Test, beauty, cleanness

d’après le cours de
Alexandre Bergel
abergel@dcc.uchile.cl
feb. 2012

Problem description

We’d like to build a 2D graphic library for editing structured drawing

- with *Widgets* such as circle and rectangle

- with *Operations* such as translate, scale

What are the responsibilities?

Containing the widgets

Modeling the widgets

Applying the operations

Version 1

Create a canvas that contains widgets

Testing the canvas

```
public class HotDrawTest {
    @Test public void testEmptyCanvas() {
        Canvas canvas = new Canvas ();
        assertEquals(canvas.getNumberOfElements(), 0);
    }
}
```

The class Canvas is not created yet!

Creating the class Canvas

```
    public class Canvas {
    public Object getNumberOfElements() {
        return 0;
    }
}
```

Introducing the containment

We need to be able to add objects in a canvas!

```
    @Test public void testCanvas() {
    Canvas canvas = new Canvas ();

    canvas.add(new Object());
    assertEquals(1, canvas.getNumberOfElements());

    canvas.add(new Object());
    canvas.add(new Object());
    assertEquals(3, canvas.getNumberOfElements());
}
```

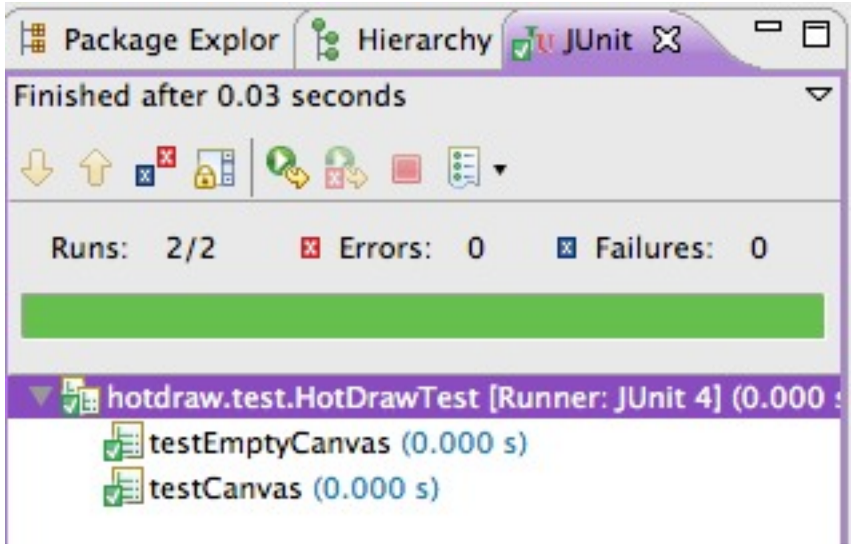
Revising the definition of Canvas

```
    public class Canvas {
    private ArrayList<Object> elements =
              new ArrayList<Object>();

    public int getNumberOfElements() {
        return elements.size();
    }

    public void add(Object object) {
        elements.add(object);
    }
}
```

Revising the definition of Canvas

```
public class Canvas {
   private ArrayList<Object> elements =
                new ArrayList<Object>();

   public int getNumberOfElements() {
      return elements.size();
   }

   public void add(Object object) {
      elements.add(object);
   }
}
```

Tests are green!



```
Finished after 0.03 seconds

Runs:  2/2     Errors:  0     Failures:  0

hotdraw.test.HotDrawTest [Runner: JUnit 4] (0.000
   testEmptyCanvas (0.000 s)
   testCanvas (0.000 s)
```

Version 2

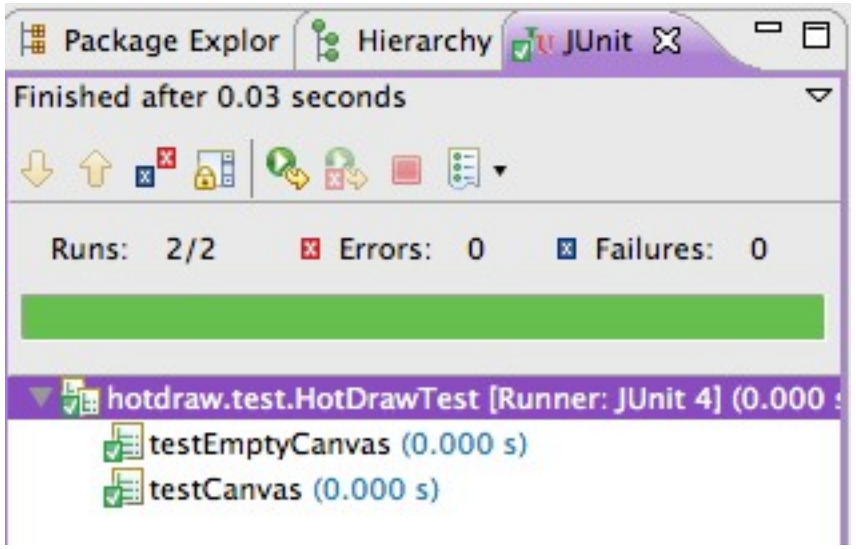
Introducing some widgets

We revise our testCanvas

```
    @Test public void testCanvas() {
    Canvas canvas = new Canvas ();

    canvas.add(new Circle());
    assertEquals(1, canvas.getNumberOfElements());

    canvas.add(new Circle());
    canvas.add(new Rectangle());
    assertEquals(3, canvas.getNumberOfElements());
}
```

Circle and Rectangle

```
    public class Circle {
}

public class Rectangle {
}
```



Package Explor | Hierarchy | JUnit

Finished after 0.03 seconds

Runs: 2/2   Errors: 0   Failures: 0

hotdraw.test.HotDrawTest [Runner: JUnit 4] (0.000

testEmptyCanvas (0.000 s)

testCanvas (0.000 s)

Adding position to circle and rectangle

```
    @Test public void testCanvas() {
    Canvas canvas = new Canvas ();

    //(10, 20), radius 5
    canvas.add(new Circle(10,20, 5));
    assertEquals(1, canvas.getNumberOfElements());

    canvas.add(new Circle());

    //(5,6) -> (10,8)
    canvas.add(new Rectangle(5, 6, 10, 8));
    assertEquals(3, canvas.getNumberOfElements());
}
```

Generated template

```
    public class Circle {

    public Circle(int i, int j, int k) {
        // TODO Auto-generated constructor stub
    }
}
```

Generated template

```
    public class Rectangle {

    public Rectangle(int i, int j, int k, int l) {
        // TODO Auto-generated constructor stub
    }
}
```

Filling the template

```
    public class Rectangle {
    private int x1, y1, x2, y2;

    public Rectangle() {
        this(2, 3, 5, 6);
    }

    public Rectangle(int x1, int y1, int x2, int y2) {
        this.x1 = x1;
        this.y1 = y1;
        this.x2 = x2;
        this.y2 = y2;
    }
}
```

Version 3

Before moving on, lets step back on what we wrote to see whether there are opportunities for *cleaning* a bit the code

abergel@dcc.uchile.cl

HotDrawTest

```
public class HotDrawTest {

   @Test public void testEmptyCanvas() {
       Canvas canvas = new Canvas ();
       ...
    }

   @Test public void testCanvas() {
       Canvas canvas = new Canvas ();
       ...
   }
}
```

HotDrawTest

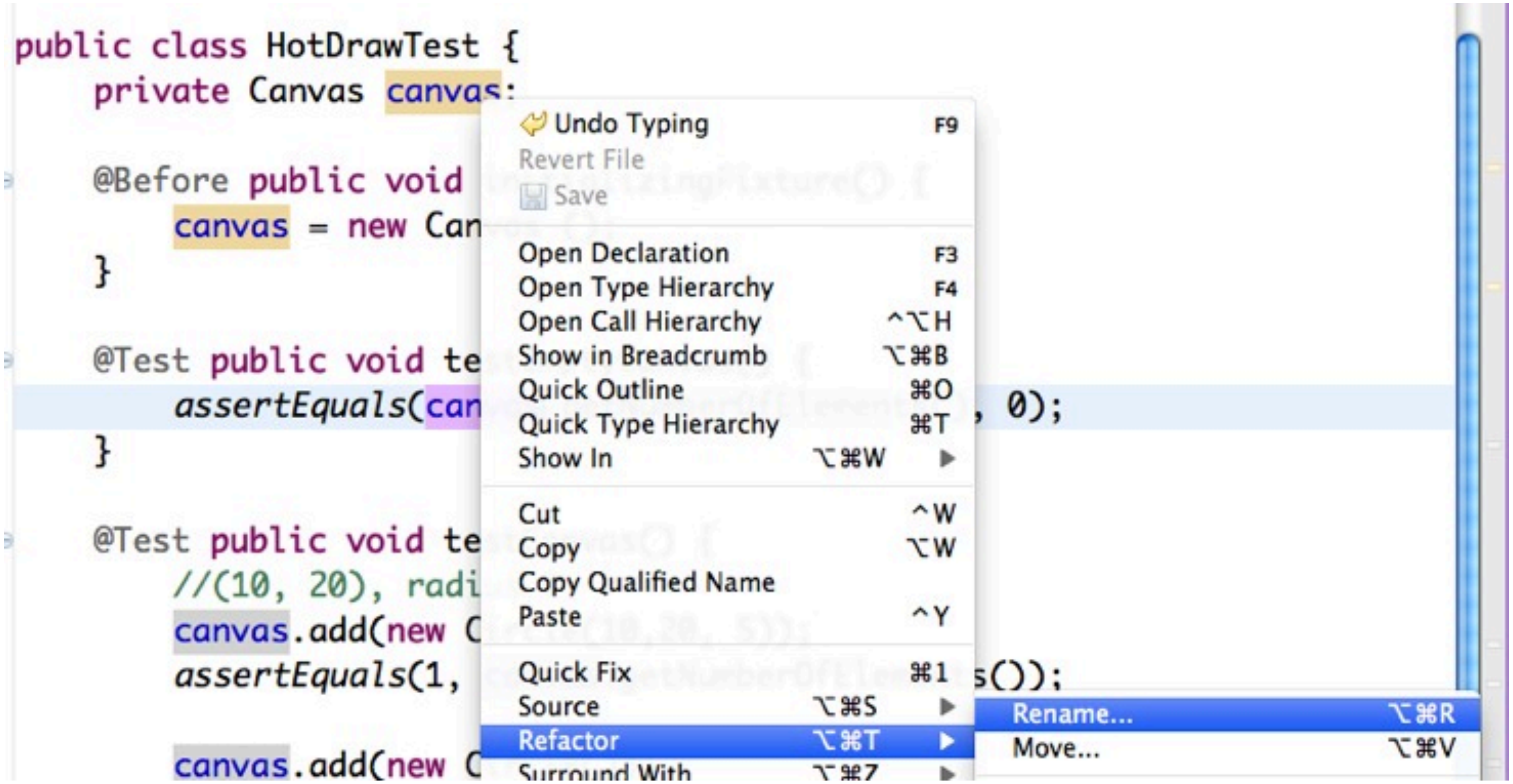
```
public class HotDrawTest {

   @Test public void testEmptyCanvas() {
       Canvas canvas = new Canvas ();
       ...
    }

   @Test public void testCanvas() {
       Canvas canvas = new Canvas ();
       ...
   }
}
```

Duplication!

Refactoring our test

```
public class HotDrawTest {
private Canvas canvas;

@Before public void initializingFixture() {
    canvas = new Canvas ();
}

@Test public void testEmptyCanvas() {
    assertEquals(canvas.getNumberOfElements(), 0);
}

@Test public void testCanvas() {
    //(10, 20), radius 5
    canvas.add(new Circle(10,20, 5));
    ...
   }
}
```

Giving a better name to the variable



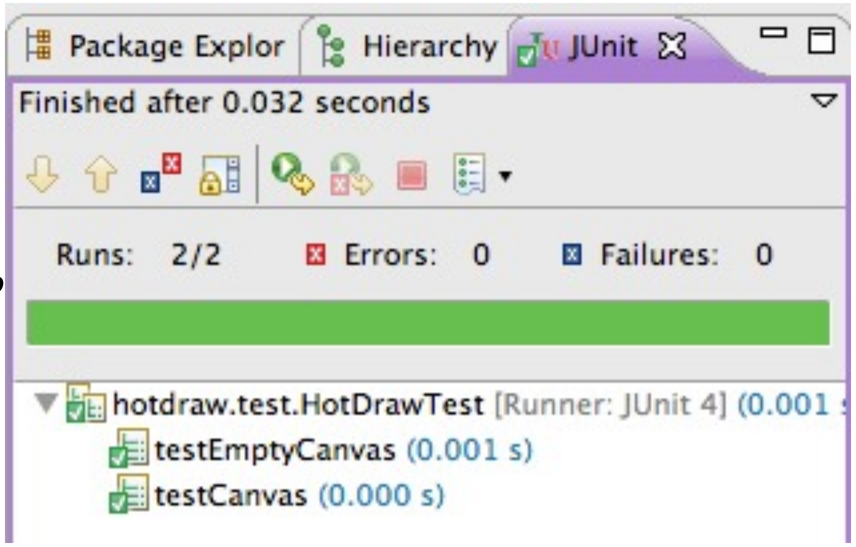
```
public class HotDrawTest {
    private Canvas canvas;

    @Before public void
        canvas = new Can
    }

    @Test public void te
        assertEquals(can                               , 0);
    }

    @Test public void te
        //(10, 20), radi
        canvas.add(new C
        assertEquals(1,                                s());

        canvas.add(new C
```

canvas -> emptyCanvas

```
public class HotDrawTest {
  private Canvas emptyCanvas;

  @Before public void initializingFixture() {
      emptyCanvas = new Canvas ();
  }

  @Test public void testEmptyCanvas() {
      assertEquals(emptyCanvas.getNumberOfElements(), 0);
  }

  @Test public void testCanvas() {
      //(10, 20), radius 5
      emptyCanvas.add(new Circle(10,20, 5));
      ...
  }
}
```

canvas -> emptyCanvas

```
public class HotDrawTest {
    private Canvas emptyCanvas;

    @Before public void initializingFixture() {
        emptyCanvas = new Canvas ();
    }

    @Test public void testEmptyCanvas() {
        assertEquals(emptyCanvas.getNumberOfElements(), 0);
    }

    @Test public void testCanvas() {
        //(10, 20), radius 5
        emptyCanvas.add(new Circle(10,
        ...
    }
}
```


Finished after 0.032 seconds

Runs: 2/2   Errors: 0   Failures: 0

```
hotdraw.test.HotDrawTest [Runner: JUnit 4] (0.001
    testEmptyCanvas (0.001 s)
    testCanvas (0.000 s)
```

Version 4

Applying the operations on the widget

Note that at that point, *we have not seen the need* to have a *common superclass* for Circle and Rectangle

As well *we have not seen the need* to have a *common interface*

We should be test driven, else it is too easy to go wrong

The class canvas also contains *a list of objects*

Adding an operation

Let’s translate our objects

Each widget should now understand the message **translate(deltaX, deltaY)**

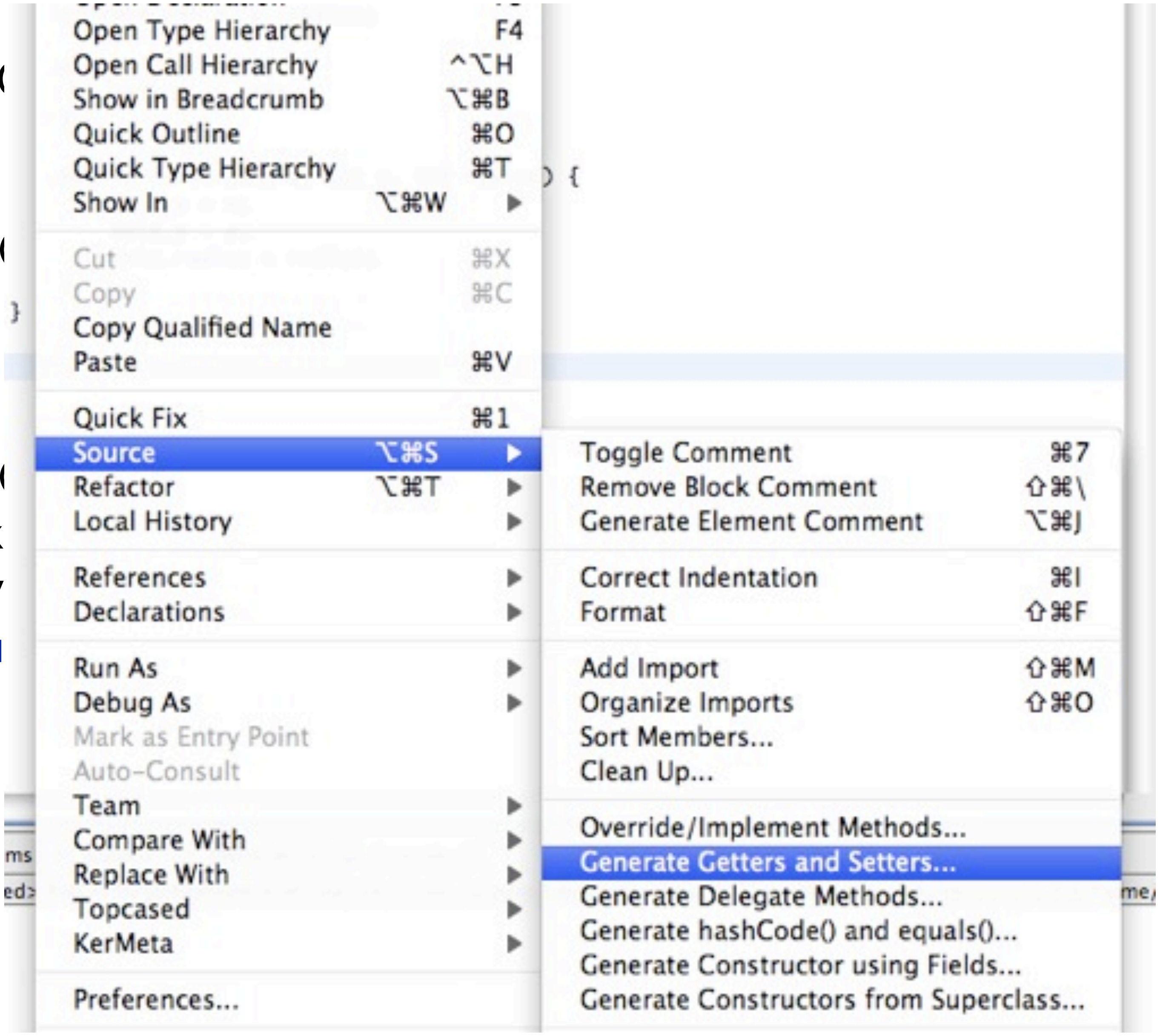
Let’s write some test first

Testing circle first

```
    @Test public void translatingCircle() {
    Circle circle = new Circle();
    int oldX = circle.getX();
    int oldY = circle.getY();

    circle.translate(2, 3);
    assertEquals(circle.getX(), oldX + 2);
    assertEquals(circle.getY(), oldY + 3);
}
```

Generate Getters

```
    public class (
    private int x

    public Circle(
        this(5, 5,
    }

    public Circle(
        this.x = x
        this.y = y
        this.radiu
    }
}
```



Modifying Circle

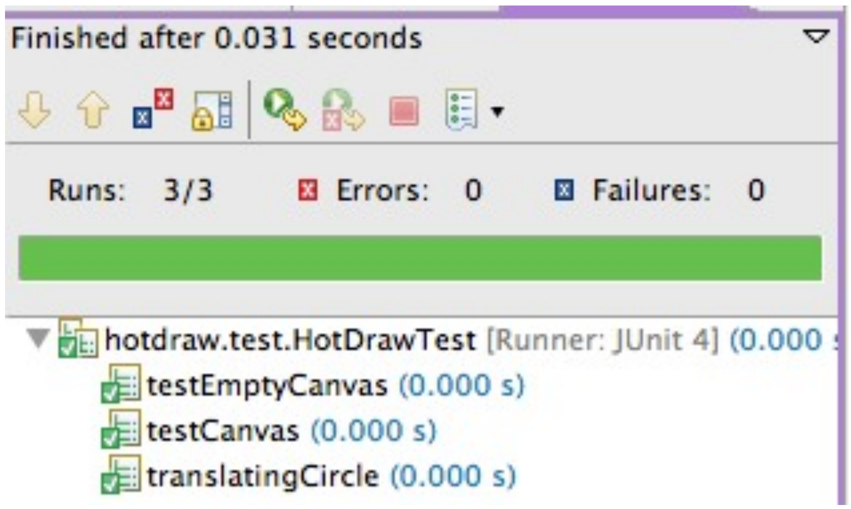
```
    public class Circle {
    private int x, y, radius;

    public int getX() {
        return x;
    }

    public int getY() {
        return y;
    }
    ...
}
// Note that there is no accessor for radius, we have not
seen the need of it!
```

Translating Circle

```
    public class Circle {
    ...
    public void translate(int dx, int dy) {
        x = x + dx;
        y = y + dy;
    }
    ...
}
```

Translating Circle

```
public class Circle {
    ...
    public void translate(int dx, int dy) {
        x = x + dx;
        y = y + dy;
    }
    ...
}
```



```
Finished after 0.031 seconds

Runs:   3/3      Errors:   0      Failures:   0

hotdraw.test.HotDrawTest [Runner: JUnit 4] (0.000
    testEmptyCanvas (0.000 s)
    testCanvas (0.000 s)
    translatingCircle (0.000 s)
```

Translating the rectangle

```
    @Test public void translatingRectangle() {
    Rectangle rectangle = new Rectangle();
    int oldX1 = rectangle.getX1();
    int oldY1 = rectangle.getY1();
    int oldX2 = rectangle.getX2();
    int oldY2 = rectangle.getY2();

    rectangle.translate(2, 3);
    assertEquals(rectangle.getX1(), oldX1 + 2);
    assertEquals(rectangle.getX2(), oldX2 + 2);
    assertEquals(rectangle.getY1(), oldY1 + 3);
    assertEquals(rectangle.getY2(), oldY2 + 3);
}
```

Updating Rectangle

```
    public class Rectangle {
    ...
    public int getX1() {...}
    public int getY1() {...}
    public int getX2() {...}
    public int getY2() {
        return y2;
    }

    public void translate(int dx, int dy) {
        x1 = x1 + dx;
        x2 = x2 + dx;
        y1 = y1 + dy;
        y2 = y2 + dy;
    }
}
```

Important

Note that we have not still see the need to have a common interface and a common superclass

If you doing it upfront, when *your design will look like what you want it to be, and not what it has to be*

Version 5

It is a bit cumbersome to have to translate each element one by one

Let’s ask the canvas to translate all the nodes

Translating the canvas

```
@Test public void translatingTheCanvas() {

    Rectangle rectangle = new Rectangle();
    int rectangleOldX1 = rectangle.getX1();
    int rectangleOldY1 = rectangle.getY1();
    int rectangleOldX2 = rectangle.getX2();
    int rectangleOldY2 = rectangle.getY2();

    Circle circle = new Circle();
    int circleOldX = circle.getX();
    int circleOldY = circle.getY();

    emptyCanvas.add(rectangle);
    emptyCanvas.add(circle);
    emptyCanvas.translate(2, 3);
    ...
```

Translating the canvas

```
        ...

    assertEquals(rectangle.getX1(), rectangleOldX1 + 2);
    assertEquals(rectangle.getX2(), rectangleOldX2 + 2);
    assertEquals(rectangle.getY1(), rectangleOldY1 + 3);
    assertEquals(rectangle.getY2(), rectangleOldY2 + 3);
    assertEquals(circle.getX(), circleOldX + 2);
    assertEquals(circle.getY(), circleOldY + 3);
}
```

Updating Canvas - what we would like to do

```
public class Canvas {

    private ArrayList<Object> elements =
                new ArrayList<Object>();

    public void add(Object object) {
        elements.add(object);
    }

    public void translate(int dx, int dy) {
        for(Object o : elements)
            o.translate(dx, dy);
    }
    ...
}
```

abergel@dcc.uchile.cl

Updating Canvas - what we would like to do

```
public class Canvas {

   private ArrayList<Object> elements =
               new ArrayList<Object>();

   public void add(Object object) {
       elements.add(object);
   }

   public void translate(int dx, int dy) {
       for(Object o : elements)
           o.translate(dx, dy);
   }
   ...
}
```

The compiler will not be happy with this

What is happening?

Only now we see the *need* to introduce a *common interface* that the object have to fulfill

This interface will only be aware of the translate(dx,dy) method

Let’s introduce the Widget interface

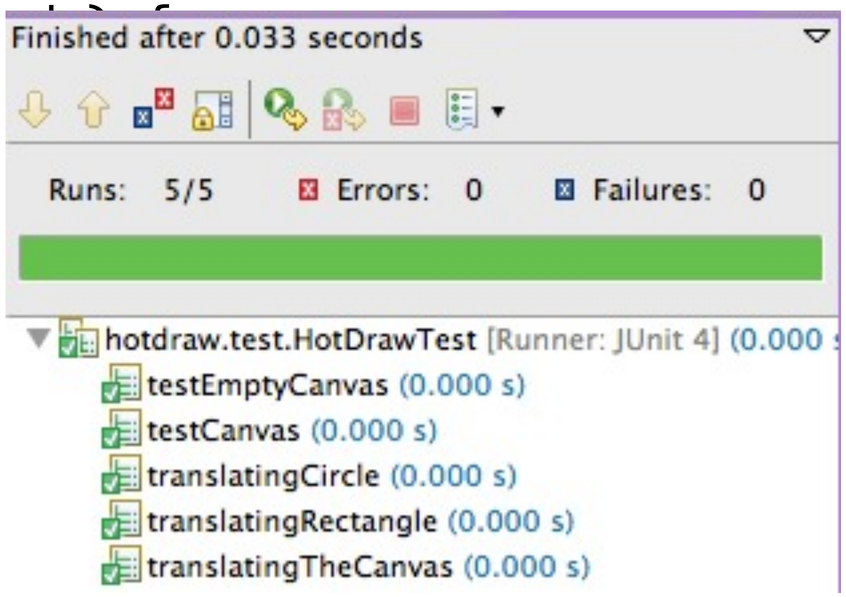
```
    public interface Widget {
    public void translate(int dx, int dy);
}

public class Rectangle implements Widget {
    ...
}

public class Circle implements Widget {
    ...
}
```

Updating Canvas

```
    public class Canvas {
    private ArrayList<Widget> elements =
              new ArrayList<Widget>();

    public void add(Widget widget) {
        elements.add(widget);
    }

    public void translate(int dx, int dy) {
        for(Widget o : elements)
            o.translate(dx, dy);
    }
    ...
}
```

Updating Canvas

```
public class Canvas {
    private ArrayList<Widget> elements =
                new ArrayList<Widget>();

    public void add(Widget widget) {
        elements.add(widget);
    }

    public void translate(int dx, int
        for(Widget o : elements)
            o.translate(dx, dy);
    }
    ...
}
```



```
Finished after 0.033 seconds

Runs:  5/5     Errors:  0     Failures:  0

hotdraw.test.HotDrawTest [Runner: JUnit 4] (0.000
    testEmptyCanvas (0.000 s)
    testCanvas (0.000 s)
    translatingCircle (0.000 s)
    translatingRectangle (0.000 s)
    translatingTheCanvas (0.000 s)
```

Version 6

We are doing a pretty good job so far

Let’s add a group of widgets that can be commonly manipulated

abergel@dcc.uchile.cl

Testing Group

```
    @Test public void groupingWidgets() {
    Group group = new Group();
    assertEquals(group.getNumberOfElements(), 0);

    group.add(new Circle());
    group.add(new Rectangle());
    assertEquals(group.getNumberOfElements(), 2);
}
```

Defining Group

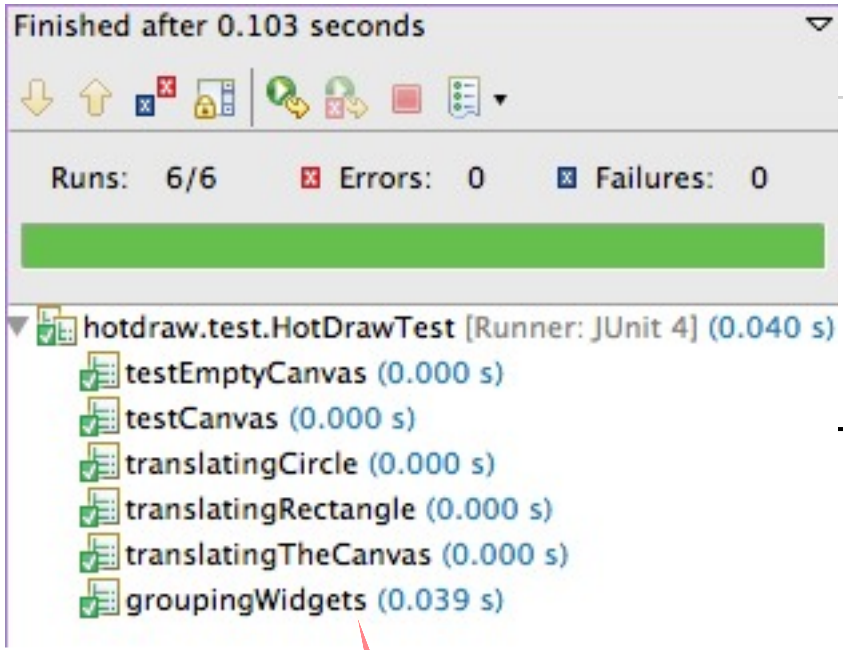
```
    public class Group {
        private ArrayList<Object> elements =
                      new ArrayList<Object>();

        public void add(Object w) {
            elements.add(w);
        }

        public int getNumberOfElements() {
            return elements.size();
        }
}
```

Defining Group

```
public class Group {
    private ArrayList<Object> elements =
                new ArrayList<Object>();

    public void add(Object w) {
        elements.add(w);
    }

    public int getNumb
        return elements.
    }
}
```

Yes! We haven’t seen the need to have Widget here

Defining Group



```
Finished after 0.103 seconds
Runs:   6/6      Errors:   0      Failures:   0
hotdraw.test.HotDrawTest [Runner: JUnit 4] (0.040 s)
    testEmptyCanvas (0.000 s)
    testCanvas (0.000 s)
    translatingCircle (0.000 s)
    translatingRectangle (0.000 s)
    translatingTheCanvas (0.000 s)
    groupingWidgets (0.039 s)
```

```
ject> elements =
.ist<Object>();

ct w) {

    }

    publi  int getNumb
        re urn elements.
    }
```

Yes! We haven't seen the need to have Widget here

This is the proof that we do not need it!

Translating a group - what we could write, but it contains a lot of duplication

```
    @Test public void translatingGroup() {
    Group group = new Group();
    group.add(new Circle());
    group.add(new Rectangle());
    group.translate(...)
}
```

But let’s refactor first

```
public class HotDrawTest {
private Canvas emptyCanvas;
private Group emptyGroup, group;
private Circle circle;
private Rectangle rectangle;

@Before public void initializingFixture() {
    emptyCanvas = new Canvas ();
    emptyGroup = new Group();
    group = new Group();
    group.add(circle = new Circle());
    group.add(rectangle = new Rectangle());
}
```

But let’s refactor first

```
    @Test public void groupingWidgets() {
    assertEquals(emptyGroup.getNumberOfElements(), 0);
    assertEquals(group.getNumberOfElements(), 2);
}
```

Translating a group

```
    @Test public void translatingGroup() {
    int circleOldX = circle.getX();
    int circleOldY = circle.getY();
    int rOldX1 = rectangle.getX1();
    int rOldY1 = rectangle.getY1();

    group.translate(2, 3);

    assertEquals(rectangle.getX1(), rOldX1 + 2);
    assertEquals(rectangle.getY1(), rOldY1 + 3);
    assertEquals(circle.getX(), circleOldX + 2);
    assertEquals(circle.getY(), circleOldY + 3);

}
```

Translating a group

```
    public class Group {
    private ArrayList<Widget> elements =
              new ArrayList<Widget>();

    public void add(Widget w) {
        elements.add(w);
    }

    public int getNumberOfElements() {
        return elements.size();
    }

    public void translate(int i, int j) {
        for(Widget w : elements)
            w.translate(i, j);
    }
}
```

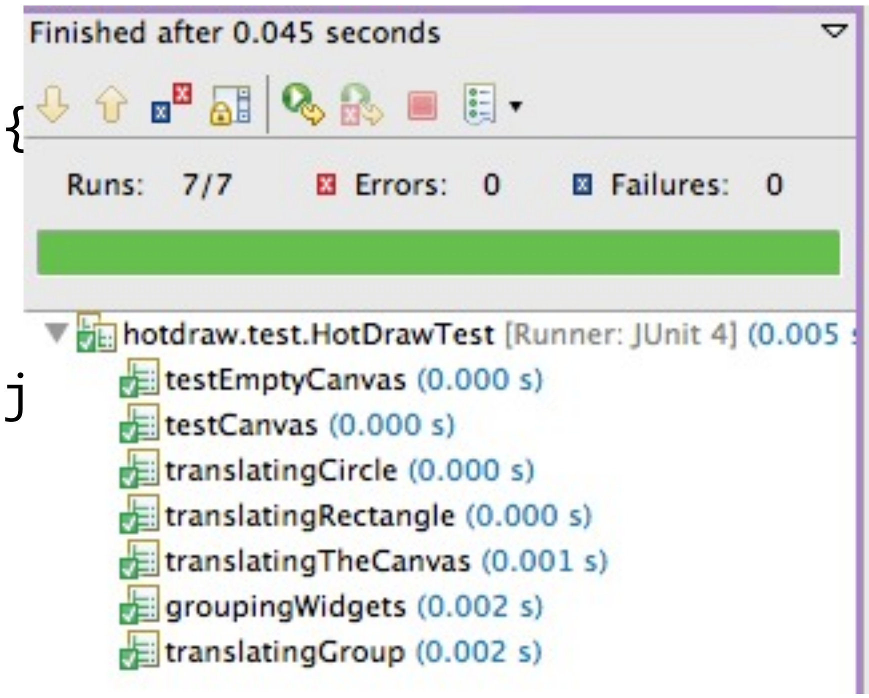
Translating a group

```
public class Group {
    private ArrayList<Widget> elements =
                new ArrayList<Widget>();

    public void add(Widget w) {
        elements.add(w);
    }

    public int getNumberOfEle
        return elements.size();
    }

    public void translate(int i, int j) {
        for(Widget w : elements)
            w.translate(i, j);
    }
}
```

Yes, we need an array of Widgets

Translating a group

```
public class Group {
    private ArrayList<Widget> elements =
                new ArrayList<Widget>();

    public void add(Widget w) {
        elements.add(w);
    }

    public int getNumberOfElements() {
        return elements.size();
    }

    public void translate(int i, int j
        for(Widget w : elements)
            w.translate(i, j);
    }
}
```



```
Finished after 0.045 seconds

Runs:   7/7        Errors:   0        Failures:   0

▼ hotdraw.test.HotDrawTest [Runner: JUnit 4] (0.005
    testEmptyCanvas (0.000 s)
    testCanvas (0.000 s)
    translatingCircle (0.000 s)
    translatingRectangle (0.000 s)
    translatingTheCanvas (0.001 s)
    groupingWidgets (0.002 s)
    translatingGroup (0.002 s)
```

Version 7

Let’s refactor Canvas

instead of containing a list of elements, it will solely contain a group

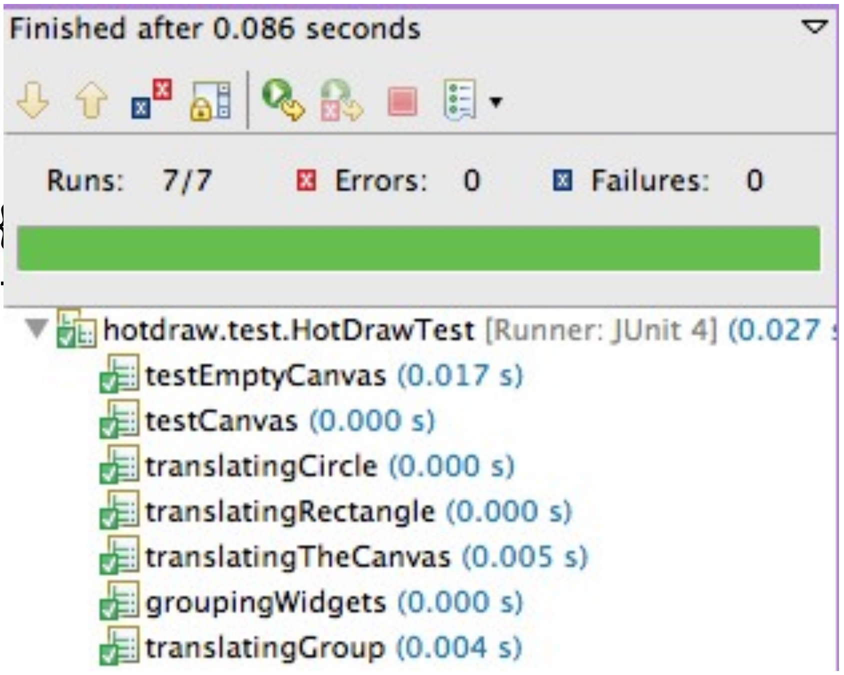
abergel@dcc.uchile.cl

Canvas is getting simpler

```
    public class Canvas {
    private Group group = new Group();

    public void add(Widget widget) {
        group.add(widget);
    }

    public void translate(int dx, int dy) {
        group.translate(dx, dy);
    }

    public int getNumberOfElements() {
        return group.getNumberOfElements();
    }
}
```

Canvas is getting simpler

```
public class Canvas {
    private Group group = new Group();

    public void add(Widget widget) {
        group.add(widget);
    }

    public void translate(int dx, int dy) {
        group.translate(dx, dy);
    }

    public int getNumberOfElements() {
        return group.getNumberOfElemen
    }
}
```

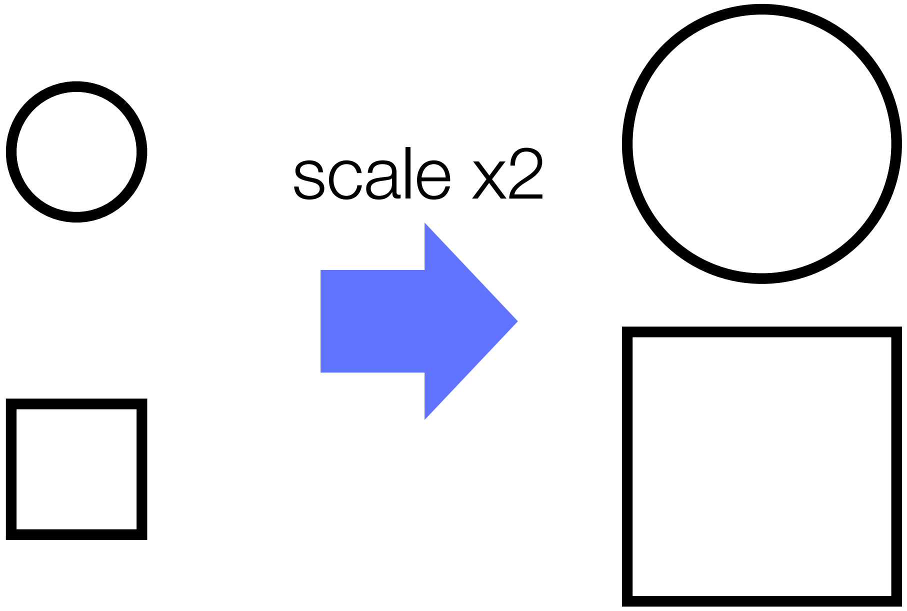


```
Finished after 0.086 seconds

Runs:  7/7     Errors:  0     Failures:  0

hotdraw.test.HotDrawTest [Runner: JUnit 4] (0.027
    testEmptyCanvas (0.017 s)
    testCanvas (0.000 s)
    translatingCircle (0.000 s)
    translatingRectangle (0.000 s)
    translatingTheCanvas (0.005 s)
    groupingWidgets (0.000 s)
    translatingGroup (0.004 s)
```

Version 8

Adding a new operation

We will now scale objects



abergel@dcc.uchile.cl

Adding a test for scalability

```
    @Test public void scalingGroup() {
    int oldRadius = circle.radius();
    int rectangleWidth = rectangle.width();
    int rectangleHeight = rectangle.height();

    group.scale(2);

    assertEquals(circle.radius(), 2 * oldRadius);
    assertEquals(rectangle.width(), 2 * rectangleWidth);
    assertEquals(rectangle.height(), 2 * rectangleHeight);
}
```

Adding a test for scalability

Accessing radius

```
@Test public void scalingGroup() {
    int oldRadius = circle.radius();
    int rectangleWidth = rectangle.width();
    int rectangleHeight = rectangle.height();

    group.scale(2);

    assertEquals(circle.radius(), 2 * oldRad
    assertEquals(rectangle.width(), 2 * rectangleW      ),
    assertEquals(rectangle.height(), 2 * rectangleHeight);
}
```

Accessing width and height

Updating Circle

```
    public class Circle implements Widget {
    private int x, y, radius;

    public int radius() {
        return radius;
    }
    ...
}
```

Updating Rectangle

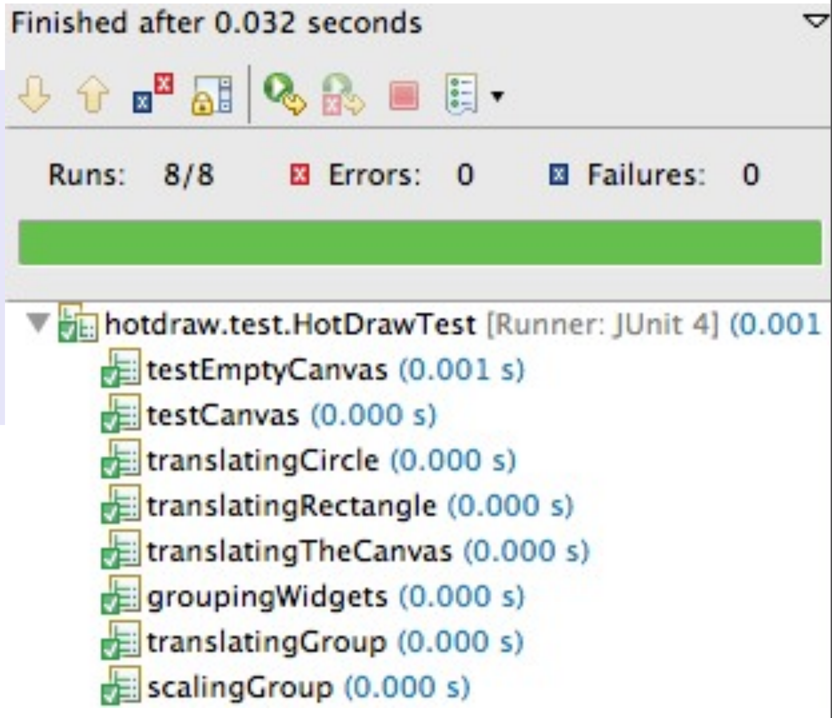
```
public class Rectangle implements Widget {
public int width() {
    return Math.abs(x2 - x1);
}

public int height() {
    return Math.abs(y2 - y1);
}
...
}
```

Scalability

```
    public class Group {
    public void scale(double s) {
        for(Widget w : elements)
            w.scale(s);
    }
   ... }

public interface Widget {
   ...
    public void scale(double s); }

public class Circle implements Widget {
    public void scale(double s) {
        radius *= s;
    }
}

public class Rectangle implements Widget {
    public void scale(double s) {
        x1 *= s; y1 *= s; x2 *= s; y2 *= s;
    }
}
```

Scalability

```
public class Group {
    public void scale(double s) {
        for(Widget w : elements)
            w.scale(s);
    }
   ... }

public interface Widget {
   ...
    public void scale(double s); }

public class Circle implements Widget {
    public void scale(double s) {
        radius *= s;
    }
}

public class Rectangle implements Widget {
    public void scale(double s) {
        x1 *= s; y1 *= s; x2 *= s; y2 *= s;
    }
}
```



```
Finished after 0.032 seconds

Runs:  8/8     Errors:  0     Failures:  0

hotdraw.test.HotDrawTest [Runner: JUnit 4] (0.001
    testEmptyCanvas (0.001 s)
    testCanvas (0.000 s)
    translatingCircle (0.000 s)
    translatingRectangle (0.000 s)
    translatingTheCanvas (0.000 s)
    groupingWidgets (0.000 s)
    translatingGroup (0.000 s)
    scalingGroup (0.000 s)
```

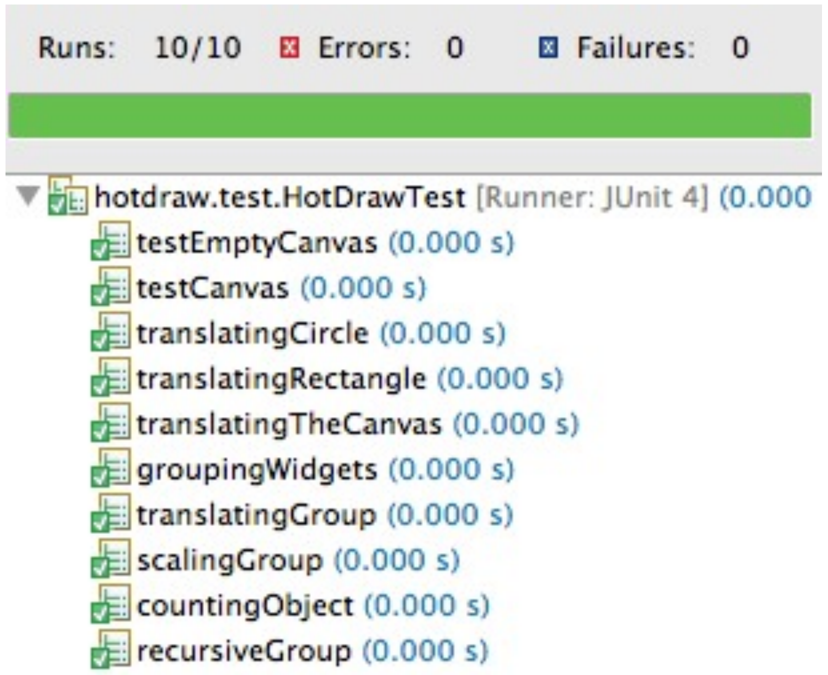
Recursive group

```
    @Test public void recursiveGroup() {
    Group unGroup = new Group();
    unGroup.add(emptyGroup);
    group.add(unGroup);
    assertEquals(emptyGroup.getNumberOfElements(), 0);
    assertEquals(unGroup.getNumberOfElements(), 1);
    assertEquals(group.getNumberOfElements(), 3);
}
```

```
group = new Group();
group.add(circle = new Circle());
group.add(rectangle = new Rectangle());
```

Group implement Widget

```
    public class Group implements Widget {
    ...
}
```


Runs: 10/10 Errors: 0 Failures: 0

```
hotdraw.test.HotDrawTest [Runner: JUnit 4] (0.000
    testEmptyCanvas (0.000 s)
    testCanvas (0.000 s)
    translatingCircle (0.000 s)
    translatingRectangle (0.000 s)
    translatingTheCanvas (0.000 s)
    groupingWidgets (0.000 s)
    translatingGroup (0.000 s)
    scalingGroup (0.000 s)
    countingObject (0.000 s)
    recursiveGroup (0.000 s)
```