UNIVERSITÄT PASSAU

Search-Based Software Engineering

Search-based Testing

Gordon Fraser
Lehrstuhl für Software Engineering II

Evolving Unit Tests

Encoding Unit Tests

```
public class Foo {
  private Bar bar;

  public Foo(Bar b) {
    this.bar = bar;
  }

  public boolean isBar(int x) {
   if(bar.getValue() > x)
      return true;
    else
      return false;
  }
}
```

```
public class Bar {
  private int value = 0;

  public void doStuff(int y) {
    if(y % 2 == 0)
      value++;
   }
  }
```

Unit Test:

```
@Test
void test() {
  Bar bar = new Bar();
  Foo foo = new Foo(bar);
  foo.isBar(42);
  // omitting assertions
  // for now
}
```

Chromosome Encoding:

- int x = 42;
- Bar bar = new Bar();
- Foo foo = new Foo(bar);
- foo.isBar(x);

```
public class Bar {
  private int value = 0;

  public void doStuff(int y) {
    if(y % 2 == 0)
      value++;
 }
}
```

```
public class Foo {
  private Bar bar;

  public Foo(Bar b) {
    this.bar = bar;
  }

  public boolean isBar(int x) {
   if(bar.getValue() > x)
      return true;
    else
      return false;
  }
}
```

Chromosome Encoding:

int x = 42;

Bar bar = new Bar();

Foo foo = new Foo(bar);

foo.isBar(x);

- A unit test is a list of statements
- A statement can be:
  - A call to a constructor
  - A method call on an existing object
  - A primitive value
  - …
- The length of a unit test is variable
- There are dependencies between statements

```
public class Bar {
  private int value = 0;

  public void doStuff(int y) {
    if(y % 2 == 0)
      value++;
 }
}


public class Foo {
  private Bar bar;

  public Foo(Bar b) {
    this.bar = bar;
  }

  public boolean isBar(int x) {
   if(bar.getValue() > x)
      return true;
    else
      return false;
  }
}
```

Insertion: Call on CUT

<Foo>.isBar(<int>);

```
public class Bar {
  private int value = 0;

  public void doStuff(int y) {
    if(y % 2 == 0)
      value++;
 }
}


public class Foo {
  private Bar bar;

  public Foo(Bar b) {
    this.bar = bar;
  }

  public boolean isBar(int x) {
   if(bar.getValue() > x)
      return true;
    else
      return false;
  }
}
```

Insertion: Call on CUT

Foo foo = new Foo(<Bar>);

<Foo>.isBar(<int>);

```
public class Bar {
  private int value = 0;

  public void doStuff(int y) {
    if(y % 2 == 0)
      value++;
 }
}
```

```
public class Foo {
  private Bar bar;

  public Foo(Bar b) {
    this.bar = bar;
  }

  public boolean isBar(int x) {
   if(bar.getValue() > x)
      return true;
    else
      return false;
  }
}
```

Insertion: Call on CUT

Foo foo = new Foo(<Bar>);

foo.isBar(<int>);

```
public class Bar {
  private int value = 0;

  public void doStuff(int y) {
    if(y % 2 == 0)
      value++;
 }
}
```

```
public class Foo {
  private Bar bar;

  public Foo(Bar b) {
    this.bar = bar;
  }

  public boolean isBar(int x) {
   if(bar.getValue() > x)
      return true;
    else
      return false;
  }
}
```

Insertion: Call on CUT

```
public class Bar {
  private int value = 0;

  public void doStuff(int y) {
    if(y % 2 == 0)
      value++;
 }
}
```

Bar bar = new Bar();

Foo foo = new Foo(<Bar>);

foo.isBar(<int>);

```
public class Foo {
  private Bar bar;

  public Foo(Bar b) {
    this.bar = bar;
  }

  public boolean isBar(int x) {
   if(bar.getValue() > x)
      return true;
    else
      return false;
  }
}
```

Insertion: Call on CUT

```
public class Bar {
  private int value = 0;

  public void doStuff(int y) {
    if(y % 2 == 0)
      value++;
 }
}
```

Bar bar = new Bar();

Foo foo = new Foo(bar);

foo.isBar(<int>);

```
public class Foo {
  private Bar bar;

  public Foo(Bar b) {
    this.bar = bar;
  }

  public boolean isBar(int x) {
   if(bar.getValue() > x)
      return true;
    else
      return false;
  }
}
```

Insertion: Call on CUT

int x = 42;

Bar bar = new Bar();

Foo foo = new Foo(bar);

foo.isBar(<int>);

```
public class Bar {
  private int value = 0;

  public void doStuff(int y) {
    if(y % 2 == 0)
      value++;
 }
}
```

```
public class Foo {
  private Bar bar;

  public Foo(Bar b) {
    this.bar = bar;
  }

  public boolean isBar(int x) {
   if(bar.getValue() > x)
      return true;
    else
      return false;
  }
}
```

Insertion: Call on CUT

int x = 42;

Bar bar = new Bar();

Foo foo = new Foo(bar);

foo.isBar(x);

```
public class Bar {
  private int value = 0;

  public void doStuff(int y) {
    if(y % 2 == 0)
      value++;
 }
}
```

```
public class Foo {
  private Bar bar;

  public Foo(Bar b) {
    this.bar = bar;
  }

  public boolean isBar(int x) {
   if(bar.getValue() > x)
      return true;
    else
      return false;
  }
}
```

Insertion: Call on CUT

int x = 42;

Bar bar = new Bar();

Foo foo = new Foo(bar);

foo.isBar(x);

<Foo>.isBar(<int>);

```
public class Bar {
  private int value = 0;

  public void doStuff(int y) {
    if(y % 2 == 0)
      value++;
 }
}
```

```
public class Foo {
  private Bar bar;

  public Foo(Bar b) {
    this.bar = bar;
  }

  public boolean isBar(int x) {
   if(bar.getValue() > x)
      return true;
    else
      return false;
  }
}
```

Insertion: Call on CUT

int x = 42;

Bar bar = new Bar();

Foo foo = new Foo(bar);

foo.isBar(x);

foo.isBar(<int>);

```
public class Bar {
  private int value = 0;

  public void doStuff(int y) {
    if(y % 2 == 0)
      value++;
 }
}
```

```
public class Foo {
  private Bar bar;

  public Foo(Bar b) {
    this.bar = bar;
  }

  public boolean isBar(int x) {
   if(bar.getValue() > x)
      return true;
    else
      return false;
  }
}
```

Insertion: Call on CUT

int x = 42;

Bar bar = new Bar();

Foo foo = new Foo(bar);

foo.isBar(x);

int y = 0;

foo.isBar(<int>);

```
public class Bar {
  private int value = 0;

  public void doStuff(int y) {
    if(y % 2 == 0)
      value++;
 }
}
```

```
public class Foo {
  private Bar bar;

  public Foo(Bar b) {
    this.bar = bar;
  }

  public boolean isBar(int x) {
   if(bar.getValue() > x)
      return true;
    else
      return false;
  }
}
```

Insertion: Call on CUT

int x = 42;

Bar bar = new Bar();

Foo foo = new Foo(bar);

foo.isBar(x);

int y = 0;

foo.isBar(y);

```
public class Bar {
  private int value = 0;

  public void doStuff(int y) {
    if(y % 2 == 0)
      value++;
 }
}
```

```
public class Foo {
  private Bar bar;

  public Foo(Bar b) {
    this.bar = bar;
  }

  public boolean isBar(int x) {
   if(bar.getValue() > x)
      return true;
    else
      return false;
  }
}
```

Insertion: Call on existing object

```
int x = 42;

Bar bar = new Bar();

Foo foo = new Foo(bar);

foo.isBar(x);

int y = 0;

foo.isBar(y);
```

```
public class Bar {
  private int value = 0;

  public void doStuff(int y) {
    if(y % 2 == 0)
      value++;
 }
}
```

```
public class Foo {
  private Bar bar;

  public Foo(Bar b) {
    this.bar = bar;
  }

  public boolean isBar(int x) {
   if(bar.getValue() > x)
      return true;
    else
      return false;
  }
}
```

Insertion: Call on existing object

int x = 42;

Bar bar = new Bar();

Foo foo = new Foo(bar);

bar

foo.isBar(x);

int y = 0;

foo.isBar(y);

```
public class Bar {
  private int value = 0;

  public void doStuff(int y) {
    if(y % 2 == 0)
      value++;
 }
}
```

```
public class Foo {
  private Bar bar;

  public Foo(Bar b) {
    this.bar = bar;
  }

  public boolean isBar(int x) {
   if(bar.getValue() > x)
      return true;
    else
      return false;
  }
}
```

Insertion: Call on existing object

int x = 42;

Bar bar = new Bar();

Foo foo = new Foo(bar);

bar.doStuff(<int>);

foo.isBar(x);

int y = 0;

foo.isBar(y);

```
public class Bar {
  private int value = 0;

  public void doStuff(int y) {
    if(y % 2 == 0)
      value++;
 }
}
```

```
public class Foo {
  private Bar bar;

  public Foo(Bar b) {
    this.bar = bar;
  }

  public boolean isBar(int x) {
   if(bar.getValue() > x)
      return true;
    else
      return false;
  }
}
```

Insertion: Call on existing object

```
int x = 42;
Bar bar = new Bar();
Foo foo = new Foo(bar);
bar.doStuff(x);
foo.isBar(x);
int y = 0;
foo.isBar(y);
```

```
public class Bar {
  private int value = 0;

  public void doStuff(int y) {
    if(y % 2 == 0)
      value++;
 }
}
```

```
public class Foo {
  private Bar bar;

  public Foo(Bar b) {
    this.bar = bar;
  }

  public boolean isBar(int x) {
   if(bar.getValue() > x)
      return true;
    else
      return false;
  }
}
```

Change statement

int x = 42;

Bar bar = new Bar();

Foo foo = new Foo(bar);

bar.doStuff(x);

foo.isBar(x);

int y = 0;

foo.isBar(y);

```
public class Bar {
  private int value = 0;

  public void doStuff(int y) {
    if(y % 2 == 0)
      value++;
 }
}
```

```
public class Foo {
  private Bar bar;

  public Foo(Bar b) {
    this.bar = bar;
  }

  public boolean isBar(int x) {
   if(bar.getValue() > x)
      return true;
    else
      return false;
  }
}
```

Change statement

int x = 38;

Bar bar = new Bar();

Foo foo = new Foo(bar);

bar.doStuff(x);

foo.isBar(x);

int y = 0;

foo.isBar(y);

```
public class Bar {
  private int value = 0;

  public void doStuff(int y) {
    if(y % 2 == 0)
      value++;
 }
}
```

```
public class Foo {
  private Bar bar;

  public Foo(Bar b) {
    this.bar = bar;
  }

  public boolean isBar(int x) {
   if(bar.getValue() > x)
      return true;
    else
      return false;
  }
}
```

Change statement

int x = 38;

Bar bar = new Bar();

Foo foo = new Foo(bar);

bar.doStuff(x);

foo.isBar(x);

int y = 0;

foo.isBar(y);

```
public class Bar {
  private int value = 0;

  public void doStuff(int y) {
    if(y % 2 == 0)
      value++;
 }
}
```

```
public class Foo {
  private Bar bar;

  public Foo(Bar b) {
    this.bar = bar;
  }

  public boolean isBar(int x) {
   if(bar.getValue() > x)
      return true;
    else
      return false;
  }
}
```

Delete statement

int x = 38;

Bar bar = new Bar();

Foo foo = new Foo(bar);

bar.doStuff(x);

foo.isBar(x);

int y = 0;

boolean ret = foo.isBar(y);

assertTrue(ret);

```
public class Bar {
  private int value = 0;

  public void doStuff(int y) {
    if(y % 2 == 0)
      value++;
 }
}
```

```
public class Foo {
  private Bar bar;

  public Foo(Bar b) {
    this.bar = bar;
  }

  public boolean isBar(int x) {
   if(bar.getValue() > x)
      return true;
    else
      return false;
  }
}
```

Delete statement

int x = 38;

Bar bar = new Bar();

Foo foo = new Foo(bar);

bar.doStuff(x);

foo.isBar(x);

int y = 0;

foo.isBar(x);

```
public class Bar {
  private int value = 0;

  public void doStuff(int y) {
    if(y % 2 == 0)
      value++;
 }
}
```

```
public class Foo {
  private Bar bar;

  public Foo(Bar b) {
    this.bar = bar;
  }

  public boolean isBar(int x) {
   if(bar.getValue() > x)
      return true;
    else
      return false;
  }
}
```

Delete statement

int x = 38;

Bar bar = new Bar();

Foo foo = new Foo(bar);

bar.doStuff(x);

foo.isBar(x);

foo.isBar(x);

```
public class Bar {
  private int value = 0;

  public void doStuff(int y) {
    if(y % 2 == 0)
      value++;
 }
}
```

```
public class Foo {
  private Bar bar;

  public Foo(Bar b) {
    this.bar = bar;
  }

  public boolean isBar(int x) {
   if(bar.getValue() > x)
      return true;
    else
      return false;
  }
}
```

Delete statement

int x = 38;

Bar bar = new Bar();

Foo foo = new Foo(bar);

bar.doStuff(x);

foo.isBar(x);

foo.isBar(x);

```
public class Bar {
  private int value = 0;

  public void doStuff(int y) {
    if(y % 2 == 0)
      value++;
 }
}
```

```
public class Foo {
  private Bar bar;

  public Foo(Bar b) {
    this.bar = bar;
  }

  public boolean isBar(int x) {
   if(bar.getValue() > x)
      return true;
    else
      return false;
  }
}
```

Delete statement

int x = 38;

Bar bar = new Bar();

Foo foo = new Foo(bar);

bar.doStuff(x);

foo.isBar(x);

foo.isBar(x);

```
public class Bar {
  private int value = 0;

  public void doStuff(int y) {
    if(y % 2 == 0)
      value++;
 }
}
```

```
public class Foo {
  private Bar bar;

  public Foo(Bar b) {
    this.bar = bar;
  }

  public boolean isBar(int x) {
   if(bar.getValue() > x)
      return true;
    else
      return false;
  }
}
```

Delete statement

int x = 38;

Bar bar = new Bar();

bar.doStuff(x);

```
public class Bar {
  private int value = 0;

  public void doStuff(int y) {
    if(y % 2 == 0)
      value++;
 }
}
```

```
public class Foo {
  private Bar bar;

  public Foo(Bar b) {
    this.bar = bar;
  }

  public boolean isBar(int x) {
   if(bar.getValue() > x)
      return true;
    else
      return false;
  }
}
```

Crossover

DateTime var0 = new DateTime()

int var1 = 54

TimeOfDay var2 = var0.toTimeOfDay()

int var3 = var0.getSecondOfMinute()

long var0 = 48

DateTime var1 = new DateTime(var0)

DateTime var2 = var1.plusWeeks(var0)

DateTime var3 = var1.minus(var2)

Crossover

DateTime var0 = new DateTime()

int var1 = 54

DateTime var2 = var1.plusWeeks(var0)

DateTime var3 = var1.minus(var2)

long var0 = 48

DateTime var1 = new DateTime(var0)

TimeOfDay var2 = var0.toTimeOfDay()

int var3 = var0.getSecondOfMinute()

Crossover

DateTime var0 = new DateTime()

int var1 = 54

DateTime var2 = var0.plusWeeks(var1)

DateTime var3 = var2.minus(var1)

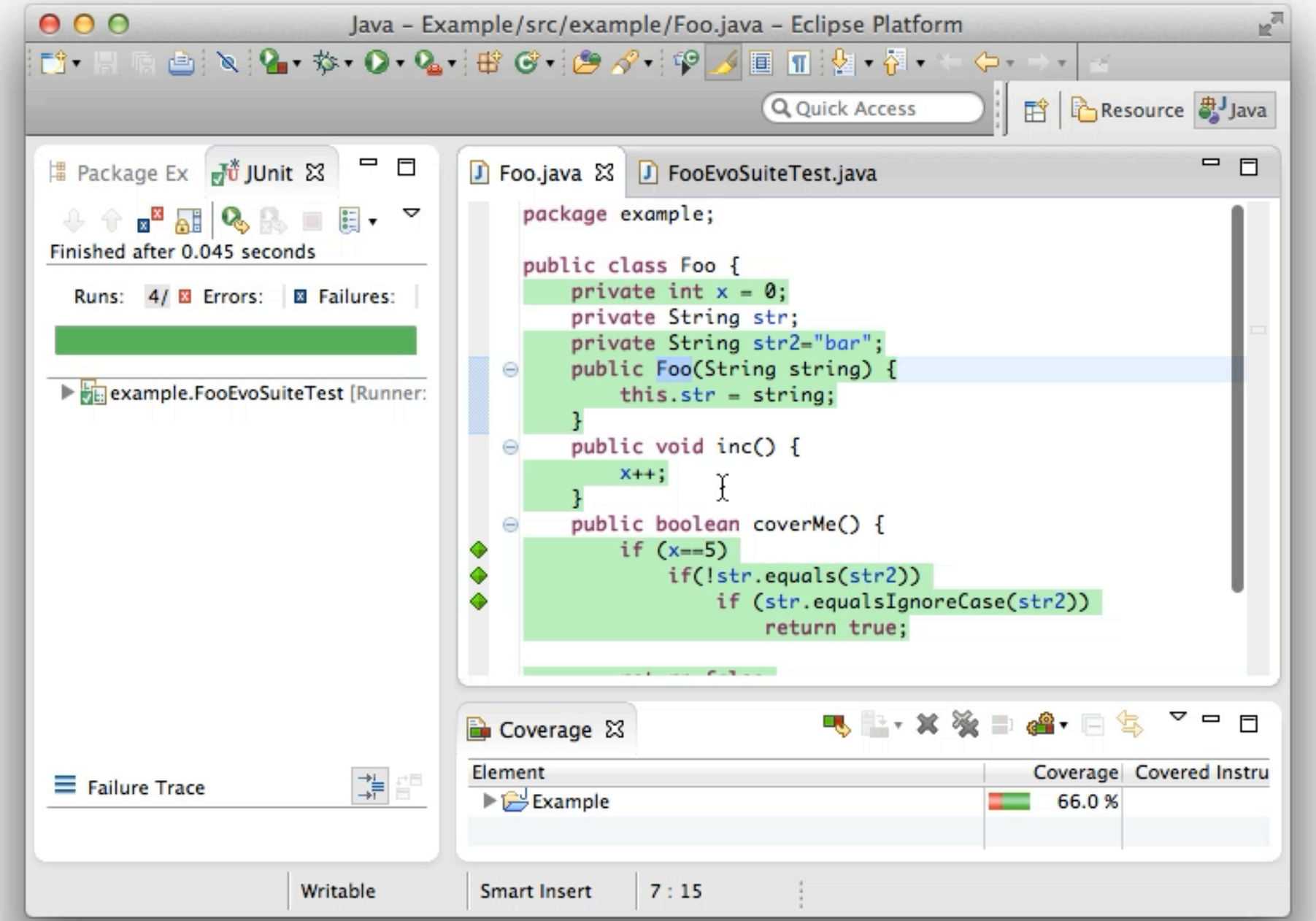
long var0 = 48

DateTime var1 = new DateTime(var0)

TimeOfDay var2 = var1.toTimeOfDay()

int var3 = var1.getSecondOfMinute()

Repair might be necessary because of dependencies


Java – Example/src/example/Foo.java – Eclipse Platform
Quick Access
Resource
Java
Package Ex
JUnit
Finished after 0.045 seconds
Runs: 4/
Errors:
Failures:
example.FooEvoSuiteTest [Runner:
Failure Trace
Foo.java
FooEvoSuiteTest.java
package example;
public class Foo {
private int x = 0;
private String str;
private String str2="bar";
public Foo(String string) {
this.str = string;
}
public void inc() {
x++;
}
public boolean coverMe() {
if (x==5)
if(!str.equals(str2))
if (str.equalsIgnoreCase(str2))
return true;
Coverage
Element
Coverage
Covered Instru
Example
66.0 %
Writable
Smart Insert
7 : 15

EVOSUITE

pynguin --module-name queue_example --project-path . --output-path . --seed ⌥⌘2

```
~/pynguin                                                    4s pynguin 07:02:22
❯ pynguin --module-name queue_example --project-path . --output-path . --seed 1629381673714481067 -v
```

Regression Assertions

Assertions

```
class Foo {
  boolean isFoo();
  boolean hasBar();
  int getZoo();
  Boo getBoo();
}
```

```
Foo var0 = new Foo()
Foo var1 = new Foo()
...
var1.bar();
Boo var9 = var1.getBoo()
```

```
assertTrue(var0.isFoo());
assertFalse(var0.hasBar());
assert(var0.getZoo()==27);
```

Assertions

```
class Foo {
  boolean isFoo();
  boolean hasBar();
  int getZoo();
  Boo getBoo();
}
```

```
Foo var0 = new Foo()
Foo var1 = new Foo()
...
var1.bar();
Boo var9 = var1.getBoo()
```

```
assertTrue(var0.isFoo());
assertFalse(var0.hasBar());
assert(var0.getZoo()==27);

assertTrue(var1.isFoo());
assertFalse(var1.hasBar());
assert(var1.getZoo()==27);

assertEqual(var0, var1);
```

Assertions

```
class Foo {
  boolean isFoo();
  boolean hasBar();
  int getZoo();
  Boo getBoo();
}
```

```
Foo var0 = new Foo()
Foo var1 = new Foo()
...
var1.bar();
Boo var9 = var1.getBoo()
```

```
assertTrue(var0.isFoo());
assertFalse(var0.hasBar());
assert(var0.getZoo()==27);

assertTrue(var1.isFoo());
assertFalse(var1.hasBar());
assert(var1.getZoo()==27);

assertEqual(var0, var1);
```

Assertions

```
class Foo {
  boolean isFoo();
  boolean hasBar();
  int getZoo();
  Boo getBoo();
}
```

```
Foo var0 = new Foo()
Foo var1 = new Foo()
...
var1.bar();
Boo var9 = var1.getBoo()
```

```
assertTrue(var0.isFoo());
assertFalse(var0.hasBar());
assert(var0.getZoo()==27);

assertTrue(var1.isFoo());
assertFalse(var1.hasBar());
assert(var1.getZoo()==27);

assertNotEqual(var0, var1);
```

Assertions

```
class Foo {
  boolean isFoo();
  boolean hasBar();
  int getZoo();
  Boo getBoo();
}
```

```
Foo var0 = new Foo()
Foo var1 = new Foo()
...
var1.bar();
Boo var9 = var1.getBoo()
```

```
assertTrue(var0.isFoo());
assertFalse(var0.hasBar());
assert(var0.getZoo()==27);

assertTrue(var1.isFoo());
assertFalse(var1.hasBar());
assert(var1.getZoo()==27);

assertNotEqual(var0, var1);

assertTrue(var9.hasBoo())
```

```
LocalDate date = new LocalDate(2010, 7, 15);
date.plusYears(1);
assertEquals(date.getYear(), 2011);
```

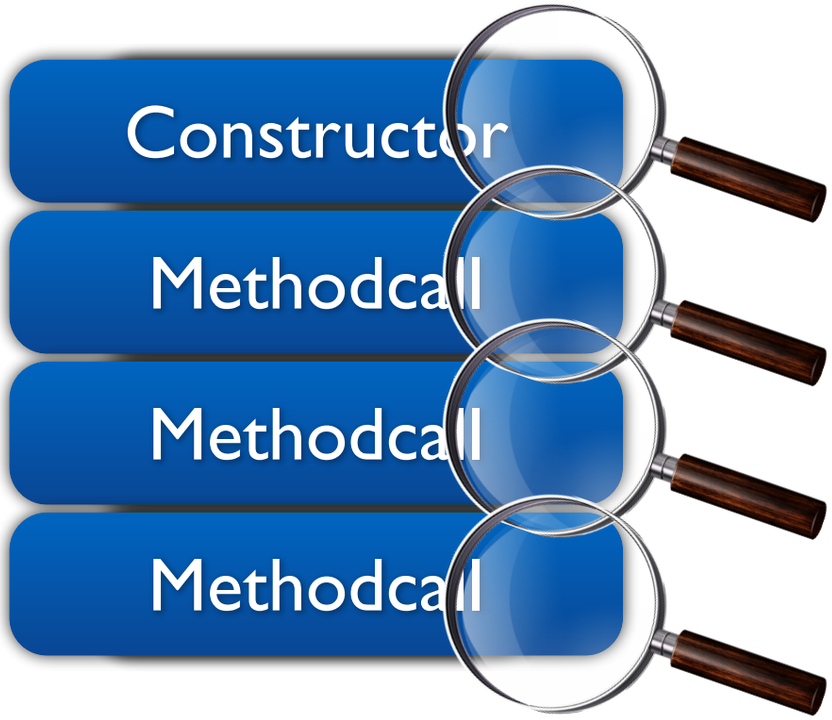
```
assertEquals(date.size(), 3);
assertEquals(date.getValue(YEAR), 2010);
assertEquals(date.getValue(MONTH_OF_YEAR), 7);
assertEquals(date.getValue(DAY_OF_MONTH), 15);
assertEquals(date.getLocalMillis(), ...);
assertEquals(date, date);
assertEquals(date.compareTo(date), 0);
assertEquals(date.getYearOfCentury(), ...);
assertEquals(date.getYear(), 2010);
assertEquals(date.getWeekyear(), ...);
assertEquals(date.getMonthOfYear(), 7);
assertEquals(date.getWeekOfWeekyear(), ...);
assertEquals(date.getDayOfWeek(), ...);
assertEquals(date.getDayOfMonth(), ...);
```

```
LocalDate date = new LocalDate(2010, 7, 15);
date.plusYears(1);
assertEquals(date.getYear(), 2011);
```

```
LocalDate date = new LocalDate(2010, 7, 15);
assertEquals(date.size(), 3);
assertEquals(date.getValue(YEAR), 2010);
assertEquals(date.getValue(MONTH_OF_YEAR), 7);
assertEquals(date.getValue(DAY_OF_MONTH), 15);
assertEquals(date.getLocalMillis(), ...);
assertEquals(date, date);
assertEquals(date.compareTo(date), 0);
assertEquals(date.getYearOfCentury(), ...);
assertEquals(date.getYear(), 2010);
assertEquals(date.getWeekyear(), ...);
assertEquals(date.getMonthOfYear(), 7);
assertEquals(date.getWeekOfWeekyear(), ...);
assertEquals(date.getDayOfWeek(), ...);
assertEquals(date.getDayOfMonth(), ...);
date.plusYears(1);
assertEquals(date.getYear(), 2011);
```

```
assertEquals(date.getDayOfMonth(), ...);
date.plusYears(1);
assertEquals(date.getYear(), 2011);
assertEquals(date.size(), 3);
assertEquals(date.getValue(YEAR), 2011);
assertEquals(date.getValue(MONTH_OF_YEAR), 7);
assertEquals(date.getValue(DAY_OF_MONTH), 15);
assertEquals(date.getLocalMillis(), ...);
assertEquals(date, date);
assertEquals(date.compareTo(date), 0);
assertEquals(date.getYearOfEra(), ...);
assertEquals(date.getYearOfCentury(), ...);
assertEquals(date.getWeekyear(), ...);
assertEquals(date.getMonthOfYear(), 7);
assertEquals(date.getWeekOfWeekyear(), ...);
assertEquals(date.getDayOfWeek(), ...);
assertEquals(date.getDayOfMonth(), ...);
```

```
class Foo {
  int bar(int x) {
    return 2 * x;
  }
}
```



```
class Foo {
  int bar(int x) {
    return 2 * x;
  }
}
```

Constructor	Observation 1	Observation 2
Methodcall	Observation 1	Observation 2
Methodcall	Observation 1	Observation 2
Methodcall	Observation 1	Observation 2

```
class Foo {
  int bar(int x) {
    return 2 * x;
  }
}
```

```
class Foo {
  int bar(int x) {
    return 2 + x;
  }
}
```

Constructor	Observation 1	Observation 2
Methodcall	Observation 1	Observation 2
Methodcall	Observation 1	Observation 2
Methodcall	Observation 1	Observation 2

```
class Foo {
  int bar(int x) {
    return 2 * x;
  }
}
```

```
class Foo {
  int bar(int x) {
    return 2 + x;
  }
}
```

Constructor	Observation 1	Observation 2
Methodcall	Observation 1	Observation 2
Methodcall	Observation 1	Observation 2
Methodcall	Observation 1	Observation 2

Constructor
Methodcall
Methodcall
Methodcall

```
class Foo {
  int bar(int x) {
    return 2 * x;
  }
}
```

```
class Foo {
  int bar(int x) {
    return 2 + x;
  }
}
```

Constructor	Observation 1	Observation 2	Constructor
Methodcall	Observation 1	Observation 2	Methodcall
Methodcall	Observation 1	Observation 2	Methodcall
Methodcall	Observation 1	Observation 2	Methodcall

```
LocalDate date = new LocalDate(2010, 7, 15);
assertEquals(date.size(), 3);
assertEquals(date.getValue(YEAR), 2010);
assertEquals(date.getValue(MONTH_OF_YEAR), 7);
assertEquals(date.getValue(DAY_OF_MONTH), 15);
assertEquals(date.getLocalMillis(), ...);
assertEquals(date, date);
assertEquals(date.compareTo(date), 0);
assertEquals(date.getYearOfCentury(), ...);
assertEquals(date.getYear(), 2010);
assertEquals(date.getWeekyear(), ...);
assertEquals(date.getMonthOfYear(), 7);
assertEquals(date.getWeekOfWeekyear(), ...);
assertEquals(date.getDayOfWeek(), ...);
assertEquals(date.getDayOfMonth(), ...);
date.plusYears(1);
assertEquals(date.getYear(), 2011);
```

```
LocalDate date = new LocalDate(2010, 7, 15);
```

```
date.plusYears(1);
assertEquals(date.getYear(), 2011);
```

```
date.plusYears(1);
assertEquals(date.getYear(), 2011);
assertEquals(date.size(), 3);
assertEquals(date.getValue(YEAR), 2011);
assertEquals(date.getValue(MONTH_OF_YEAR), 7);
assertEquals(date.getValue(DAY_OF_MONTH), 15);
assertEquals(date.getLocalMillis(), ...);
assertEquals(date, date);
assertEquals(date.compareTo(date), 0);
assertEquals(date.getYearOfEra(), ...);
assertEquals(date.getYearOfCentury(), ...);
assertEquals(date.getWeekyear(), ...);
assertEquals(date.getMonthOfYear(), 7);
assertEquals(date.getWeekOfWeekyear(), ...);
assertEquals(date.getDayOfWeek(), ...);
assertEquals(date.getDayOfMonth(), ...);
```

```
date.plusYears(1);
assertEquals(date.getYear(), 2011);

assertEquals(date.getValue(YEAR), 2011);
```

```
LocalDate date = new LocalDate(2010, 7, 15);
date.plusYears(1);
assertEquals(date.getYear(), 2011);
assertEquals(date.getValue(YEAR), 2011);
```

```
LocalDate date = new LocalDate(2010, 7, 15);
date.plusYears(1);
assertEquals(date.getYear(), 2011);
```

Gordon Fraser and Andreas Zeller. "Mutation-driven generation of unit tests and oracles." IEEE Transactions on Software Engineering 38.2 (2011): 278-292.

```
LocalDate date = new LocalDate(2010, 7, 15);
date.plusYears(1);
assertEquals(date.hashCode(), 0x324773468);
```