JUnitPerf

Informatica
Corso di laurea

Università di Roma
TOR VERGATA

Overview

- JUnitPerf is a collection of JUnit test decorators used to measure the performance and scalability of functionality contained within existing JUnit tests.

- JUnitPerf contains the following JUnit test decorators:
  - TimedTest
    A TimedTest is a test decorator that runs a test and measures the elapsed time of the test.
  - LoadTest
    A LoadTest is a test decorator that runs a test with a simulated number of concurrent users and iterations.

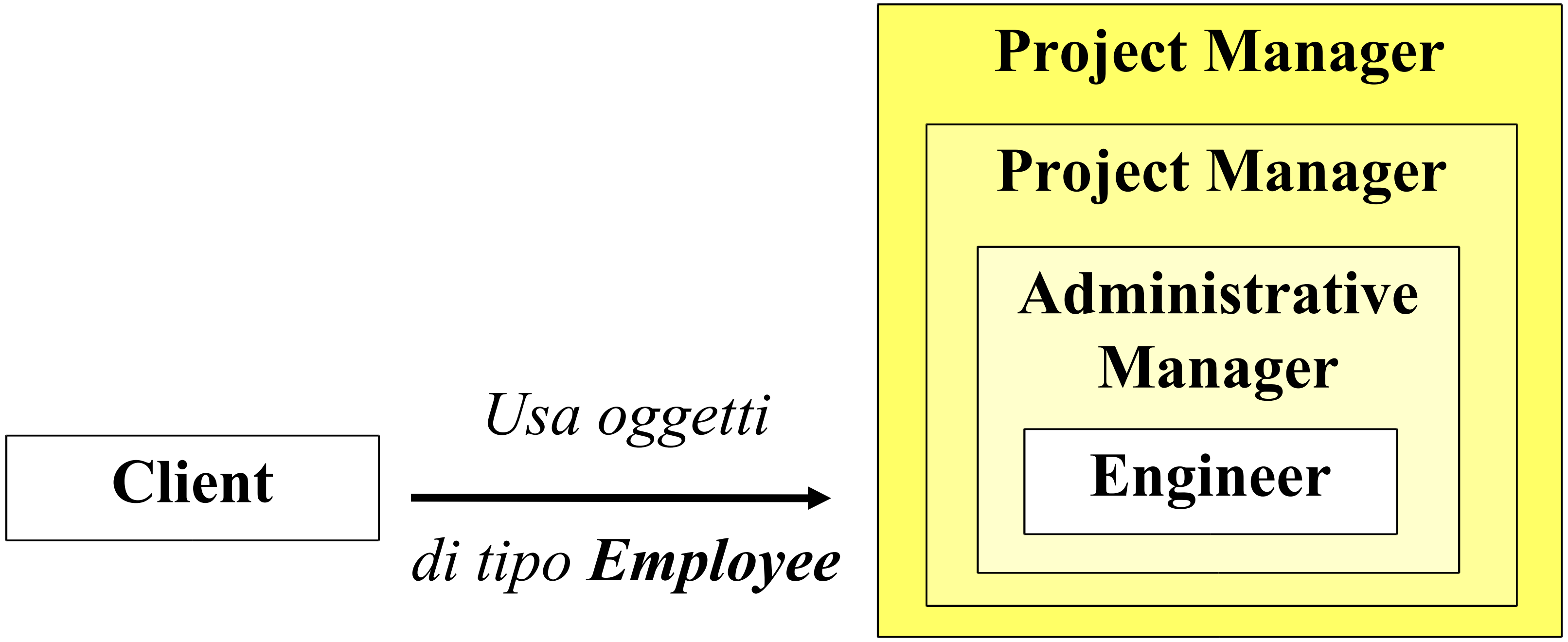
The Decorator Pattern (1/2)

Synopsis:

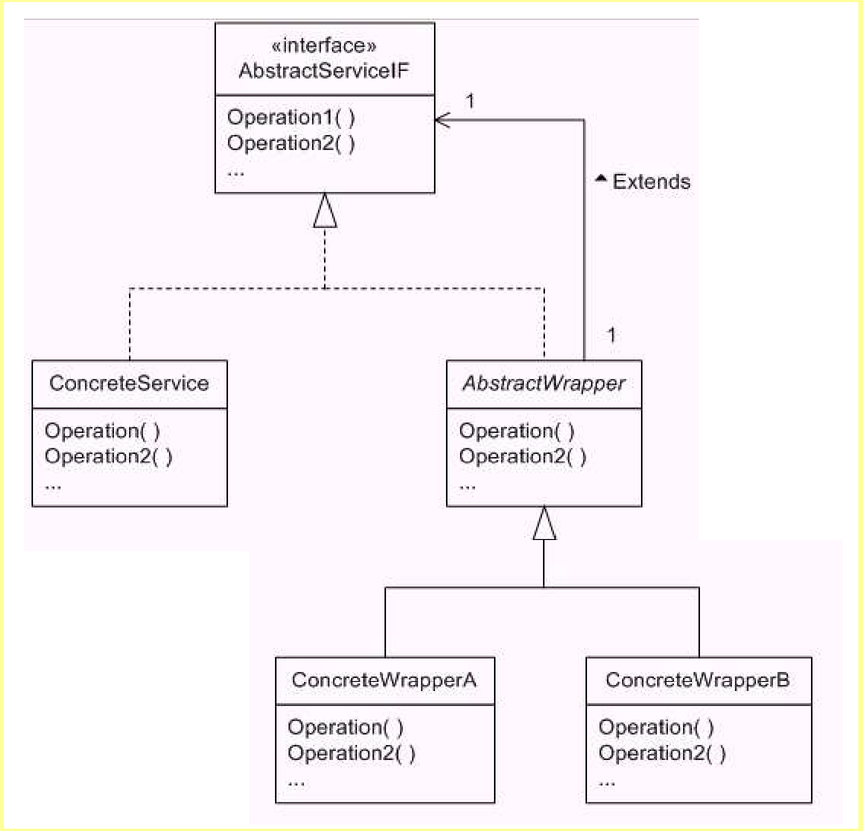
- **Il pattern Decorator estende dinamicamente le funzionalità di un oggetto in maniera trasparente ai suoi client.**
- **GoF sintetizza il pattern Decorator in questo modo: "Attach additional responsabilities to an object dinamically. Decorators provide a flexible alternative to subclassing for extending functionality"**

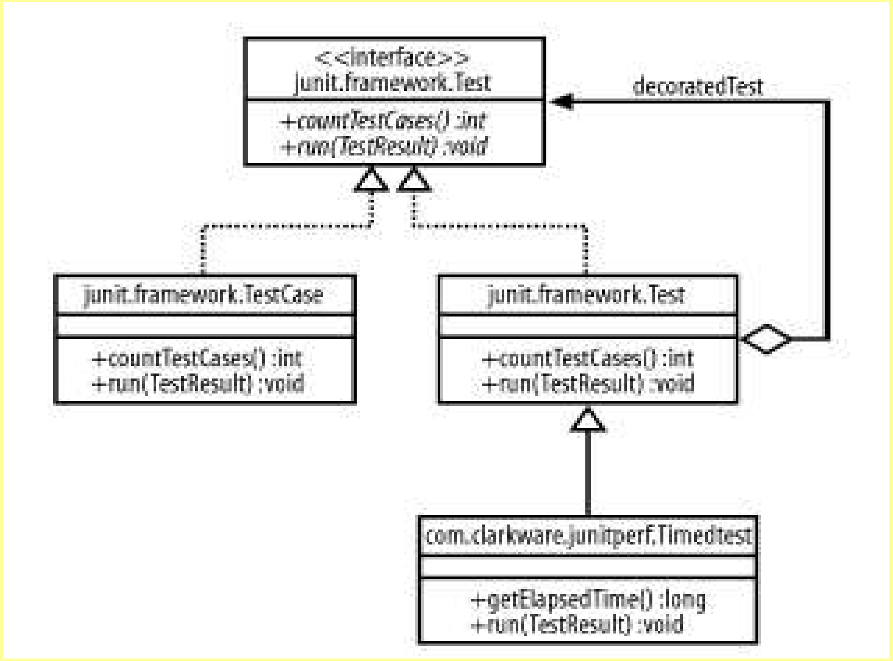
The Decorator Pattern (2/2)

- **Esempio:**



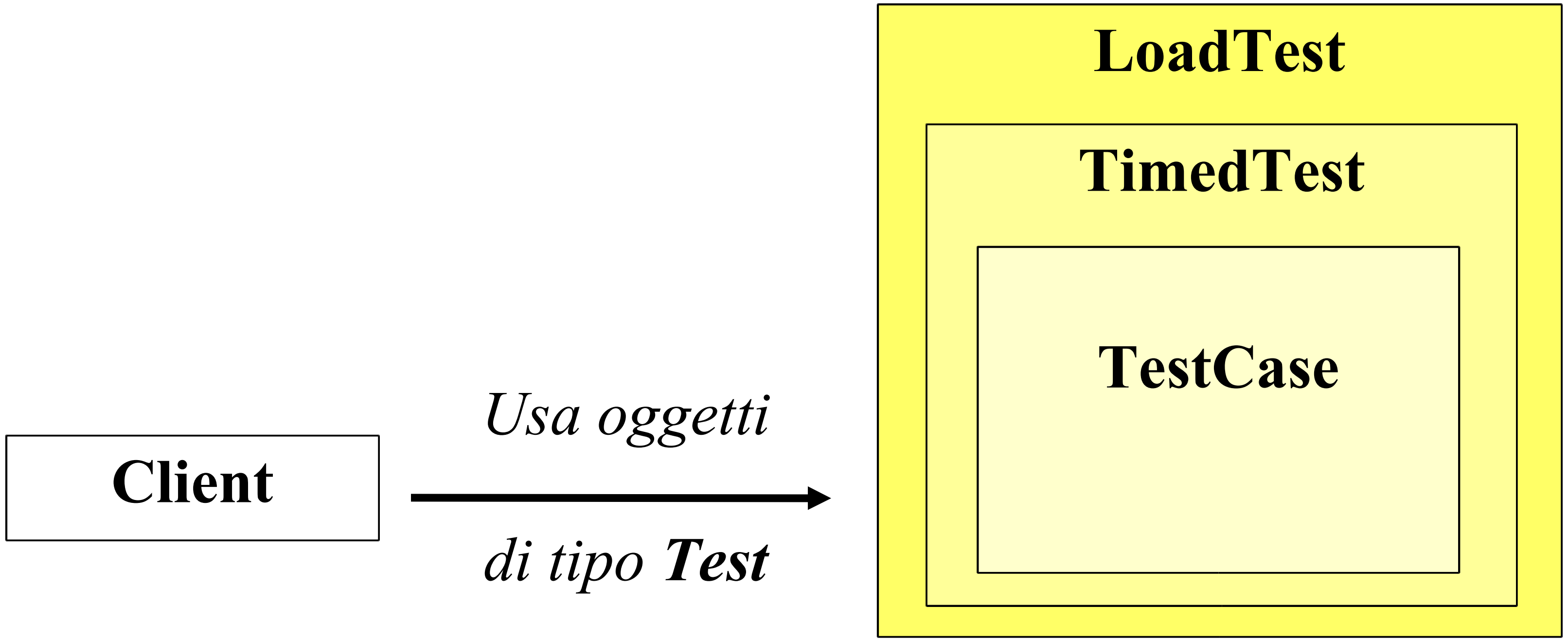
The Decorator Pattern and JunitPerf





JunitPerf usage

- **Esempio:**



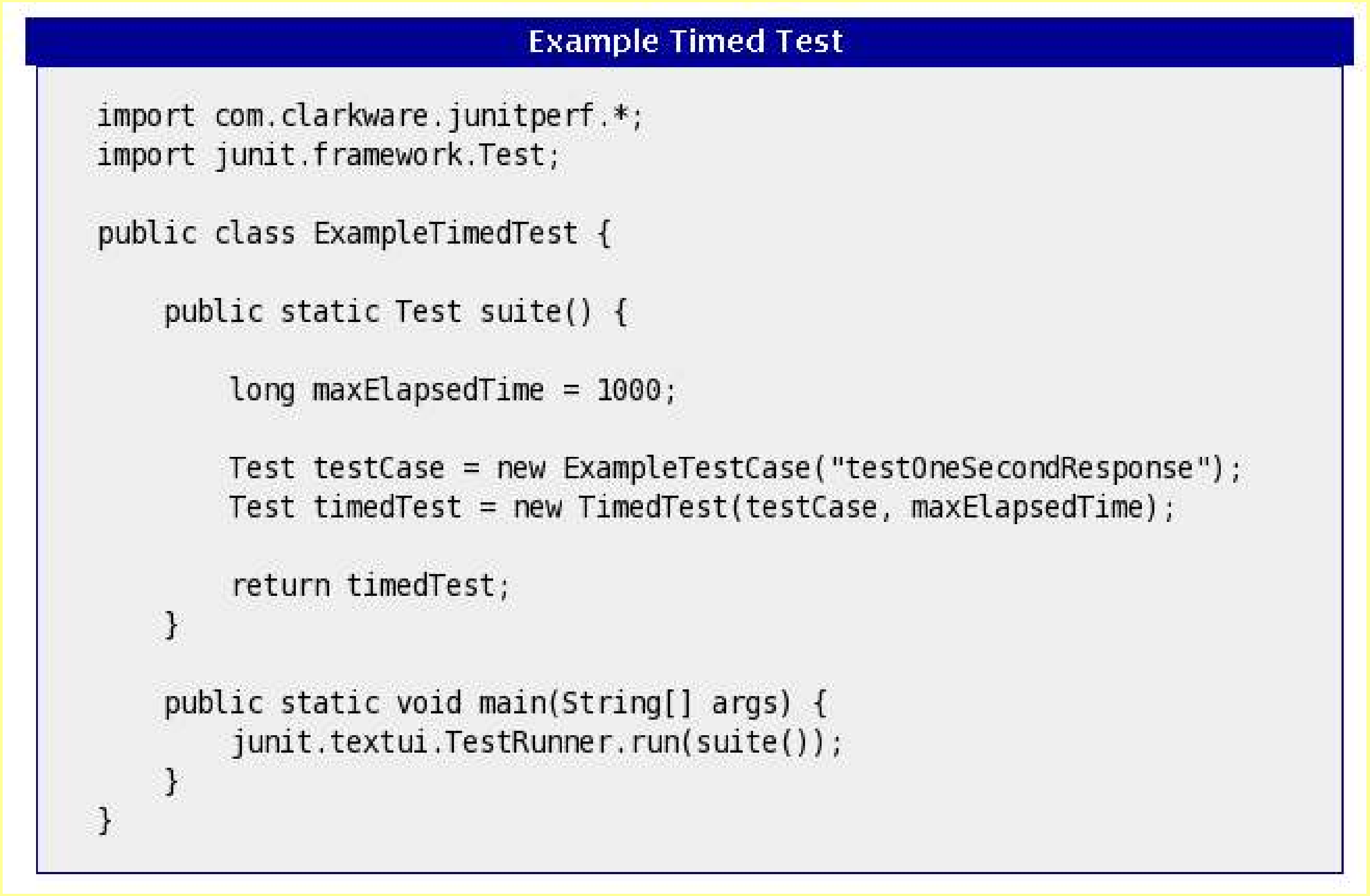
When to use JUnitPerf

- Use JunitPerf to ensure that new features and refactoring do not slow down code that used to be fast enough.
  - It is a tool for continuous performance testing
- Use a commercial profiling tool, such as Jprobe or OptimizIt, to manually inspect code and identify application bottlenecks.

Creating a TimedTest

- You need to make sure that code executes within a given amount of time.
  - use a commercial profiling tool, such as Jprobe or OptimizIt, to manually inspect code and identify application bottlenecks.

Example Timed Test



Example Timed Test

```
import com.clarkware.junitperf.*;
import junit.framework.Test;

public class ExampleTimedTest {

    public static Test suite() {

        long maxElapsedTime = 1000;

        Test testCase = new ExampleTestCase("testOneSecondResponse");
        Test timedTest = new TimedTest(testCase, maxElapsedTime);

        return timedTest;
    }

    public static void main(String[] args) {
        junit.textui.TestRunner.run(suite());
    }
}
```

About ExampleTimedTest

- To create a timed test that fails immediately when the elapsed time of the test method exceeds 1 second, use:

```
long maxElapsedTime = 1000;
Test testCase = new ExampleTestCase("testOneSecondResponse");
Test timedTest = new TimedTest(testCase, maxElapsedTime, false);
```

- The granularity of the test decoration design offered by JUnit, and used by JUnitPerf, imposes some limitations.
  - The elapsed time measured by a TimedTest decorating a single testXXX() method of a TestCase includes the total time of the setUp(), testXXX(), and tearDown() methods.

Creating a LoadTest

- You need to make sure that code executes correctly under varying load conditions, such as a large number of concurrent users.

- A load test of 10 concurrent users with each user running the test method once and all users starting simultaneously:

```
int users = 10;
Test testCase = new ExampleTestCase("testOneSecondResponse");
Test loadTest = new LoadTest(testCase, users);
```

Creating a LoadTest

- **A load test of 10 concurrent users with each user running the test method once and with a 1 second delay between the addition of users:**

```
int users = 10;
Timer timer = new ConstantTimer(1000);
// Timer timer = new RandomTimer(1000, 500);
Test testCase = new ExampleTestCase
   ("testOneSecondResponse");
Test loadTest = new LoadTest(testCase, users, timer);
```

Creating a LoadTest

- **A load test of 10 concurrent users with each user running the test method for 20 iterations, and with a 1 second delay between the addition of users:**

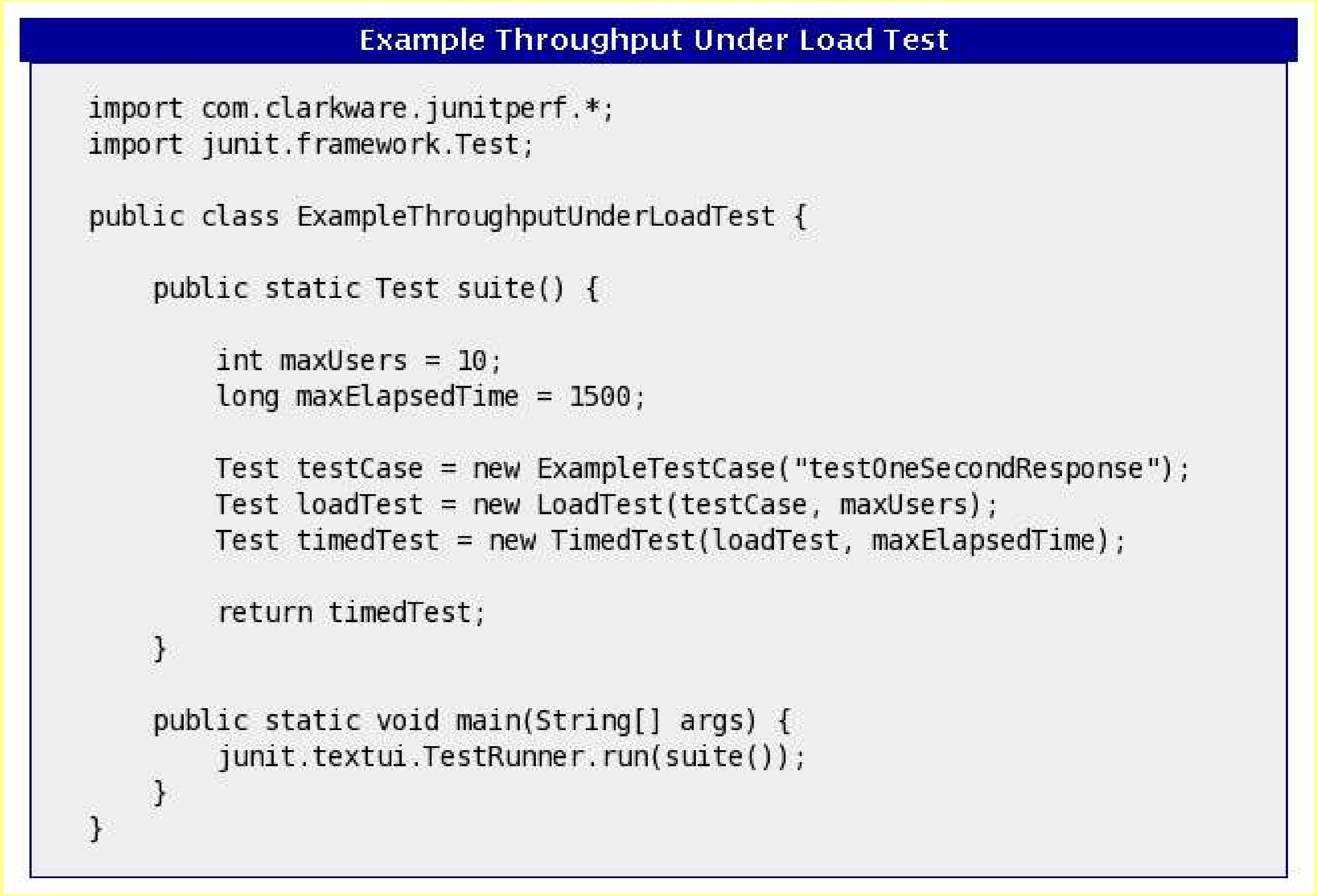
```
int users = 10;
int iterations = 20;
Timer timer = new ConstantTimer(1000);
Test testCase = new ExampleTestCase
   ("testOneSecondResponse");
Test repeatedTest = new RepeatedTest(testCase, iterations);
Test loadTest = new LoadTest(repeatedTest, users, timer);
```

Creating a Timed Test for Varying Loads

- You need to test throughput under varying load conditions.

- The application does not screech to a halt as the number of users increase.

Creating a Timed Test for Varying Loads



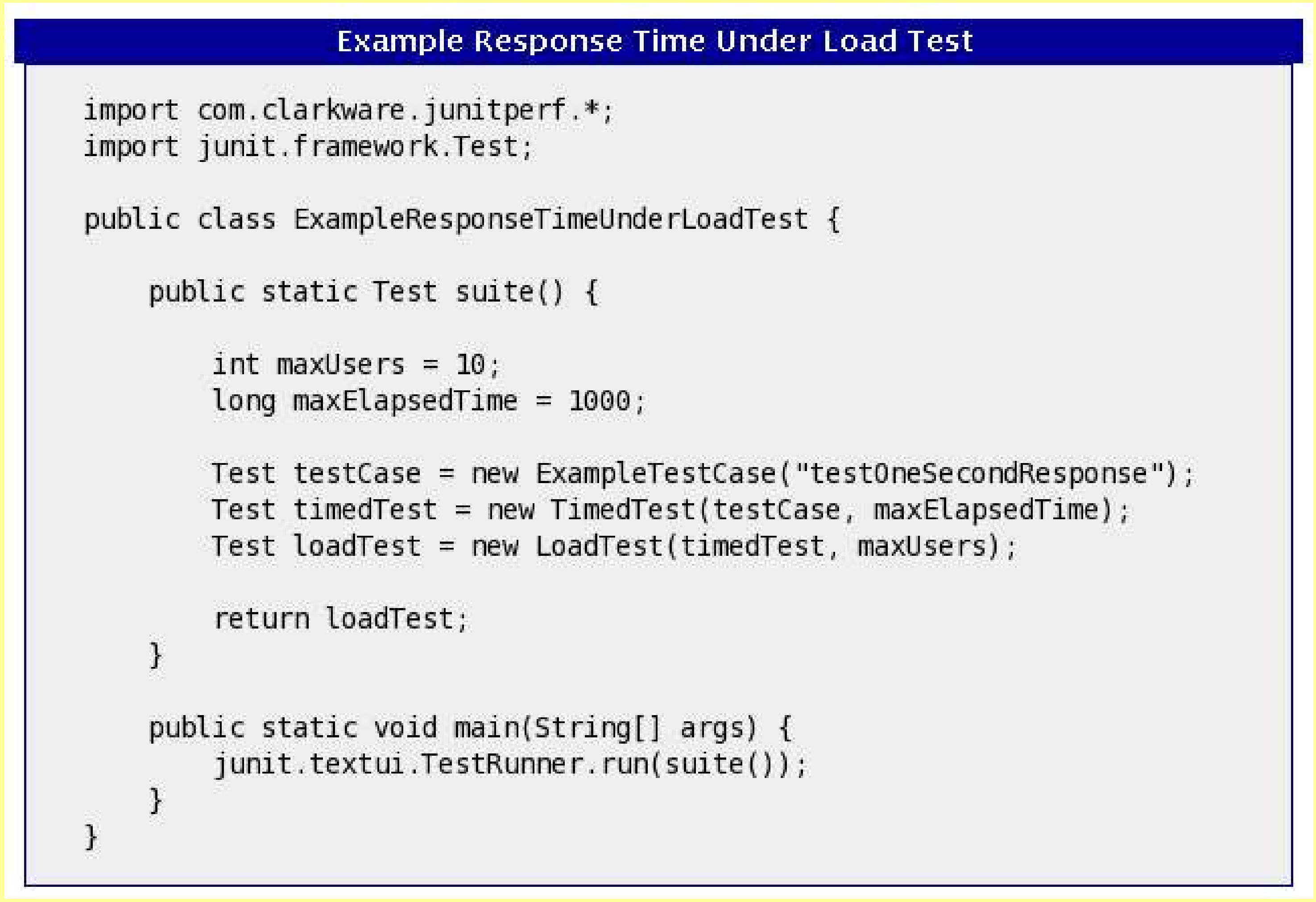
Example Throughput Under Load Test

```
import com.clarkware.junitperf.*;
import junit.framework.Test;

public class ExampleThroughputUnderLoadTest {

    public static Test suite() {

        int maxUsers = 10;
        long maxElapsedTime = 1500;

        Test testCase = new ExampleTestCase("testOneSecondResponse");
        Test loadTest = new LoadTest(testCase, maxUsers);
        Test timedTest = new TimedTest(loadTest, maxElapsedTime);

        return timedTest;
    }

    public static void main(String[] args) {
        junit.textui.TestRunner.run(suite());
    }
}
```

Testing Individual Response Times Under Load

- You need to test that a single user's response time is adequate under heavy loads.
- Useful for stressing testing and helps pinpoint the load that causes the code to break down. If there is a bottleneck, each successive user's response time increases.

Testing Individual Response Times Under Load



Example Response Time Under Load Test

```
import com.clarkware.junitperf.*;
import junit.framework.Test;

public class ExampleResponseTimeUnderLoadTest {

    public static Test suite() {

        int maxUsers = 10;
        long maxElapsedTime = 1000;

        Test testCase = new ExampleTestCase("testOneSecondResponse");
        Test timedTest = new TimedTest(testCase, maxElapsedTime);
        Test loadTest = new LoadTest(timedTest, maxUsers);

        return loadTest;
    }

    public static void main(String[] args) {
        junit.textui.TestRunner.run(suite());
    }
}
```

Running a TestSuite with Ant

- **Add another target to the Ant buildfile that executes a junit tast for all JunitPerf classes.**

```
<target name="junitperf" depends="compile">
  <junit printsummary="on" fork="false" haltonfailure="false">
    <classpath refid="classpath.project"/>
    <formatter type="plain" usefile="false"/>
    <batchtest fork="false" todir="${dir.build}">
      <fileset dir="${dir.src}">
        <include name="**/TestPerf*.java"/>
      </fileset>
    </batchtest>
  </junit>
</target>
```

Bibliografy and Links

- junitperf-1.9.1/docs/JUnitPerf.html
- Java Extreme Programming Cookbook
  Eric Burke, Brian Coyner - O'Reilly & Associates
- http://www.clarkware.com/software/junitperf-1.9.1.zip

Writing Effective JUnitPerf Tests

- **Timed Tests**
  - **Waiting Timed Tests**
  - **Non-Waiting Timed Tests**
- **Load Tests**
  - **Non-Atomic Load Tests**
  - **Atomic Load Tests**