


```

    ...
}

```

JUnit
가
가
가
(Gamma, E., Applying Design Patterns in Java, in Java Gems, SIGS Reference Library, 1997).
1 TestCase



3.2 in-run ()

```

" "
TestCase 가 TestCase
가
가
가 가
"
가 ( )
가 가

```

```

public void run()
{
    setUp ();
    runTest ();
    tearDown ();
}

```

```
}
```

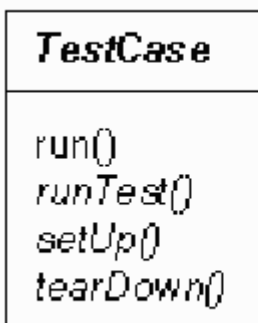
```
protected void runTest ()  
{  
    ...  
}
```

```
protected void setUp ()  
{  
    ...  
}
```

```
protected void tearDown ()  
{  
    ...  
}
```

```
setUp    tearDown
```

2



Template Method

3.3

TestCase가

가

?

가

가

가

TestCase

Smalltalk

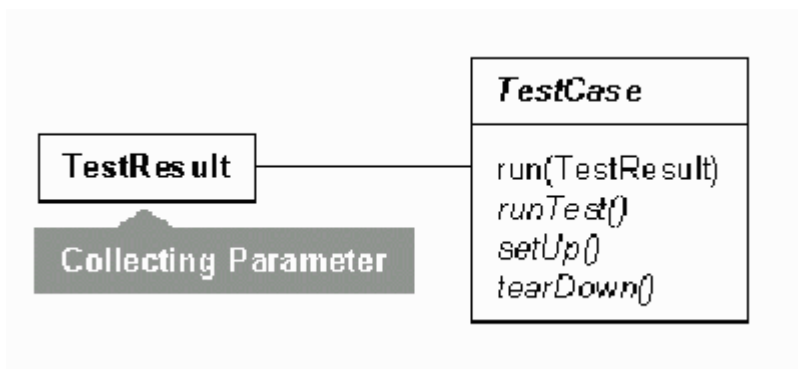
(Beck, K. Smalltalk

, Prentice Hall, 1996

) 가

가

TestResult



가

가

가

JUnit
가

ArrayIndexOutOfBoundsException
AssertionFailedError

가 catch (1)

(2)

```

public void run (TestResult result) {
    result.startTest (this);
    try {
        runTest ();
    }
    catch (AssertionFailedError e) { // 1
        result.addFailure (this, e);
    }
    catch (Throwable e) { // 2
        result.addError (this, e);
    }
    finally {
        tearDown ();
    }
}
    
```

AssertionFailedError TestCase assert
JUnit assert
가

```

protected void assertTrue (boolean condition) {
    if (! condition)
        throw new AssertionFailedError ();
}
    
```

```

AssertionFailedError (TestCase)가
    TestCase.run ()
        AssertionFailedError

```

```

    AssertionFailedError {
    public AssertionFailedError () {}
}

```

```

    . TestResult

```

```

        addError ( , Throwable t) {
    fErrors.addElement (new TestFailure (test, t));
}

```

```

        addFailure ( , Throwable t) {
    fFailures.addElement (new TestFailure (test, t));
}

```

TestFailure

```

    TestFailure Object {
    protected Test fFailedTest
        Throwable fThrownException;
}

```

```

        TestResult 가
        " "
        TestResult
    throw
        MoneyTest
        TestResult 가

```

```

public void testMoneyEquals () {
    assertTrue (! f12CHF.equals (null));
    assertEquals (f12CHF, f12CHF);
    assertEquals (f12CHF, new Money (12, "CHF"));
    assertTrue (! f12CHF.equals (f14CHF));
}

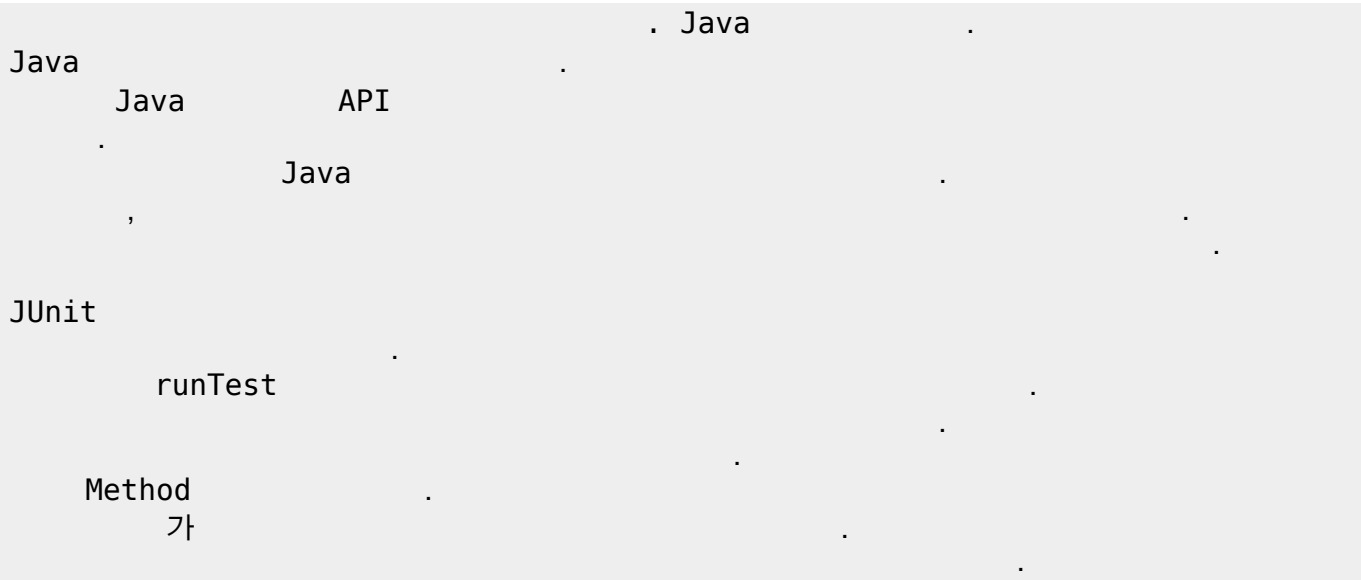
```

```

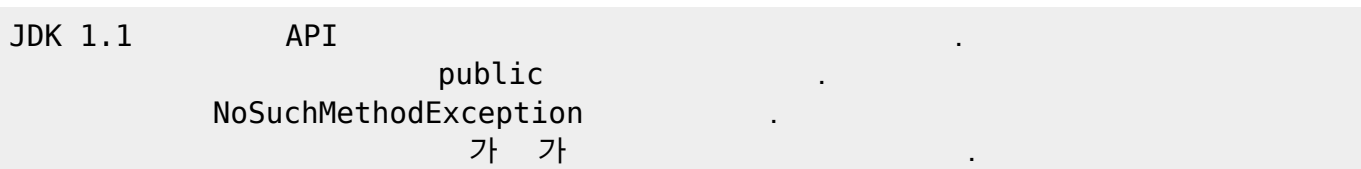
JUnit TestResult
TextTestResult
    UITestResult
JUnit Test Runner
TestResult
    TestResult
HTMLTestResult
    HTML

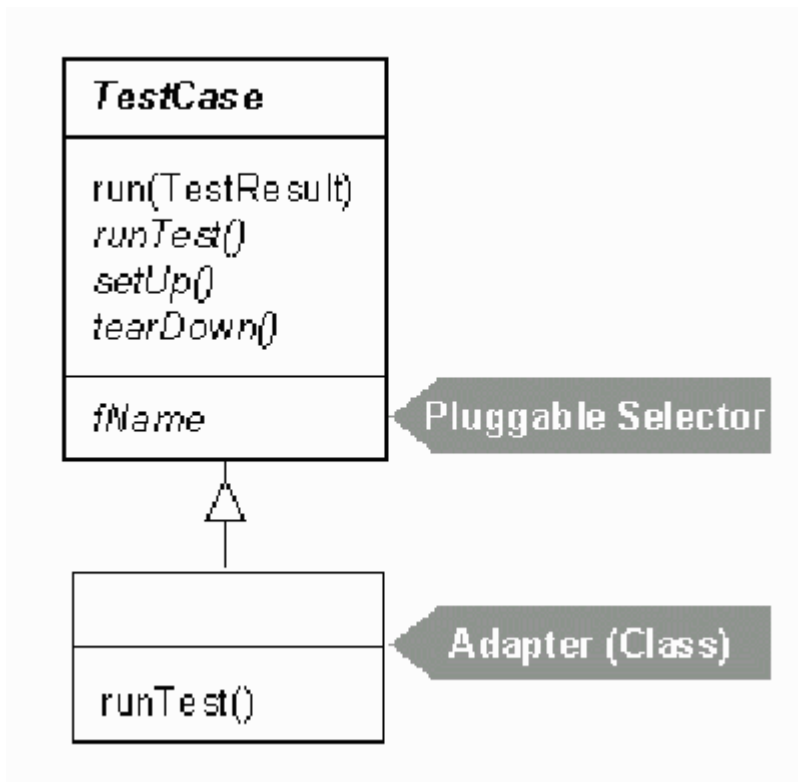
```

3.4 -TestCase

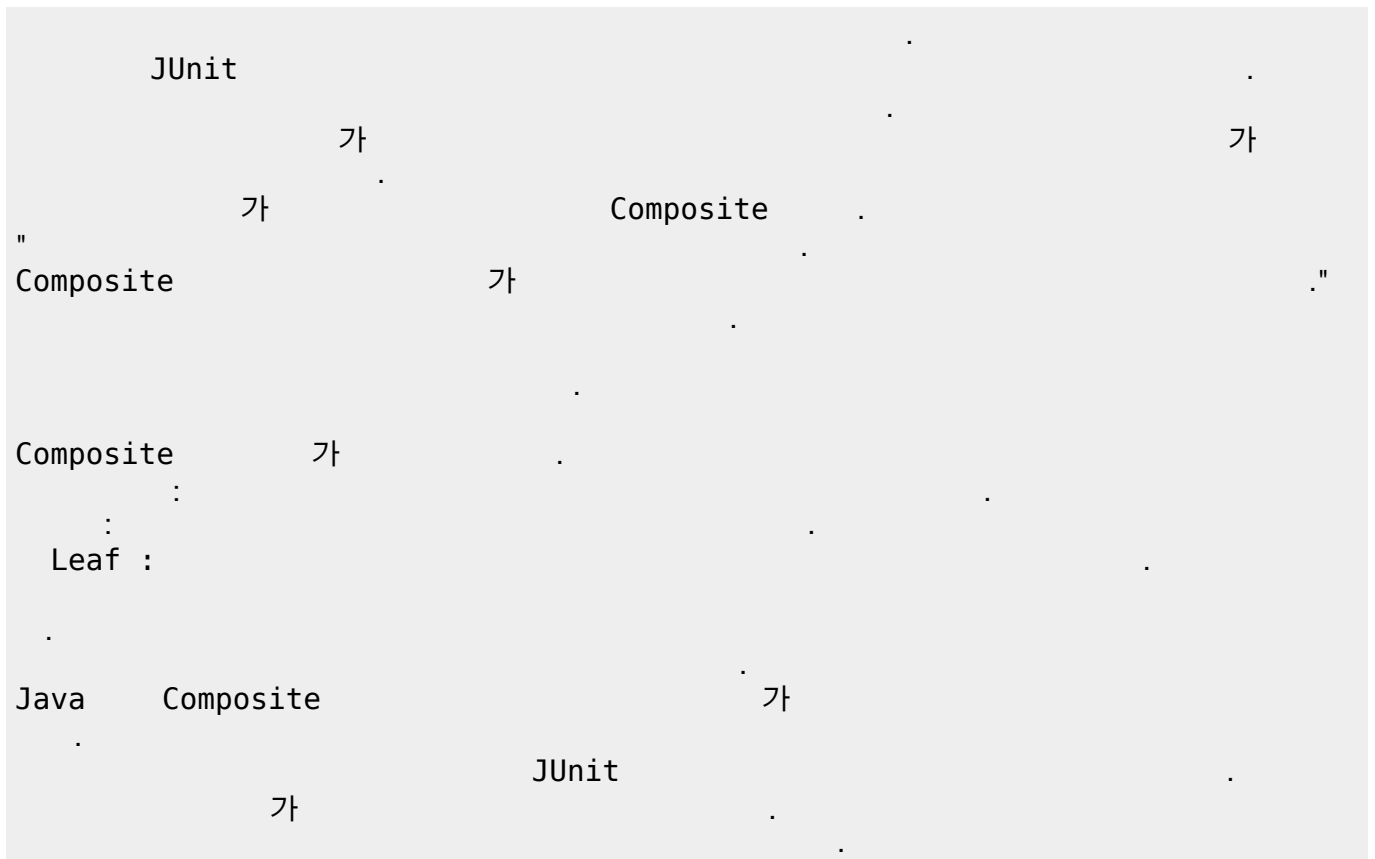


```
protected void runTest () throws Throwable {
    Method runMethod = null;
    try {
        runMethod = getClass (). getMethod ( fName, new Class [0]);
    } catch (NoSuchMethodException e) {
        assertTrue ( "Method \" " + fName + "\" " " ", false);
    }
    try {
        runMethod.invoke (this, new Class [0]);
    }
    // InvocationTargetException   IllegalAccessException
}
```





3.5 -TestSuite



```

public interface Test {
    public abstract void run (TestResult result);
}
    
```

```

TestCase Composite Leaf
가
TestSuite
TestSuite

```

```

TestSuite Test {
private Vector fTests = new Vector ();
}

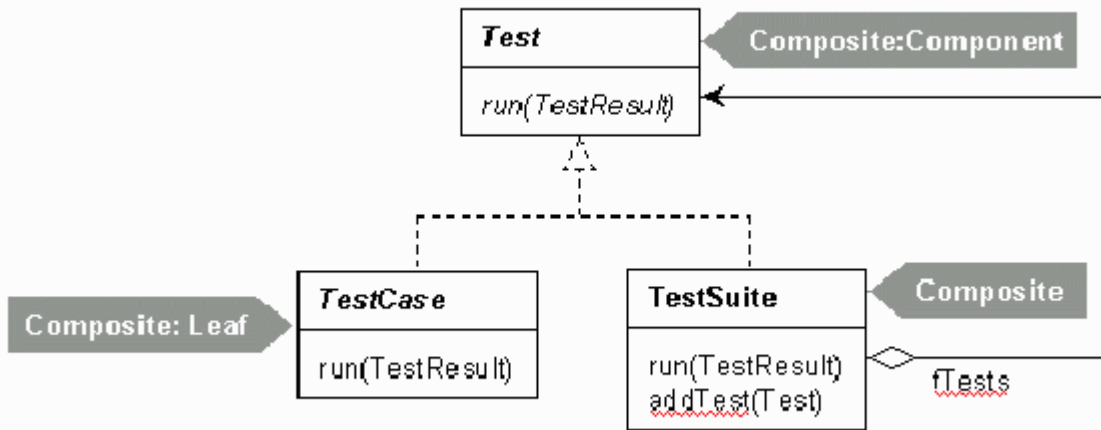
```

```
run ()
```

```

public void run (TestResult result) {
for (Enumeration e = fTests.elements (); e.hasMoreElements ();) {
Test test = (Test) e.nextElement ();
test.run ();
}
}

```



```
가 addTest
```

```

public void addTest ( ) {
fTests.addElement (test);
}

```

```

가
TestCase TestSuite Test
TestSuite
TestSuite
TestSuite

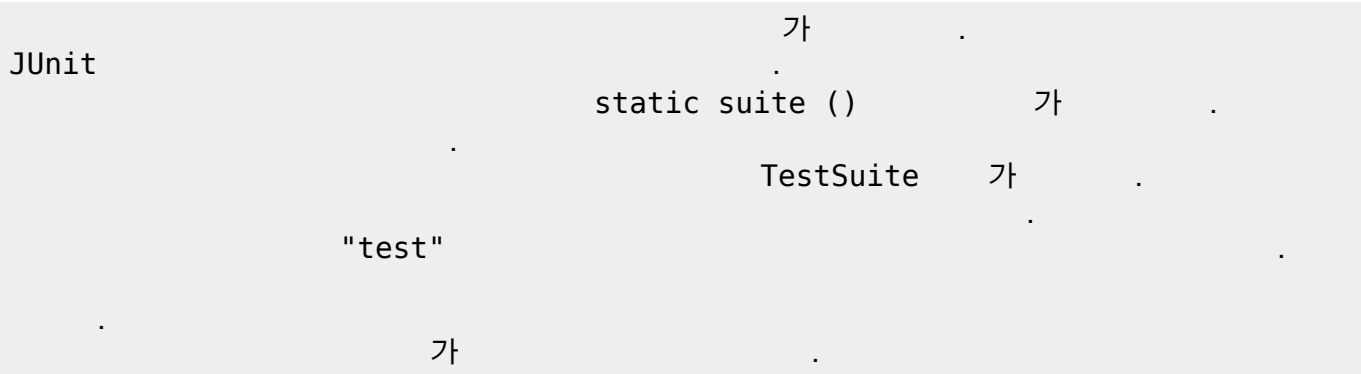
```

```

public static Test suite()
{
TestSuite suite = new TestSuite ();
suite.addTest (new MoneyTest ( "testMoneyEquals"));
}

```

```
suite.addTest (new MoneyTest ( "testSimpleAdd"));
}
```

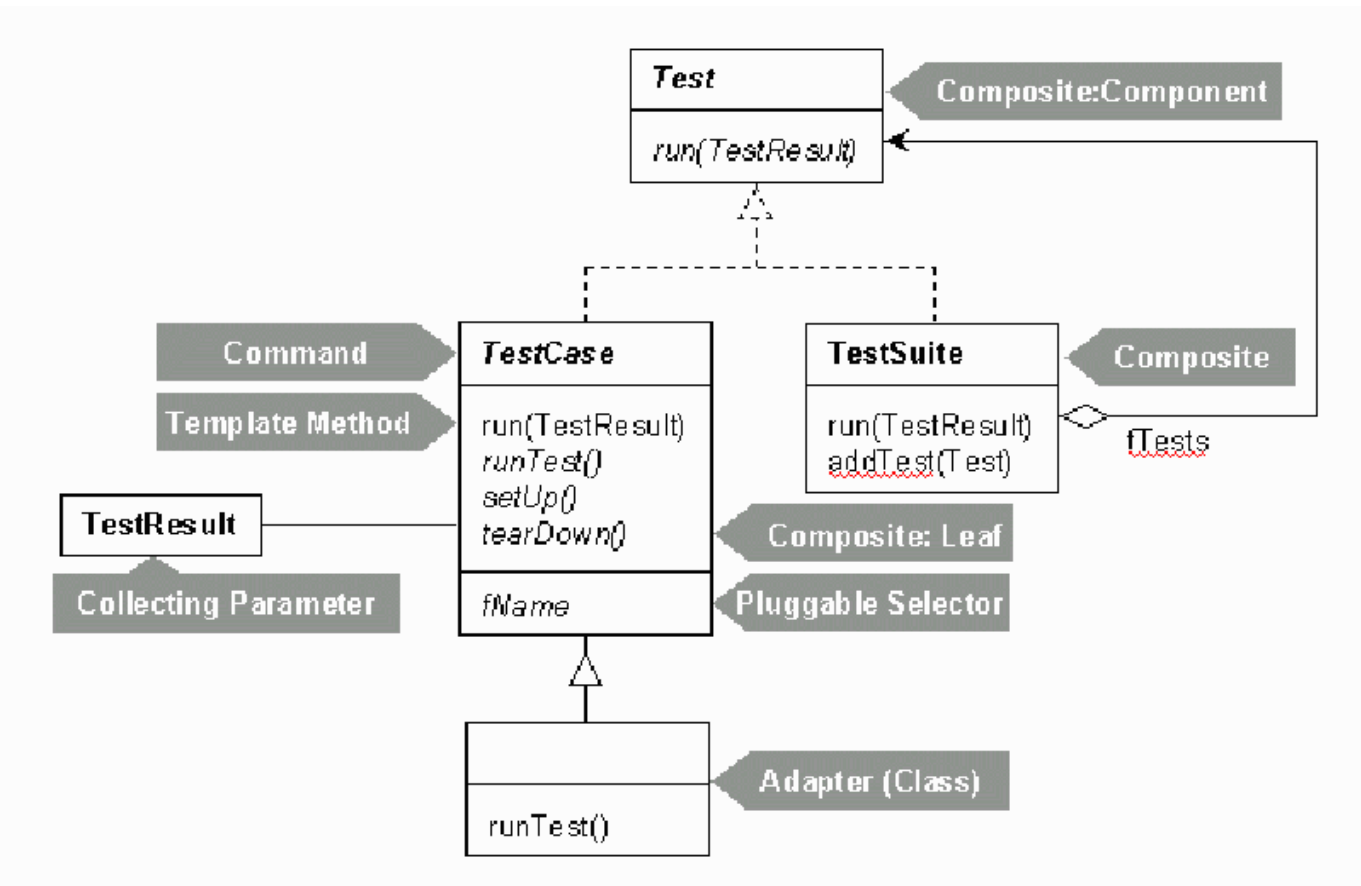


```
public static Test suite ()
{
    return new TestSuite (MoneyTest.class);
}
```

The original way is still useful when you want to run a subset of the test cases only.

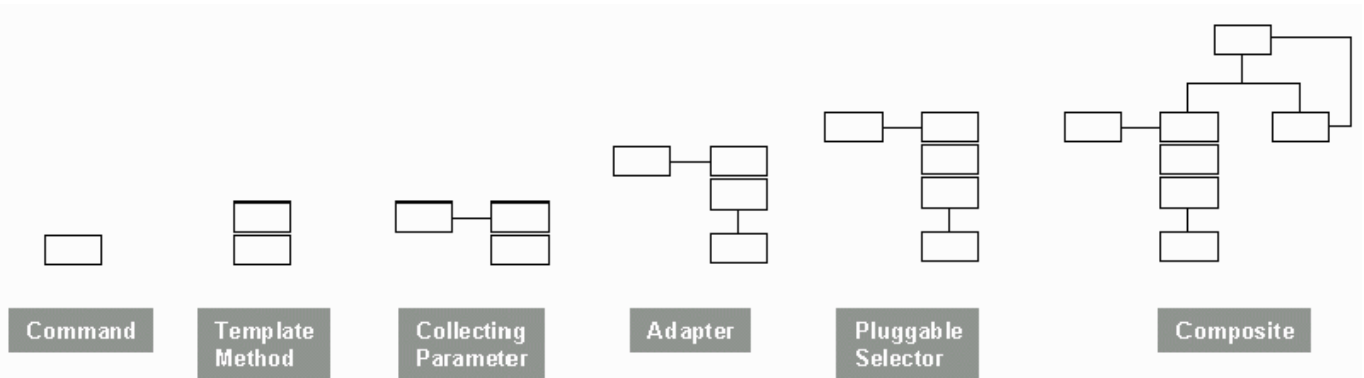
3.6

We are at the end of our cook’s tour through JUnit. The following figure shows the design of JUnit at a glance explained with patterns.



Notice how TestCase, the central abstraction in the framework, is involved in four patterns.
 Pictures of mature object designs show this same "pattern density".
 The star of the design has a rich set of relationships with the supporting players.

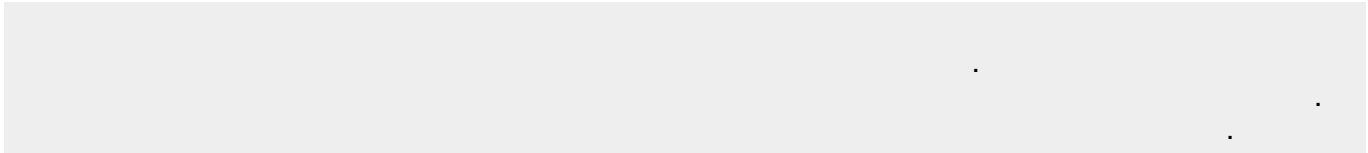
Here is another way of looking at all of the patterns in JUnit.
 In this storyboard you see an abstract representation of the effect of each of the patterns in turn.
 So, the Command pattern creates the TestCase class, the Template Method pattern creates the run method, and so on.
 (The notation of the storyboard is the notation of figure 6 with all the text deleted).



One point to notice about the storyboard is how the complexity of the picture jumps when we apply Composite.
 This is pictorial corroboration for our intuition that Composite is a powerful pattern, but that it "complicates the picture." It should therefore be used with caution.

4.
 가 .

Patterns



Pattern density

