foo {
    public int foo() { return 10;}

    public int fooTwo() {return 20; }

    public int sum() {
        return foo() + fooTwo();
    }
}
```

43

Eclipse Move
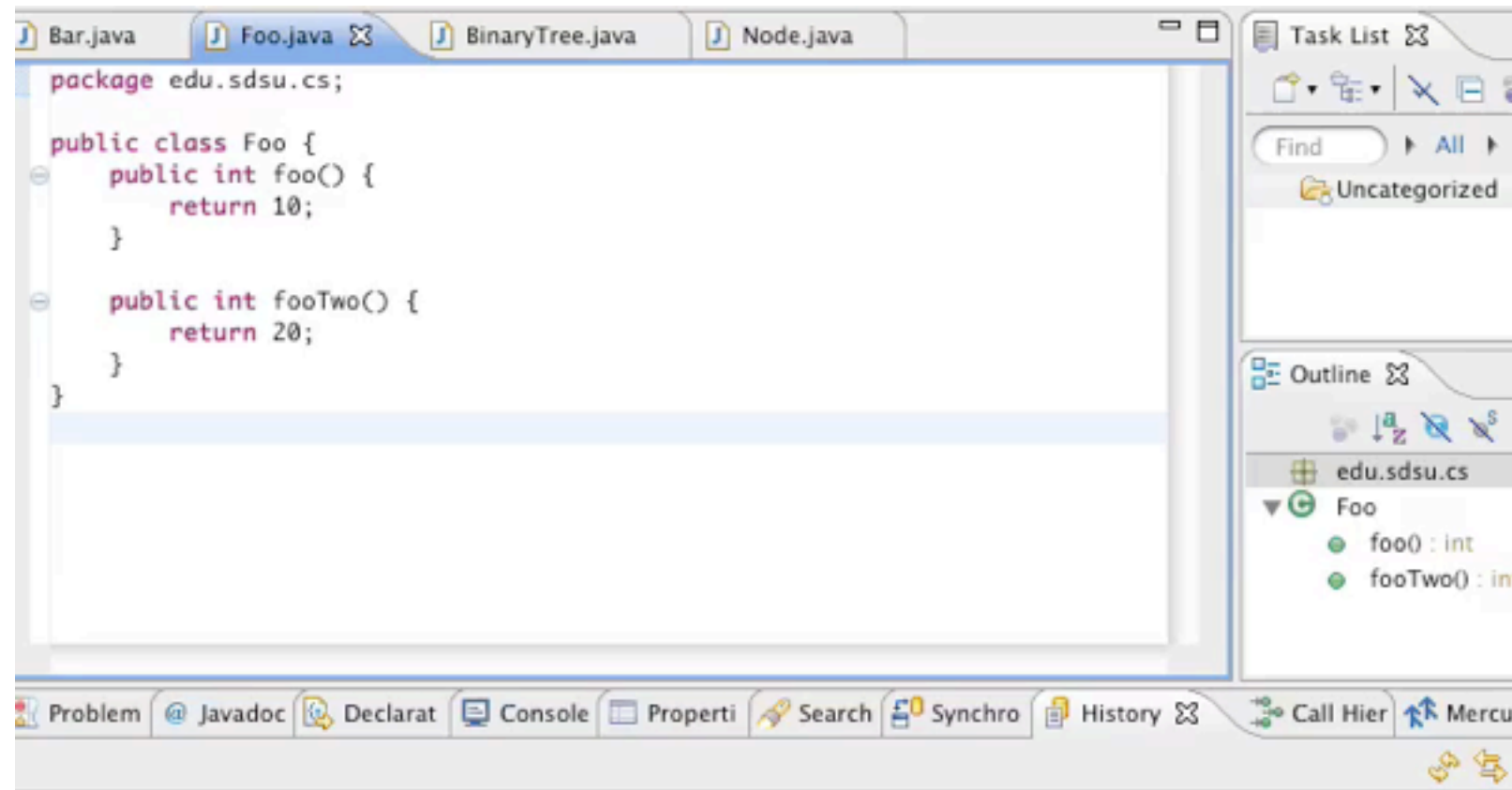
Extract Class

Refactoring Tool Issue

People tend to only use the features they know

Refactoring Tool Issue

Is a tool hard to use because I am unfamiliar with it or is it just hard to use

Refactoring by 41 Professional Programmers

	Number of Programmers used Refactoring	Total Times used
IntroduceFactory	1	1
PushDown	1	1
UseSupertype	1	6
EncapsulateField	2	5
Introduce Parameter	3	25
Convert Local to Field	5	37
Extract Interface	10	26
Inline	11	185
Modify Parameters	11	79
Pull up	11	37
Extract Method	20	344
Move	24	212
Rename	41	2396

Try In Eclipse

Rename

Move

Encapsulate Field

Extract Method

Extract Class

Unit Testing

Tuesday, February 3, 15

Testing

Johnson's Law

If it is not tested it does not work

The more time between coding and testing

    More effort is needed to write tests
    More effort is needed to find bugs
    Fewer bugs are found
    Time is wasted working with buggy code
    Development time increases
    Quality decreases

Unit Testing

Tests individual code segments

Automated tests

What wrong with:

Using print statements

Writing driver program in main

Writing small sample programs to run code

Running program and testing it be using it

Tuesday, February 3, 15

We have a QA Team, so why should I write tests?

When to Write Tests

First write the tests

Then write the code to be tested

Writing tests first saves time

Makes you clear of the interface & functionality of the code

Removes temptation to skip tests

What to Test

Everything that could possibly break

Test values
   Inside valid range
   Outside valid range
   On the boundary between valid/invalid

GUIs are very hard to test
   Keep GUI layer very thin
   Unit test program behind the GUI, not the GUI

Common Things Programs Handle Incorrectly

Adapted with permission from "A Short Catalog of
Test Ideas" by Brian Marick,

http://www.testing.com/writings.html

Strings

Empty String

Collections

Empty Collection

Collection with one element

Collection with duplicate elements

Collections with maximum possible size

Numbers

Zero

The smallest number

Just below the smallest number

The largest number

Just above the largest number

XUnit

Free frameworks for Unit testing

SUnit originally written by Kent Beck 1994

JUnit written by Kent Beck & Erich Gamma

Available at: http://www.junit.org/

Ports to many languages at:
    http://www.xprogramming.com/software.htm

XUnit Versions

3.x

Old version

Works with a versions of Java

4.x

Current version 4.8.1

Uses Annotations

Requires Java 5 or later

Simple Class to Test

```java
public class Adder {
    private int base;
    public Adder(int value) {
        base = value;
    }

    public int add(int amount) {
        return base + amount;
    }
}
```

Creating Test Case in Eclipse
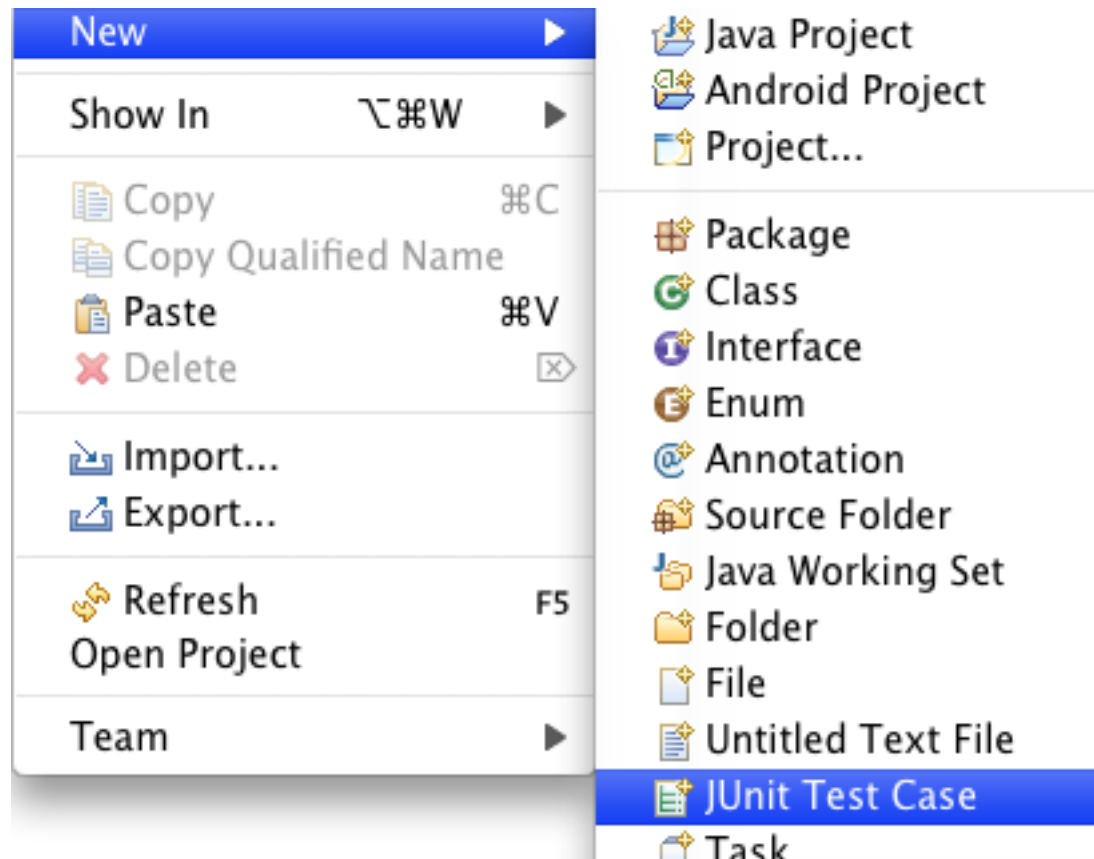
Creating Test Case in Eclipse

Fill in dialog window & create the test cases

Test Class

```java
import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertTrue;
import org.junit.Test;

public class TestAdder {

    @Test
    public void testAdd() {
        Adder example = new Adder(3);
        assertEquals(4, example.add(1));
    }

    @Test
    public void testAddFail() {
        Adder example = new Adder(3);
        assertTrue(3 == example.add(1));
    }
}
```
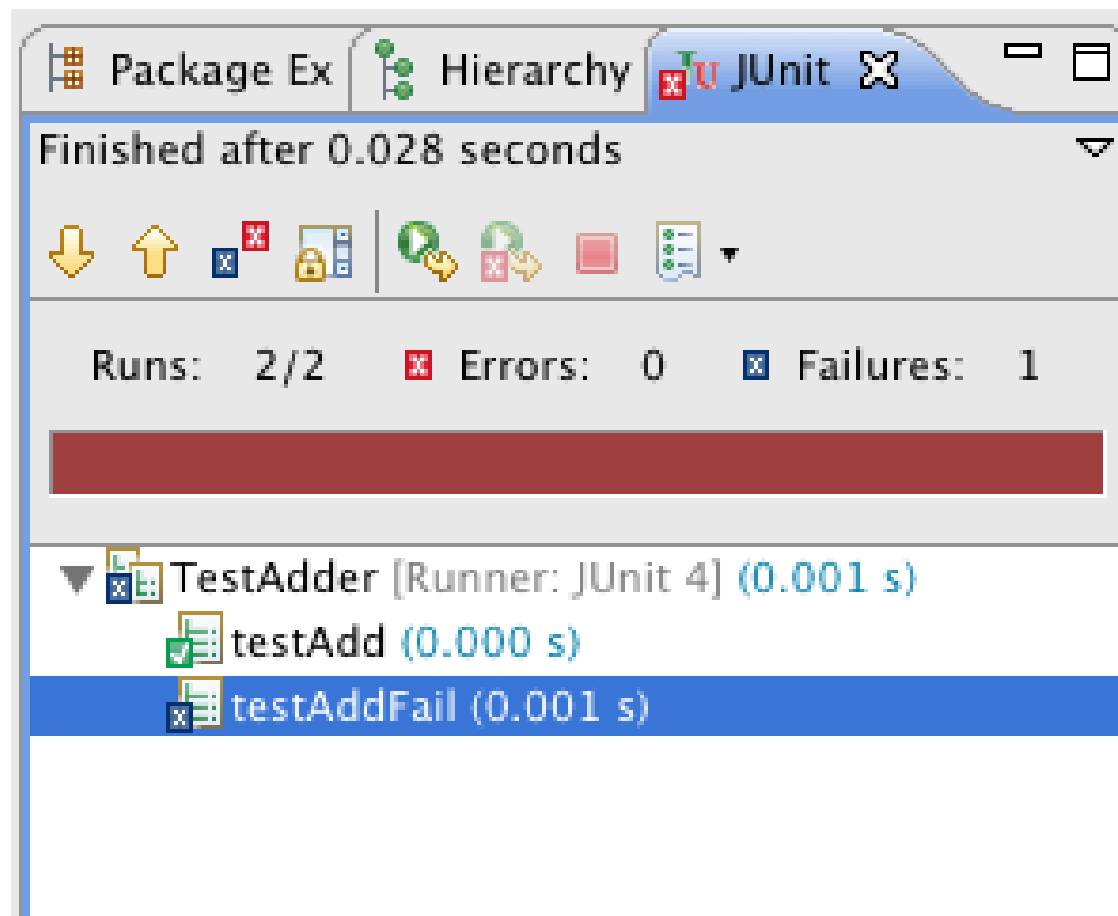
63

Running the Tests

The result

Assert Methods

assertArrayEquals()

assertTrue()

assertFalse()

assertEquals()

assertNotEquals()

assertSame()

assertNotSame()

assertNull()

assertNotNull()

fail()

For a complete list see http://kentbeck.github.com/junit/javadoc/latest/

Annotations

After

AfterClass

Before

BeforeClass

Ignore

Rule

Test

Using Before

```
import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertTrue;

import org.junit.Before;
import org.junit.Test;

public class TestAdder {
    Adder example;
    @Before
    public void setupExample() {
        example = new Adder(3);
    }

    @Test
    public void testAdd() {
        assertEquals(4, example.add(1));
    }
}
```