

An Introduction into JUnit

Praxis der Software-Entwicklung 2010/11

Daniel Bruns Erik Burger | January 18, 2011

INSTITUT FÜR THEORETISCHE INFORMATIK – INSTITUT FÜR PROGRAMMSTRUKTUREN UND DATENORGANISATION



KIT – University of the State of Baden-Wuerttemberg and National Research Center of the Helmholtz Association

www.kit.edu

イロト イヨト イミト イヨト 一日 - のへの





Program testing can be used to show the presence of bugs, but never to show their absence!

Dijkstra, 1972

Testing 0000000 Daniel Bruns, Erik Burger – JUnit <ロ> < 部> < E> < E> E のへで



Functional Tests

- Correctness according to specification
- Concurrency/Thread safeness

Non-Functional

- Performance
- Security
- Usability
- Interoperability
- Reliability



Functional Tests

- Correctness according to specification
- Concurrency/Thread safeness

Non-Functional

- Performance
- Security
- Usability
- Interoperability
- Reliability

▲ロ > ▲母 > ▲目 > ▲目 > ▲目 > ● ●



Functional Tests

- Correctness according to specification
- Concurrency/Thread safeness

Non-Functional

- Performance
- Security
- Usability
- Interoperability
- Reliability

< □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □



Functional Tests

- Correctness according to specification
- Concurrency/Thread safeness

Non-Functional

- Performance
- Security
- Usability
- Interoperability
- Reliability

▲ロ > ▲母 > ▲目 > ▲目 > ▲目 > ● ●



Functional Tests

- Correctness according to specification
- Concurrency/Thread safeness

Non-Functional

- Performance
- Security
- Usability
- Interoperability
- Reliability

▲ロ > ▲母 > ▲目 > ▲目 > ▲目 > ● ●



Functional Tests

- Correctness according to specification
- Concurrency/Thread safeness

Non-Functional

- Performance
- Security
- Usability
- Interoperability
- Reliability

▲ロ > ▲母 > ▲目 > ▲目 > ▲目 > ● ●



Functional Tests

- Correctness according to specification
- Concurrency/Thread safeness

Non-Functional

- Performance
- Security
- Usability
- Interoperability
- Reliability



Knowledge

- black-box tests
- white-box tests

Structure

- Unit
- Integration
- System

<ロト < 回 > < 三 > < 三 > < 三 > < 三 > < < つ < </p>



Knowledge

- black-box tests
- white-box tests

Structure

- Unit
- Integration
- System

<ロ> < 団> < 巨> < 巨> < 巨> < 巨</p>



Knowledge

- black-box tests
- white-box tests

Structure

Unit

- Integration
- System



Knowledge

- black-box tests
- white-box tests

Structure

- Unit
- Integration
- System

<ロ> < 団> < 豆> < 豆> < 豆> < 豆> < 豆</p>



Knowledge

- black-box tests
- white-box tests

Structure

- Unit
- Integration
- System

<ロ> < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □



Object oriented classes often have dependencies on other classes

- A lot of classes cannot be tested independently
- lacksquare ightarrow micro integration tests
- Starting from a certain degree of dependencies, test effort rises disproportionately high



Object oriented classes often have dependencies on other classes

- A lot of classes cannot be tested independently
- lacksquare ightarrow micro integration tests
- Starting from a certain degree of dependencies, test effort rises disproportionately high



- Object oriented classes often have dependencies on other classes
- A lot of classes cannot be tested independently
- lacksim ightarrow micro integration tests
- Starting from a certain degree of dependencies, test effort rises disproportionately high



- Object oriented classes often have dependencies on other classes
- A lot of classes cannot be tested independently
- ightarrow ightarrow micro integration tests
- Starting from a certain degree of dependencies, test effort rises disproportionately high



Test represent typical usage scenarios

- Less dependencies → easier to use
- High degree of dependencies
 - Lack of modularisation?
 - Bad design?
 - Bad code dependency management
- lacksquare \rightarrow Refactoring

▲ロ > ▲母 > ▲目 > ▲目 > ▲目 > ● ●

Testing ○○●○○○○ Daniel Bruns, Erik Burger – JUnit



- Test represent typical usage scenarios
- Less dependencies \rightarrow easier to use
- High degree of dependencies
 - Lack of modularisation?
 - Bad design?
 - Bad code dependency management
- lacksquare \rightarrow Refactoring

▲ロ > ▲母 > ▲目 > ▲目 > ▲目 > ● ●

Testing ○○●○○○○ Daniel Bruns, Erik Burger – JUnit



- Test represent typical usage scenarios
- Less dependencies \rightarrow easier to use
- High degree of dependencies
 - Lack of modularisation?
 - Bad design?
 - Bad code dependency management
- lacksquare \rightarrow Refactoring



- Test represent typical usage scenarios
- Less dependencies \rightarrow easier to use
- High degree of dependencies
 - Lack of modularisation?
 - Bad design?
 - Bad code dependency management
- lacksquare \rightarrow Refactoring

▲□▶ ▲□▶ ▲臣▶ ▲臣▶ 三臣 - のへで



- Test represent typical usage scenarios
- Less dependencies \rightarrow easier to use
- High degree of dependencies
 - Lack of modularisation?
 - Bad design?
 - Bad code dependency management
- lacksquare \rightarrow Refactoring

▲□▶ ▲□▶ ▲目▶ ▲目▶ ▲目▶ ▲□



- Test represent typical usage scenarios
- Less dependencies \rightarrow easier to use
- High degree of dependencies
 - Lack of modularisation?
 - Bad design?
 - Bad code dependency management

ightarrow ightarrow Refactoring

Testing 00000000 Daniel Bruns, Erik Burger – JUnit



- Test represent typical usage scenarios
- Less dependencies \rightarrow easier to use
- High degree of dependencies
 - Lack of modularisation?
 - Bad design?
 - Bad code dependency management
- lacksquare \to Refactoring



Types of coverage

- Statement coverage (Anweisungsüberdeckung)
- Branch coverage (Zweigüberdeckung)
- Path coverage (Pfadüberdeckung)

<ロ> < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □



Types of coverage

Statement coverage (Anweisungsüberdeckung)

- Branch coverage (Zweigüberdeckung)
- Path coverage (Pfadüberdeckung)

<ロ> < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □



Types of coverage

- Statement coverage (Anweisungsüberdeckung)
- Branch coverage (Zweigüberdeckung)
- Path coverage (Pfadüberdeckung)



Types of coverage

- Statement coverage (Anweisungsüberdeckung)
- Branch coverage (Zweigüberdeckung)
- Path coverage (Pfadüberdeckung)

Example Method



```
public int foo (int x, int y) {
    int z = 0;
    if (y != 0) {
        while (x != 0) {
             if (x > 0) {
                 z += y;
                 x - - ;
             } else {
                 z -= y;
                 x++;
             }
        }
    }
    return z;
}
```





Testing 0000000 Daniel Bruns, Erik Burger – JUnit





Testing 0000000 Daniel Bruns, Erik Burger – JUnit





Testing 0000000 Daniel Bruns, Erik Burger – JUnit





<ロ> < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □

Testing 0000000 Daniel Bruns, Erik Burger – JUnit





<ロ> < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □

Testing 0000000 Daniel Bruns, Erik Burger – JUnit





<ロ> < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □
Control Flow Graph





Testing 0000000 Daniel Bruns, Erik Burger – JUnit

January 18, 2011 9/44

Control Flow Graph





Testing 0000000 Daniel Bruns, Erik Burger – JUnit

January 18, 2011 9/44



Equivalence classes

- Assumption: similar control flow for similar values
- Last example: 3 test needed for full branch coverage
- Equivalence classes:

Extreme values

- Variant of equivalence classes approach
- "Off-by-one" most prominent error
- Extreme values for integers: MIN_VALUE, -1, 0, 1, MAX_VALUE, someArray.length
- Extreme values for objects: null, empty strings, empty collections

▲□▶ ▲□▶ ▲目▶ ▲目▶ ▲目▶ ▲□

Testing OCOCO Daniel Bruns, Erik Burger – JUnit



Equivalence classes

- Assumption: similar control flow for similar values
- Last example: 3 test needed for full branch coverage
- Equivalence classes: $\{\{(x, y) \mid y = 0\}, \{(x, y) \mid y \neq 0 \land x > 0\}, \{(x, y) \mid y \neq 0 \land x \le 0\}\}$

Extreme values

- Variant of equivalence classes approach
- "Off-by-one" most prominent error
- Extreme values for integers: MIN_VALUE, -1, 0, 1, MAX_VALUE, someArray.length
- Extreme values for objects: null, empty strings, empty collections



Equivalence classes

- Assumption: similar control flow for similar values
- Last example: 3 test needed for full branch coverage
- Equivalence classes: {(0,0), (23,42), (-23,666)}

Extreme values

- Variant of equivalence classes approach
- "Off-by-one" most prominent error
- Extreme values for integers: MIN_VALUE, -1, 0, 1, MAX_VALUE, someArray.length
- Extreme values for objects: null, empty strings, empty collections

ペロト < 団 > < 三 > < 三 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < 回 > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □



Equivalence classes

- Assumption: similar control flow for similar values
- Last example: 3 test needed for full branch coverage
- Equivalence classes: {(0,0), (23,42), (-23,666)}

Extreme values

- Variant of equivalence classes approach
- "Off-by-one" most prominent error
- Extreme values for integers: MIN_VALUE, -1, 0, 1, MAX_VALUE, someArray.length
- Extreme values for objects: null, empty strings, empty collections

▲□▶▲□▶▲□▶▲□▶▲□▶▲□



Equivalence classes

- Assumption: similar control flow for similar values
- Last example: 3 test needed for full branch coverage
- Equivalence classes: {(0,0), (23, 42), (-23, 666)}

Extreme values

- Variant of equivalence classes approach
- "Off-by-one" most prominent error
- Extreme values for integers: MIN_VALUE, -1, 0, 1, MAX_VALUE, someArray.length
- Extreme values for objects: null, empty strings, empty collections

▲□▶▲□▶▲□▶▲□▶▲□▼



Equivalence classes

- Assumption: similar control flow for similar values
- Last example: 3 test needed for full branch coverage
- Equivalence classes: {(0,0), (23,42), (-23,666)}

Extreme values

- Variant of equivalence classes approach
- "Off-by-one" most prominent error
- Extreme values for integers: MIN_VALUE, -1, 0, 1, MAX_VALUE, someArray.length
- Extreme values for objects: null, empty strings, empty collections





Assertions



Fixtures

- Definition
- Example
- Parameterised Tests
- Test Suites



Eclipse Integration

Test Runners

Overview

Assertions

- イロト イヨト イヨト イヨト ヨー のへぐ

Fixtures

Eclipse Integration

Daniel Bruns, Erik Burger - JUnit

January 18, 2011 11/44



JUnit is a framework for writing tests

- JUnit uses Java's reflection capabilities (Java programs can examine their own code)
- JUnit helps the programmer:
 - define and execute tests and test suites
 - formalize requirements and clarify architecture
 - write and debug code
 - integrate code and always be ready to release a working version

JUnit is not included in Sun's SDK, but almost all IDEs include it (e.g. Eclipse)

Overview •0000000 Daniel Bruns, Erik Burger – JUnit Assertions

シック・ 山 (山下)(山下)(山下)(山下)

Fixtures

Eclipse Integration

January 18, 2011



JUnit is a framework for writing tests

- JUnit uses Java's reflection capabilities (Java programs can examine their own code)
- JUnit helps the programmer:
 - define and execute tests and test suites

Assertions

- formalize requirements and clarify architecture
- write and debug code
- integrate code and always be ready to release a working version
- JUnit is not included in Sun's SDK, but almost all IDEs include it (e.g. Eclipse)

Overview •0000000 Daniel Bruns, Erik Burger – JUnit A D P A B P A B P B P A C P P A C

Fixtures	Eclipse Integ
000000000000000000000000000000000000000	00

ration



JUnit is a framework for writing tests

- JUnit uses Java's reflection capabilities (Java programs can examine their own code)
- JUnit helps the programmer:
 - define and execute tests and test suites
 - formalize requirements and clarify architecture
 - write and debug code
 - integrate code and always be ready to release a working version
- JUnit is not included in Sun's SDK, but almost all IDEs include it (e.g. Eclipse)

Overview •0000000 Daniel Bruns, Erik Burger – JUnit Assertions 000 うせん 聞い ふぼう ふぼう ふしゃ

Fixtures		Eclipse	Integration
000000000000000000000000000000000000000	0	00	



JUnit is a framework for writing tests

- JUnit uses Java's reflection capabilities (Java programs can examine their own code)
- JUnit helps the programmer:
 - define and execute tests and test suites
 - formalize requirements and clarify architecture
 - write and debug code
 - integrate code and always be ready to release a working version
- JUnit is not included in Sun's SDK, but almost all IDEs include it (e.g. Eclipse)

Overview •0000000 Daniel Bruns, Erik Burger – JUnit Assertions

シック・ 山 (山下)(山下)(山下)(山下)

Fixtures Eclip

Eclipse Integration



JUnit is a framework for writing tests

- JUnit uses Java's reflection capabilities (Java programs can examine their own code)
- JUnit helps the programmer:
 - define and execute tests and test suites
 - formalize requirements and clarify architecture
 - write and debug code

Overview •••••••• Daniel Bruns, Erik Burger - JUnit Assertions

 < □ > < □ > < □ > < □ > < □ > < 3 DQ C

Fixtures	Eclipse Integratio	n
0000000000000	00	
January 18	8.2011 12/44	4



JUnit is a framework for writing tests

- JUnit uses Java's reflection capabilities (Java programs can examine their own code)
- JUnit helps the programmer:
 - define and execute tests and test suites
 - formalize requirements and clarify architecture
 - write and debug code
 - integrate code and always be ready to release a working version

JUnit is not included in Sun's SDK, but almost all IDEs include it (e.g. Eclipse)

Overview •0000000 Daniel Bruns, Erik Burger – JUnit Assertions 000 - 《日》《母》《臣》《臣》 臣 のへの

Fixtures	E
000000000000000000000000000000000000000	0

clipse Integration



JUnit is a framework for writing tests

- JUnit uses Java's reflection capabilities (Java programs can examine their own code)
- JUnit helps the programmer:
 - define and execute tests and test suites
 - formalize requirements and clarify architecture
 - write and debug code
 - integrate code and always be ready to release a working version
- JUnit is not included in Sun's SDK, but almost all IDEs include it (e.g. Eclipse)

				୶ୣ
Overview	Assertions	Fixtures	Eclipse Inte	gration
●00000000	000	0000000000000	00	
Daniel Bruns, Erik Burger - JUnit		January 1	8, 2011	12/44



History

 JUnit was written by Erich Gamma (of Design Patterns) and Kent Beck (creator of XP methodology)

 JUnit inspired various other unit testing frameworks for other programming languages, like NUnit (.NET), CppUnit(C++)

JUnit is the de facto standard for test driven Java development

Overview 0000000 Daniel Bruns, Erik Burger – JUnit Assertions

- * ロ > * 母 > * 注 > * 注 > ・ 注 ・ の < @

Fixtures

Eclipse Integration

January 18, 2011



History

- JUnit was written by Erich Gamma (of Design Patterns) and Kent Beck (creator of XP methodology)
- JUnit inspired various other unit testing frameworks for other programming languages, like NUnit (.NET), CppUnit(C++)

JUnit is the de facto standard for test driven Java development

Overview 0000000 Daniel Bruns, Erik Burger – JUnit Assertions

- < ロ > < 回 > < 臣 > < 臣 > 三 - の < @

Fixtures

Eclipse Integration

January 18, 2011



History

- JUnit was written by Erich Gamma (of Design Patterns) and Kent Beck (creator of XP methodology)
- JUnit inspired various other unit testing frameworks for other programming languages, like NUnit (.NET), CppUnit(C++)
- JUnit is the de facto standard for test driven Java development

Overview oeoooooo Daniel Bruns, Erik Burger – JUnit Assertions

- < ロ > < 回 > < 臣 > < 臣 > 三 - の < @

Fixtures

Eclipse Integration



JUnit4

- JUnit4 was a complete redevelopment

Overview 000000000 Daniel Bruns, Erik Burger - JUnit Assertions

 < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ nac

Fixtures	Eclipse Integr	ration
000000000000000000	00	
January	18.2011 1	4/44



JUnit4

- JUnit4 was a complete redevelopment
- includes ideas from other frameworks and uses features of Java 1.5

Overview 000000000 Daniel Bruns, Erik Burger - JUnit Assertions

3 nac

ixtures	Eclipse Inte	gration
000000000000000000000000000000000000000	00	
Januarv	18. 2011	14/44



JUnit4

- JUnit4 was a complete redevelopment
- includes ideas from other frameworks and uses features of Java 1.5
- uses Java annotations (like @Test)

Overview 000000000 Daniel Bruns, Erik Burger - JUnit Assertions

< ロ > < 回 > < 回 > < 回 > < 回 > < 1 nac

Fixtures	Eclipse Integr	ration
000000000000000000	00	
January	18.2011 1	4/44



JUnit4

- JUnit4 was a complete redevelopment
- includes ideas from other frameworks and uses features of Java 1.5
- uses Java annotations (like @Test)
- This lecture is based on JUnit 4

Overview 000000000 Daniel Bruns, Erik Burger - JUnit Assertions

3 nac

Fixtures **Eclipse Integration** January 18, 2011



JUnit4

- JUnit4 was a complete redevelopment
- includes ideas from other frameworks and uses features of Java 1.5
- uses Java annotations (like @Test)
- This lecture is based on JUnit 4

Be careful

- Many (web) tutorials are still based on JUnit 3
- JUnit 4 is backwards compatible to version 3
- but JUnit 4 is much cleaner

Overview 00000000 Daniel Bruns, Erik Burger – JUnit Assertions 000 Fixtures

Eclipse Integration

January 18, 2011



JUnit4

- JUnit4 was a complete redevelopment
- includes ideas from other frameworks and uses features of Java 1.5
- uses Java annotations (like @Test)
- This lecture is based on JUnit 4

Be careful

- Many (web) tutorials are still based on JUnit 3
- JUnit 4 is backwards compatible to version 3
- but JUnit 4 is much cleaner

Overview 0000000 Daniel Bruns, Erik Burger – JUnit Assertions 000 ・ロト・西ト・ヨト・ヨー うへの

Fixtures

Eclipse Integration

January 18, 2011



JUnit4

- JUnit4 was a complete redevelopment
- includes ideas from other frameworks and uses features of Java 1.5
- uses Java annotations (like @Test)
- This lecture is based on JUnit 4

Be careful

- Many (web) tutorials are still based on JUnit 3
- JUnit 4 is backwards compatible to version 3
- but JUnit 4 is much cleaner

Overview 000000000 Daniel Bruns, Erik Burger - JUnit Assertions

Eclipse Integration

3

January 18, 2011

<ロト < 同ト < ヨト < ヨト -

14/44

DQ C



Karbruhe Institute of Technology

- A unit test tests the methods in a single class
- A test case tests (insofar as possible) a single method
- You can have multiple test cases for a single method
- A test suite combines unit tests
- The test fixture provides software support for all this
- The test runner runs unit tests or an entire test suite

Overview 00000000 Daniel Bruns, Erik Burger – JUnit Assertions 000 Fixtures Eclipse Integration

イロト イポト イヨト イヨト

January 18, 2011

15/44

nar



Karlsruhe Institute of Technology

- A unit test tests the methods in a single class
- A test case tests (insofar as possible) a single method
- You can have multiple test cases for a single method
- A test suite combines unit tests
- The test fixture provides software support for all this
- The test runner runs unit tests or an entire test suite

Overview 00000000 Daniel Bruns, Erik Burger – JUnit Assertions 000 Fixtures Eclipse Integration

イロト イポト イヨト イヨト

15/44

nar



test runner





test fixture

- A unit test tests the methods in a single class
- A test case tests (insofar as possible) a single method
- You can have multiple test cases for a single method
- A test suite combines unit tests
- The test fixture provides software support for all this
- The test runner runs unit tests or an entire test suite

Overview 00000000 Daniel Bruns, Erik Burger – JUnit Assertions 000 Fixtures Eclipse Integration

<ロト < 同ト < ヨト < ヨト -

15/44

nar



test fixture



= nar

- A unit test tests the methods in a single class
- A test case tests (insofar as possible) a single method
- You can have multiple test cases for a single method
- A test suite combines unit tests
- The test fixture provides software support for all this
- The test runner runs unit tests or an entire test suite

Overview 00000000 Daniel Bruns, Erik Burger – JUnit Assertions

Fixtures Eclipse Integration

<ロト < 同ト < ヨト < ヨト -



test suite



- A unit test tests the methods in a single class
- A test case tests (insofar as possible) a single method
- You can have multiple test cases for a single method
- A test suite combines unit tests

Overview 000000000 Daniel Bruns, Erik Burger - JUnit Assertions

Fixtures Eclipse Integration

15/44

3 nac



nac 3



- A unit test tests the methods in a single class
- A test case tests (insofar as possible) a single method
- You can have multiple test cases for a single method
- A test suite combines unit tests
- The test fixture provides software support for all this

Overview 000000000 Daniel Bruns, Erik Burger - JUnit Assertions

イロト イポト イヨト イヨト Fixtures Eclipse Integration 15/44 January 18, 2011



test runner test suite unit test (for one class) test case (for one method) another test case another unit test another test case another test case another test case unit test (for one class) test case (for one method) another test case test fixture

- A unit test tests the methods in a single class
- A test case tests (insofar as possible) a single method
- You can have multiple test cases for a single method
- A test suite combines unit tests
- The test fixture provides software support for all this
- The test runner runs unit tests or an entire test suite

Overview 000000000 Daniel Bruns, Erik Burger - JUnit Assertions

Fixtures Eclipse Integration January 18, 2011

15/44

nac 1

Test Case Verdicts



A verdict is the declared result of executing a single test.

- Pass: the test case achieved its intended purpose, and the software under test performed as expected.
- Fail: the test case achieved its intended purpose, but the software under test did not perform as expected.
- Error: the test case did not achieve its intended purpose. Potential reasons:
 - An unexpected event occurred during the test case.
 - The test case could not be set up properly

Overview 00000000 Daniel Bruns, Erik Burger – JUnit Assertions

- 《日》《母》《臣》《臣》 臣 のへの

Fixtures	Eclipse I	ntegration
000000000000000000000000000000000000000	00	
Januarv	18, 2011	16/44

Test Case Verdicts



- A verdict is the declared result of executing a single test.
- Pass: the test case achieved its intended purpose, and the software under test performed as expected.
- Error: the test case did not achieve its intended purpose.

Overview 000000000 Daniel Bruns, Erik Burger - JUnit Assertions

イロト 人間ト イヨト イヨト DQ C

Fixtures	Eclipse I	ntegratio
000000000000000000000000000000000000000	00	
Januarv	18, 2011	16/44

Test Case Verdicts



- A verdict is the declared result of executing a single test.
- Pass: the test case achieved its intended purpose, and the software under test performed as expected.
- Fail: the test case achieved its intended purpose, but the software under test did not perform as expected.
- Error: the test case did not achieve its intended purpose. Potential reasons:

An unexpected event occurred during the test case.

The test case could not be set up properly

Overview 00000000 Daniel Bruns, Erik Burger – JUnit Assertions

シック・ 川 ・ (川・・川・・ (四・・日・)

Fixtures E

Eclipse Integration

January 18, 2011
Test Case Verdicts



- A verdict is the declared result of executing a single test.
- Pass: the test case achieved its intended purpose, and the software under test performed as expected.
- Fail: the test case achieved its intended purpose, but the software under test did not perform as expected.
- Error: the test case did not achieve its intended purpose. Potential reasons:

An unexpected event occurred during the test case.

The test case could not be set up properly

Overview 000000000 Daniel Bruns, Erik Burger - JUnit Assertions

イロト イポト イヨト イヨト DQ (P

Fixtures **Eclipse Integration** January 18, 2011

Test Case Verdicts



- A verdict is the declared result of executing a single test.
- Pass: the test case achieved its intended purpose, and the software under test performed as expected.
- Fail: the test case achieved its intended purpose, but the software under test did not perform as expected.
- Error: the test case did not achieve its intended purpose. Potential reasons:
 - An unexpected event occurred during the test case.
 - The test case could not be set up properly

Overview 00000000 Daniel Bruns, Erik Burger – JUnit Assertions 000 - 《ロ》 《聞》 《臣》 《臣》 三目 のへの

Fixtures 0000000000000 Eclipse Integration

January 18, 2011

Test Case Verdicts



- A verdict is the declared result of executing a single test.
- Pass: the test case achieved its intended purpose, and the software under test performed as expected.
- Fail: the test case achieved its intended purpose, but the software under test did not perform as expected.
- Error: the test case did not achieve its intended purpose. Potential reasons:
 - An unexpected event occurred during the test case.
 - The test case could not be set up properly

イロト イポト イヨト イヨト

DQ C



A test "script" is just a collection of Java methods.

General idea is to create a few Java objects, do something interesting with them, and then determine if the objects have the correct properties.

What is added? Assertions.

• A package of methods that checks for various properties:

- "equality" of objects
- identical object references
- null / non-null object references

Assertions

The assertions are used to determine the test case verdict.

Overview 00000000 Daniel Bruns, Erik Burger – JUnit うせん 川 ふかく ボット 白マ

Fixtures	Eclipse Ir	ntegrati
00000000000000	00	
Januarv	18.2011	17/4



A test "script" is just a collection of Java methods.

General idea is to create a few Java objects, do something interesting with them, and then determine if the objects have the correct properties.

What is added? Assertions.

• A package of methods that checks for various properties:

- "equality" of objects
- identical object references
- null / non-null object references
- The assertions are used to determine the test case verdict.

Overview	
00000000	
Daniel Bruns, Erik Burger - JUnit	

Assertions

Fixtures	
000000000000000000000000000000000000000	

Eclipse Integration

Organisation of JUnit Tests



- Each method represents a single test case that can independently have a verdict (pass, error, fail).
- Normally, all the tests for one Java class are grouped together into a separate class.
- Naming convention:
 - Class to be tested: Value
 - Class containing tests: ValueTest

Assertions

イロト イポト イヨト イヨト

DQ C

Writing a JUnit test class



Start by importing these JUnit 4 classes

```
import org.junit.*
import static org.junit.Assert.*; // note static import
```

Declare your test class in the usual way

```
public class MyProgramTest {
}
```

Declare an instance of the class being tested

```
public class MyProgramTest {
    MyProgram program;
    int someVariable;
}
```

Overview 00000000 Daniel Bruns, Erik Burger – JUnit Assertions

しって 川 ・ ・ 川 ・ ・ 雪 ・ トロッ

Fixtures

Eclipse Integration

A simple example



```
import org.junit.*;
1
   import static org.junit.Assert.*;
3
   public class ArithmeticTest {
4
       @Test
       public void testMultiply() {
           assertEquals(4, Arithmetic.multiply(2, 2));
7
           assertEquals(-15, Arithmetic.multiply(3, -5));
       }
       @Test
       public void testIsPositive() {
           assertTrue(Arithmetic.isPositive(5));
           assertFalse(Arithmetic.isPositive(-5));
14
           assertFalse(Arithmetic.isPositive(0));
       }
15
16
   }
                                        -
```

Overview

Assertions

Fixtures

Eclipse Integration



Assertions are defined in the JUnit class Assert

- If an assertion is true, the method continues executing.
- If any assertion is false, the method stops executing at that point, and the result for the test case will be fail.
- If any other exception is thrown during the method, the result for the test case will be error.
- If no assertions were violated for the entire method, the test case will pass.

All assertion methods are static methods.

Overview 00000000 Daniel Bruns, Erik Burger – JUnit Assertions •00 ・ 「 「 」 ・ 「 」 ・ (山) ・ 「 目) ・ (日) ・

Fixtures	Eclipse Integrat	io
0000000000000	00	
January ⁻	18.2011 21/-	44



Assertions are defined in the JUnit class Assert

- If an assertion is true, the method continues executing.
- If any assertion is false, the method stops executing at that point, and the result for the test case will be fail.
- If any other exception is thrown during the method, the result for the test case will be error.
- If no assertions were violated for the entire method, the test case will pass.

All assertion methods are static methods.

Overview 00000000 Daniel Bruns, Erik Burger – JUnit Assertions •00 うとの 叫 ふかく 山を (中) へしゃ

Fixtures	Eclipse Ir	ntegratior
00000000000000000	00	
January	18. 2011	21/44



Assertions are defined in the JUnit class Assert

- If an assertion is true, the method continues executing.
- If any assertion is false, the method stops executing at that point, and the result for the test case will be fail.
- If any other exception is thrown during the method, the result for the test case will be error.

All assertion methods are static methods.

Overview Daniel Bruns, Erik Burger - JUnit Assertions 000

イロト イポト イヨト イヨト DQ C



Assertions are defined in the JUnit class Assert

- If an assertion is true, the method continues executing.
- If any assertion is false, the method stops executing at that point, and the result for the test case will be fail.
- If any other exception is thrown during the method, the result for the test case will be error.
- If no assertions were violated for the entire method, the test case will pass.

All assertion methods are static methods.

Overview 00000000 Daniel Bruns, Erik Burger – JUnit Assertions •00 ▲日マート 4 回マート 1 回 2 0 0 0

Assertion Methods



Boolean conditions are true or false

```
assertTrue(condition)
assertFalse(condition)
```

Objects are null or non-null

```
assertNull(object)
assertNotNull(object)
```

Objects are identical (i.e. two references to the same object), or not identical.

```
assertSame(expected, actual)
assertNotSame(expected, actual)
```

Overview 00000000 Daniel Bruns, Erik Burger – JUnit Assertions

・ロト・(部ト・ミト・ミト) そののの Foliopea Integration Foliopea Integration

Fixtures

Eclipse Integration

January 18, 2011

Assertion Methods



"Equality" of objects

assertEquals(expected, actual)
valid if: expected.equals(actual)

"Equality" of arrays

assertArrayEquals(expected, actual)

- arrays must have same length
- for each valid value for i, check as appropriate:

assertEquals(expected[i],actual[i])

assertArrayEquals(expected[i],actual[i])

Assertions

There is also an unconditional failure assertion fail() that always results in a fail verdict.

Overview 00000000 Daniel Bruns, Erik Burger – JUnit

Fixtures	Eclipse Integration	on
0000000000000000	00	
January	18, 2011 23/4	4

Assertion Methods



"Equality" of objects

assertEquals(expected, actual)
valid if: expected.equals(actual)

"Equality" of arrays

```
assertArrayEquals(expected, actual)
```

- arrays must have same length
- for each valid value for i, check as appropriate:

```
assertEquals(expected[i],actual[i])
assertArrayEquals(expected[i],actual[i])
```

Assertions

000

```
There is also an unconditional failure assertion fail() that always results in a fail verdict.
```

Overview 00000000 Daniel Bruns, Erik Burger – JUnit < ロ > < 団 > < 三 > < 三 > < 三 < つ < へ

Fixtures	
000000000000000000000000000000000000000	

Eclipse Integration



Test Fixture

- A test fixture is the context in which a test case runs.
- Typically, test fixtures include:
 - Objects or resources that are available for use by any test case.
 Activities required to make these objects available and/or resource allocation and de-allocation: "setup" and "teardown".
- Allows multiple tests of the same or similar objects

Share fixture code for multiple tests

Overview 000000000 Assertions 000 うてん 聞 (木田)・木田)・木日)・

Daniel Bruns, Erik Burger - JUnit

January 18, 2011



Test Fixture

- A test fixture is the context in which a test case runs.
- Typically, test fixtures include:
 - Objects or resources that are available for use by any test case.
 - Activities required to make these objects available and/or resource allocation and de-allocation: "setup" and "teardown".
- Allows multiple tests of the same or similar objects

Share fixture code for multiple tests

Overview 000000000 Assertions 000 ・ ロマ・ 4 国 マ 4 国 マ 4 国 マ 9 0 0 0

Fixtures

Eclipse Integration

January 18, 2011

1 24/44

Daniel Bruns, Erik Burger – JUnit

Frik Burger – III Init

lions



Test Fixture

- A test fixture is the context in which a test case runs.
- Typically, test fixtures include:
 - Objects or resources that are available for use by any test case.
 - Activities required to make these objects available and/or resource allocation and de-allocation: "setup" and "teardown".
- Allows multiple tests of the same or similar objects

Share fixture code for multiple tests

Overview 000000000 Assertions 000 ・ 「 「 」 ・ 「 」 ・ (山) ・ (口) ・ (口) ・

Fixtures

Eclipse Integration

January 18, 2011

24/44

Daniel Bruns, Erik Burger – JUnit



Test Fixture

- A test fixture is the context in which a test case runs.
- Typically, test fixtures include:
 - Objects or resources that are available for use by any test case.
 - Activities required to make these objects available and/or resource allocation and de-allocation: "setup" and "teardown".
- Allows multiple tests of the same or similar objects

Share fixture code for multiple tests

Overview 000000000 Assertions 000 シック・ 画・ 《川を《川を《日》 (日)

Fixtures ••••••• Eclipse Integration

January 18, 2011

24/44

Daniel Bruns, Erik Burger - JUnit



Test Fixture

- A test fixture is the context in which a test case runs.
- Typically, test fixtures include:
 - Objects or resources that are available for use by any test case.
 - Activities required to make these objects available and/or resource allocation and de-allocation: "setup" and "teardown".
- Allows multiple tests of the same or similar objects

Share fixture code for multiple tests

Overview 000000000 Assertions 000 - * ロ * * 母 * * 目 * * 目 * * のへで

Fixtures

Eclipse Integration

January 18, 2011

24/44

Daniel Bruns, Erik Burger – JUnit



Test Fixture

- A test fixture is the context in which a test case runs.
- Typically, test fixtures include:
 - Objects or resources that are available for use by any test case.
 - Activities required to make these objects available and/or resource allocation and de-allocation: "setup" and "teardown".
- Allows multiple tests of the same or similar objects
- Share fixture code for multiple tests

		< □ > < 🗗 > < 🗄		E nac
Overview	Assertions	Fixtures	Eclips	e Integration
00000000	000	• 0 000000000000	00	-
Daniel Bruns, Erik Burger –	JUnit	January 1	8, 2011	24/44

Before/After



• @Before: Methods annotated with @Before are executed before every test.

- @After: Methods annotated with @After are executed after every test.
- If there are e.g. 10 test, every @Before method is executed 10 times
- More than one @Before or @After is allowed
- Names of these methods are irrelevant, but must be public void

Assertions

シック・ 声・ 《川 》 《 山 》 《 日 》 -

Fixtures	Eclipse I	ntegration
0000000000000	00	
January 18.	2011	25/44



- @Before: Methods annotated with @Before are executed before every test.
- @After: Methods annotated with @After are executed after every test.
- If there are e.g. 10 test, every @Before method is executed 10 times
- More than one @Before or @After is allowed

Names of these methods are irrelevant, but must be public void



- @Before: Methods annotated with @Before are executed before every test.
- @After: Methods annotated with @After are executed after every test.
- If there are e.g. 10 test, every @Before method is executed 10 times
- More than one @Before or @After is allowed
- Names of these methods are irrelevant, but must be public void

うんら 同 《画》《画》《画》《曰》

Fixtures	Eclipse Integ	ratio
000000000000000	00	
January 18	2011	25/44



- @Before: Methods annotated with @Before are executed before every test.
- @After: Methods annotated with @After are executed after every test.
- If there are e.g. 10 test, every @Before method is executed 10 times
- More than one @Before or @After is allowed
- Names of these methods are irrelevant, but must be public void

うんら 同 《画》《画》《画》《曰》

ixtures	Eclipse Inte	gratic
000000000000000	00	
January .	18 2011	25/4



DQ P

25/44

Eclipse Integration

イロト イポト イヨト イヨト

January 18, 2011

- @Before: Methods annotated with @Before are executed before every test.
- @After: Methods annotated with @After are executed after every test.
- If there are e.g. 10 test, every @Before method is executed 10 times
- More than one @Before or @After is allowed
- Names of these methods are irrelevant, but must be public void

Fixture – Example



```
1
   public class MoneyTest {
2
        private Money f12CHF;
        private Money f14CHF;
4
        private Money f28USD;
       @Before
7
        public void setUp() {
            f12CHF= new Money(12, "CHF");
            f14CHF = new Money(14, "CHF");
            f28USD= new Money(28, "USD");
        }
12
   }
```

Overview

Assertions

・ロト・雪ト・声・・声・ かくの

Fixtures

Eclipse Integration

Daniel Bruns, Erik Burger - JUnit

January 18, 2011

Setup and Teardown



Setup

Use the @Before annotation on a method containing code to run before each test case.

Teardown (regardless of the verdict)

Use the @After annotation on a method containing code to run after each test case. These methods will run even if exceptions are thrown in the test case or an assertion fails.

It is allowed to have any number of these annotations

All methods annotated with @Before will be run before each test case, but they may be run in any order.

Overview 00000000 Daniel Bruns, Erik Burger – JUnit Assertions 000 - 《口》 《聞》 《臣》 《臣》 三目 のへの

Fixtures Eclipse Integration

January 18, 2011

Setup and Teardown



Setup

Use the @Before annotation on a method containing code to run before each test case.

Teardown (regardless of the verdict)

Use the @After annotation on a method containing code to run after each test case. These methods will run even if exceptions are thrown in the test case or an assertion fails.

It is allowed to have any number of these annotations

All methods annotated with @Before will be run before each test case, but they may be run in any order.

Overview 00000000 Daniel Bruns, Erik Burger – JUnit Assertions

Fixtures

Eclipse Integration

January 18, 2011

Setup and Teardown



Setup

Use the @Before annotation on a method containing code to run before each test case.

Teardown (regardless of the verdict)

Use the @After annotation on a method containing code to run after each test case. These methods will run even if exceptions are thrown in the test case or an assertion fails.

It is allowed to have any number of these annotations

All methods annotated with @Before will be run before each test case, but they may be run in any order.

Overview	Assertions	Fixtures	Eclipse Integratio
00000000	000	000000000000000000000000000000000000000	00
Daniel Bruns, Erik Burger - JUnit		January 1	3, 2011 27/44



- @BeforeClass: executed once before a test suite
- @AfterClass: executed once after a test suite
- Only one @BeforeClass and @AfterClass allowed
- Methods must be static

Overview 000000000

Daniel Bruns, Erik Burger - JUnit

Assertions 000 - * ロ > * 母 > * 油 > * 油 > ・ 油 ・ つくぐ

Fixtures

Eclipse Integration

January 18, 2011

Fixture – Example



```
public class MoneyTest {
1
        private static string currency;
       @BeforeClass
       public static void setGlobalCurrency() {
            this.currency = "CHF";
7
        }
        @Before
        public void setUp() {
            m12= new Money(12, this.currency);
            m14= new Money(14, this.currency);
        }
13
14
   }
```

Overview

Assertions

- 《日》《四》《田》《田》 ヨークへぐ

Fixtures

Eclipse Integration

Daniel Bruns, Erik Burger - JUnit

January 18, 2011



Exceptions that are expected on test executing

- Annotation using @Test
- @Test(expected=MyException.class)
- If no exception is thrown, or an unexpected exception occurs, the test will fail.
- That is, reaching the end of the method with no exception will cause a test case failure.

Eclipse Integration Fixtures January 18, 2011 30/44



Exceptions that are expected on test executing

- Annotation using @Test
- @Test(expected=MyException.class)
- If no exception is thrown, or an unexpected exception occurs, the test will fail.
- That is, reaching the end of the method with no exception will cause a test case failure.

シック・ 声・ 《 手 》 《 雪 》 《 雪 》 《 句 》 -

Fixtures	Eclipse Integration	on
000000000000000000000000000000000000000	00	
January 1	8, 2011 30/4	4



- Exceptions that are expected on test executing
- Annotation using @Test
- @Test(expected=MyException.class)
- If no exception is thrown, or an unexpected exception occurs, the test will fail.
- That is, reaching the end of the method with no exception will cause a test case failure.

● ◇ □ ◇ → ◇ 画 ◇ ▲ ■ ◇ ◇ ● ◇ ● ◇ ◇ ●

Fixtures	Eclipse Int	egration
000000000000000000000000000000000000000	00	
January 1	8, 2011	30/44



- Exceptions that are expected on test executing
- Annotation using @Test
- @Test(expected=MyException.class)
- If no exception is thrown, or an unexpected exception occurs, the test will fail.
- That is, reaching the end of the method with no exception will cause a test case failure.

Daniel Bruns, Erik Burger - JUnit

シック・ 画・ 《画》 《画》 《日》

Fixtures	Eclipse Integra	tio
000000000000000000000000000000000000000	00	
January	18, 2011 30	/44
Expected Exception



- Exceptions that are expected on test executing
- Annotation using @Test
- @Test(expected=MyException.class)

Assertions

- If no exception is thrown, or an unexpected exception occurs, the test will fail.
- That is, reaching the end of the method with no exception will cause a test case failure.

- 《ロ》 《聞》 《臣》 《臣》 三目 のへの

Fixtures	Eclipse Int	egratio
00000000000000000000000000000000000000	00	
January 1	8, 2011	30/44



```
new ArrayList <Object >().get(0);
```

Should throw an IndexOutOfBoundsException

```
@Test(expected = IndexOutOfBoundsException.class)
public void empty() {
    new ArrayList <Object >().get(0);
}
```

Assertions

イロト イポト イヨト イヨト nac

Fixtures **Eclipse Integration** January 18, 2011

Ignore/Timeout



Ignore

- Tests annotated using @Ignore are not executed
- Test runner reports that test was not run
- @Ignore("Reason") allows to specify a reason why a test was not run

Timeout

- Test allows to specify a timeout parameter
- @Test(timeout=10) fails if the test takes more than 10 milliseconds

Overview

Assertions

・ ▲ ロ ト ▲ 国 ト ▲ 国 ト ▲ 国 ト ▲ 日 ト

Fixtures	Eclipse Inte	gratio
000000000000000000000000000000000000000	00	
January 1	8, 2011	32/4

Daniel Bruns, Erik Burger - JUnit

Ignore/Timeout



Ignore

- Tests annotated using @Ignore are not executed
- Test runner reports that test was not run
- @Ignore("Reason") allows to specify a reason why a test was not run

Timeout

- Test allows to specify a timeout parameter
- @Test(timeout=10) fails if the test takes more than 10 milliseconds

		<ロ> < 聞 > < 言 > <	E> E ∽Q(
Overview	Assertions	Fixtures	Eclipse Integration
00000000	000	0000000000000	00
Daniel Bruns, Erik Burger - JUnit		January 18, 201	1 32/44



Motivation

If you want a test to run with several parameter values, you'd have to

- loop over a collection of values
- which means if there was a failure, the loop wouldn't terminate
- write unique test cases for each test data combination
- which could prove to be a lot of coding

Support in JUnit

With JUnit, you can create highly flexible testing scenarios easily

Overview 000000000 Assertions 000 シック・ 山 ・山下・山下・山下・山下・

Fixtures Eclipse Integration

Daniel Bruns, Erik Burger - JUnit



Motivation

If you want a test to run with several parameter values, you'd have to

- loop over a collection of values
- which means if there was a failure, the loop wouldn't terminate
- write unique test cases for each test data combination
- which could prove to be a lot of coding

Support in JUnit

With JUnit, you can create highly flexible testing scenarios easily

Overview 000000000

Daniel Bruns, Erik Burger - JUnit

Assertions 000 ● ◇ □ ◇ → ◇ 画 ◇ ▲ ■ ◇ ◇ ● ◇ ● ◇ ◇ ●

Fixtures Eclipse Integration



Motivation

If you want a test to run with several parameter values, you'd have to

- loop over a collection of values
- which means if there was a failure, the loop wouldn't terminate
- write unique test cases for each test data combination
- which could prove to be a lot of coding

Support in JUnit

With JUnit, you can create highly flexible testing scenarios easily

Overview 000000000

Daniel Bruns, Erik Burger - JUnit

Assertions 000 シック・ 声・ 《 手 》 《 雪 》 《 雪 》 《 句 》 -

Fixtures Eclipse Integration



Motivation

If you want a test to run with several parameter values, you'd have to

- loop over a collection of values
- which means if there was a failure, the loop wouldn't terminate
- write unique test cases for each test data combination
- which could prove to be a lot of coding

Support in JUnit

With JUnit, you can create highly flexible testing scenarios easily

Overview 000000000

Daniel Bruns, Erik Burger - JUnit

Assertions 000 シック・ 叫 ・ (明) ・ (明) ・ コ > -

Fixtures

Eclipse Integration

January 18, 2011



Motivation

If you want a test to run with several parameter values, you'd have to

- loop over a collection of values
- which means if there was a failure, the loop wouldn't terminate
- write unique test cases for each test data combination
- which could prove to be a lot of coding

Support in JUnit

With JUnit, you can create highly flexible testing scenarios easily

Overview 000000000

Daniel Bruns, Erik Burger - JUnit

Assertions 000 シック・ 叫 ・ (明) ・ (明) ・ コ > -

Fixtures

Eclipse Integration

January 18, 2011



Motivation

If you want a test to run with several parameter values, you'd have to

- loop over a collection of values
- which means if there was a failure, the loop wouldn't terminate
- write unique test cases for each test data combination
- which could prove to be a lot of coding

Support in JUnit

With JUnit, you can create highly flexible testing scenarios easily

Overview	
000000000	

Daniel Bruns, Erik Burger - JUnit

Assertions

Fi

・ロト・<回ト< Eト < Eト < E < のへで

Fixtures

Eclipse Integration

January 18, 2011



Creating a parameterised test

1	Create a generic test and decorate it with the @Test annotation
	Create a static feeder method that returns a Collection type and decorate it with the @Parameters annotation
	Create class members for the parameter types required in the generic method defined in Step 1
	Create a constructor that takes these parameter types and correspondingly links them to the class members defined in Step 3
	Specify the test case be run with the Parameterized class via the @RunWith annotation

Overview 000000000 Daniel Bruns, Erik Burger – JUnit Assertions 000 - ▲ ロ ▶ ▲ 団 ▶ ▲ 豆 ▶ ▲ 豆 ▶ ▲ 団 ▶ ▲ 団 ▶ ▲ 団 ▶ ▲ 団 ▶ ▲ 豆 ▶ ▲ 豆 ▶ ▲ 豆 ▶ ▲ 豆 ▶ ▲ 豆 ▶ ▲

Fixtures	Eclipse	Integration
000000000000000000000000000000000000000	00	
January '	18, 2011	34/44



Creating a parameterised test

- Oreate a generic test and decorate it with the @Test annotation
- ② Create a static feeder method that returns a Collection type and decorate it with the @Parameters annotation
- Oreate class members for the parameter types required in the generic method defined in Step 1
- Oreate a constructor that takes these parameter types and correspondingly links them to the class members defined in Step 3
- Specify the test case be run with the Parameterized class via the @RunWith annotation

Overview 000000000

Daniel Bruns, Erik Burger - JUnit

Assertions 000 うせん 聞い ふぼう ふぼう ふしゃ

Fixtures	Eclipse I	ntegratior
000000000000000000000000000000000000000	00	
January '	18, 2011	34/44



Creating a parameterised test

- Oreate a generic test and decorate it with the @Test annotation
- ② Create a static feeder method that returns a Collection type and decorate it with the @Parameters annotation
- ③ Create class members for the parameter types required in the generic method defined in Step 1
- Oreate a constructor that takes these parameter types and correspondingly links them to the class members defined in Step 3
- Specify the test case be run with the Parameterized class via the @RunWith annotation

Overview 000000000

Daniel Bruns, Erik Burger - JUnit

Assertions 000 シック・ 声・ 《川 》 《 山 》 《 日 》 -

Fixtures Eclipse Integration



Creating a parameterised test

- Oreate a generic test and decorate it with the @Test annotation
- ② Create a static feeder method that returns a Collection type and decorate it with the @Parameters annotation
- ③ Create class members for the parameter types required in the generic method defined in Step 1
- Generation of the second se
- Specify the test case be run with the Parameterized class via the @RunWith annotation

Overview 000000000

Daniel Bruns, Erik Burger - JUnit

Assertions 000 Fixtures Eclipse Integration



Creating a parameterised test

- Oreate a generic test and decorate it with the @Test annotation
- ② Create a static feeder method that returns a Collection type and decorate it with the @Parameters annotation
- ③ Create class members for the parameter types required in the generic method defined in Step 1
- Generation of the second se
- Specify the test case be run with the Parameterized class via the @RunWith annotation

		<	< ≡ >	E ∽Q
Overview	Assertions	Fixtures	Eclips	se Integration
Daniel Bruns, Erik Burger – JU	nit	January 18	, 2011	34/44

Parameterised Test – Example





Creating a test suite

- Tests can be combined to test suites

Overview

Assertions

イロト イポト イヨト イヨト nac 3

Eclipse Integration Fixtures

Daniel Bruns, Erik Burger - JUnit

January 18, 2011



Creating a test suite

- Tests can be combined to test suites
- suites can contain other suites

Overview

Assertions

イロト イポト イヨト イヨト nac 1

Eclipse Integration Fixtures January 18, 2011

36/44

Daniel Bruns, Erik Burger - JUnit



Creating a test suite

- Tests can be combined to test suites
- suites can contain other suites
- useful for partitioning your test scenarios
- well supported by Test Runners (see example)

Overview 000000000

Daniel Bruns, Erik Burger - JUnit

Assertions 000 うせん 川 ふかく ボット 白マ

Fixtures	Eclipse	Integratio
000000000000000000000000000000000000000	00	
January	18, 2011	36/44



Creating a test suite

- Tests can be combined to test suites
- suites can contain other suites
- useful for partitioning your test scenarios
- well supported by Test Runners (see example)

Daniel Bruns, Erik Burger - JUnit

Assertions 000 - ▲ロ > ▲ 圖 > ▲ 圖 > ▲ 圖 - 夕 Q (~

Fixtures **Eclipse Integration** January 18, 2011 36/44

Test Suite – Example



```
1
   import org.junit.runner.RunWith;
   import org.junit.runners.Suite;
   @RunWith(Suite.class)
   @Suite.SuiteClasses({
       MyTest1.class,
       MyTest2.class,
       MyTest3.class
       }
11
   )
   public class AllTests {
   }
```

Overview

Assertions

 Fixtures
 Eclipse Integration

 ○○○○○○○○○○○●
 ○○

Daniel Bruns, Erik Burger - JUnit

January 18, 2011



- The JUnit framework does not provide a graphical test runner. Instead, it provides an API that can be used by IDEs to run test cases and a textual runner than can be used from a command line.
- Eclipse and Netbeans each provide a graphical test runner that is integrated into their respective environments.

う ク ク ・ ゴ ・ イ 山 ・ 〈 山 ・ 〈 山 ・ 〈

Fixtures	Eclipse Integration
0000000000000	•0
January 18	3, 2011 38/44



With the runner provided by JUnit:

- When a class is selected for execution, all the test case methods in the class will be run.
- The order in which the methods in the class are called (i.e. the order of test case execution) is not predictable.

Other Runners

- Test runners provided by IDEs may allow the user to select particular methods, or to set the order of execution.
- It is good practice to write tests with are independent of execution order, and that are without dependencies on the state any previous test(s).

Overview 00000000 Daniel Bruns, Erik Burger – JUnit Assertions 000 - < ロ > < 回 > < 目 > < 目 > < 回 > < 回 > < 回 > <

Fixtures	Eclipse Integration
0000000000000	00
January 18, 2	011 39/44



With the runner provided by JUnit:

- When a class is selected for execution, all the test case methods in the class will be run.
- The order in which the methods in the class are called (i.e. the order of test case execution) is not predictable.

Other Runners

- Test runners provided by IDEs may allow the user to select particular methods, or to set the order of execution.
- It is good practice to write tests with are independent of execution order, and that are without dependencies on the state any previous test(s).

Overview 00000000 Daniel Bruns, Erik Burger – JUnit Assertions 000 ● ◇ □ ◇ → ◇ 画 ◇ ▲ ■ ◇ ◇ ● ◇ ● ◇ ◇ ●

Fixtures	Eclipse Integration
0000000000000	00
January 18,	2011 39/44



With the runner provided by JUnit:

- When a class is selected for execution, all the test case methods in the class will be run.
- The order in which the methods in the class are called (i.e. the order of test case execution) is not predictable.

Other Runners

- Test runners provided by IDEs may allow the user to select particular methods, or to set the order of execution.
- It is good practice to write tests with are independent of execution order, and that are without dependencies on the state any previous test(s).

Overview 000000000

Daniel Bruns, Erik Burger - JUnit

Assertions 000 ◆□▶▲□▶▲□▶▲□▶▲□

Fixtures	Eclipse Integration
0000000000000	0•
January 18	, 2011 39/44



With the runner provided by JUnit:

- When a class is selected for execution, all the test case methods in the class will be run.
- The order in which the methods in the class are called (i.e. the order of test case execution) is not predictable.

Other Runners

- Test runners provided by IDEs may allow the user to select particular methods, or to set the order of execution.
- It is good practice to write tests with are independent of execution order, and that are without dependencies on the state any previous test(s).

Overview 00000000 Daniel Bruns, Erik Burger – JUnit Assertions 000 ・ロト・日下・モー・モー しょうくの

Fixtures	Eclipse Integration	n
0000000000000	0	
January 18, 2	.011 39/44	



Hand-writing test cases is a tedious job...

... and may be another source of error.

Test case generation (TCG) does all the dirty work.

- Input: parameters to test and oracle of some sort
- Sometimes not even those required (e.g., in Verification-based Testing (VBT))

▲ロ > ▲母 > ▲目 > ▲目 > ▲目 > ● ●



- Hand-writing test cases is a tedious job...
- ... and may be another source of error.

Test case generation (TCG) does all the dirty work.

- Input: parameters to test and oracle of some sort
- Sometimes not even those required (e.g., in Verification-based Testing (VBT))



- Hand-writing test cases is a tedious job...
- ... and may be another source of error.

Test case generation (TCG) does all the dirty work.

- Input: parameters to test and oracle of some sort
- Sometimes not even those required (e.g., in Verification-based Testing (VBT))

Testing with JMLUnitNG



JMLUnitNG

- By Dan Zimmerman (U Washington—Tacoma), 2010
- Complete rewrite of JMLUnit (now support for Java 1.5)
- Based on TestNG (instead of JUnit)
- Current version 1.0a2, released 25 December 2010
- Builds (input and) oracle from JML specifications

Classification of Tests

- Pass: Result matches post-condition
- Fail: Result does not match post-condition (or unexpected exception)
- Meaningless: Test input does not match pre-condition

▲□▶▲□▶▲目▶▲目▶▲目▼

JMLUnitNG 0000 Daniel Bruns, Erik Burger – JUnit

Testing with JMLUnitNG



JMLUnitNG

- By Dan Zimmerman (U Washington—Tacoma), 2010
- Complete rewrite of JMLUnit (now support for Java 1.5)
- Based on TestNG (instead of JUnit)
- Current version 1.0a2, released 25 December 2010
- Builds (input and) oracle from JML specifications

Classification of Tests

- Pass: Result matches post-condition
- Fail: Result does not match post-condition (or unexpected exception)
- Meaningless: Test input does not match pre-condition

JMLUnitNG 0000 Daniel Bruns, Erik Burger – JUnit



X	y	res	verdict
-1	-1	-1	meaningless
	0	0	pass
	1	1	pass
1	0	0	fail
1	1	1	fail
÷	÷	-	?

JMLUnitNG 00000 Daniel Bruns, Erik Burger – JUnit 

x	У	res	verdict
-1	-1	A1	meaningless
0	0	0	pass
0	- 1	1	pass
-1	0	0	fail
1	1	1	fail
-			?

JMLUnitNG 00000 Daniel Bruns, Erik Burger – JUnit 

X	y	res	verdict
-1	-1	H1	meaningless
0	0	0	pass
0	1	1	pass
1		0	fail
1	1	1	fail
-			?

JMLUnitNG 00000 Daniel Bruns, Erik Burger – JUnit ・ロト・日本・モート ヨー うへつ



x	y	res	verdict
-1	-1	H	meaningless
0	0	0	pass
0	1	1	pass
1	0	0	fail
1	1	1	fail
:			?

JMLUnitNG 00000 Daniel Bruns, Erik Burger – JUnit 

x	y	res	verdict
-1	-1	A1	meaningless
0	0	0	pass
0	1	1	pass
1	0	0	fail
1	1	1	fail
÷			?

JMLUnitNG 00000 Daniel Bruns, Erik Burger – JUnit ・ロト・日本・モート ヨー うへつ
Example



x	y	res	verdict
-1	-1	A1	meaningless
0	0	0	pass
0	1	1	pass
1	0	0	fail
1	1	1	fail
:	-		?

JMLUnitNG 00000 Daniel Bruns, Erik Burger – JUnit

Example



x	У	res	verdict
-1	-1	A1	meaningless
0	0	0	pass
0	1	1	pass
1	0	0	fail
1	1	1	fail
:	÷	÷	?

JMLUnitNG 00000 Daniel Bruns, Erik Burger – JUnit 

Starting point: Java classes with JML specifications

Phases

▲ロト ▲ □ ト ▲ 三 ト ▲ 三 ● の < ⊙

JMLUnitNG 00000 Daniel Bruns, Erik Burger – JUnit



Starting point: Java classes with JML specifications



▲ロト ▲ □ ト ▲ 三 ト ▲ 三 ● の < ⊙

JMLUnitNG 00000 Daniel Bruns, Erik Burger – JUnit



Starting point: Java classes with JML specifications





Starting point: Java classes with JML specifications



JMLUnitNG 00000 Daniel Bruns, Erik Burger – JUnit ・ロト・西ト・山下・山下・ ひゃつ



Starting point: Java classes with JML specifications



Live Demo

・ロト・日本・山田・山田・山口・

JMLUnitNG 00000 Daniel Bruns, Erik Burger – JUnit



Starting point: Java classes with JML specifications

Phases	
Preparation: create necessary test classes	automatic
Provide data: enter specific test data and fixtures	manual
 Global data: used in every test Local data: used in this test 	
Test generation	automatic
Test running	automatic

nac

< □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > <



Starting point: Java classes with JML specifications



Live Demo

JMLUnitNG 00000 Daniel Bruns, Erik Burger – JUnit ▲□▶▲□▶▲□▶▲□▶▲□▶▲□

Conclusion



Disclaimer

- JMLUnitNG is still in alpha stage!
- Does not provide much automation / Eclipse integration yet
- May have bugs itself
- Contact Dan Zimmerman <dmz@acm.org> in doubt