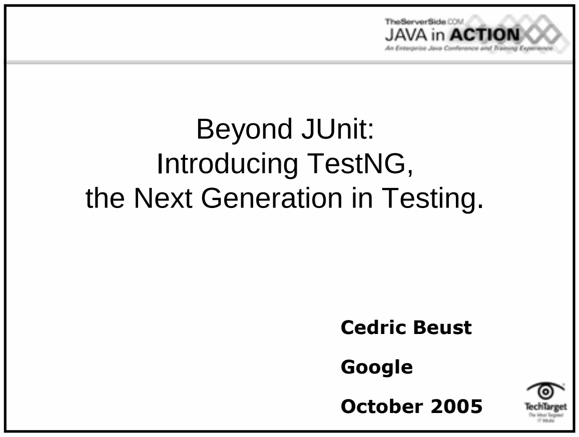
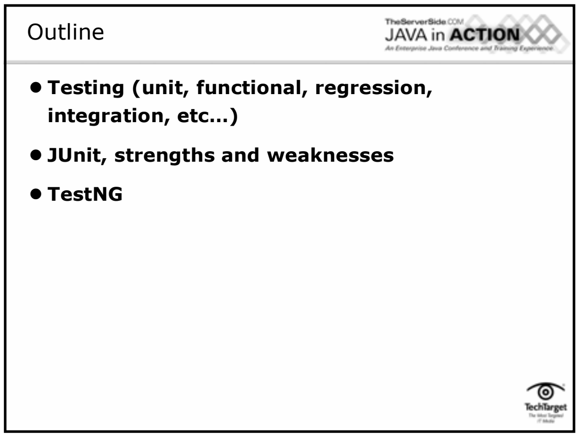


Beyond JUnit: Introducing TestNG, the Next Generation in Testing.

Cedric Beust

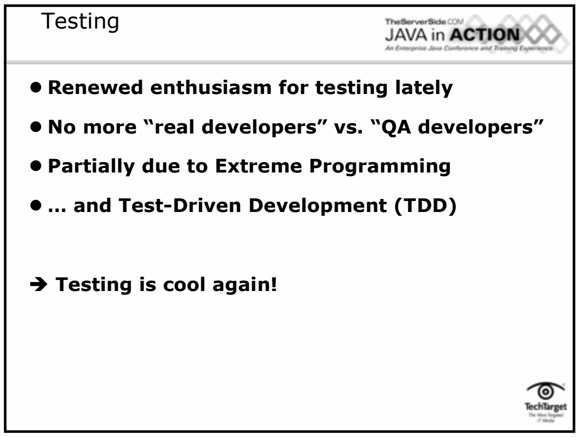
Google

October 2005



Outline

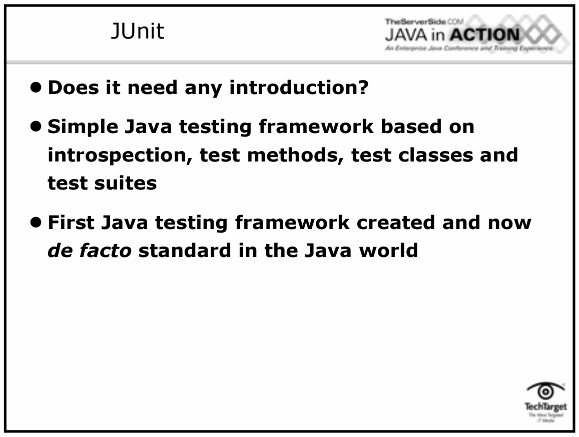
- **Testing (unit, functional, regression, integration, etc…)**
- **JUnit, strengths and weaknesses**
- **TestNG**



Testing

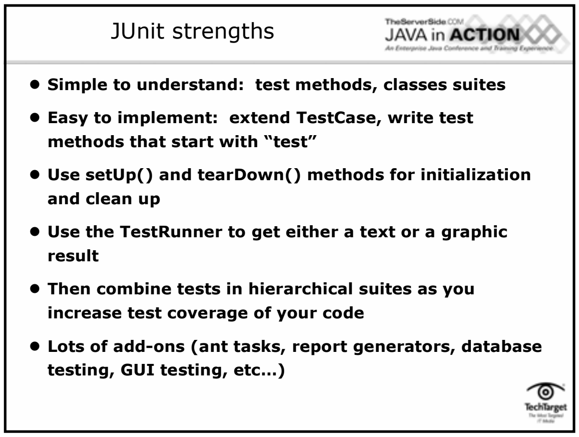
- **Renewed enthusiasm for testing lately**
- **No more "real developers" vs. "QA developers"**
- **Partially due to Extreme Programming**
- **… and Test-Driven Development (TDD)**

➔ Testing is cool again!



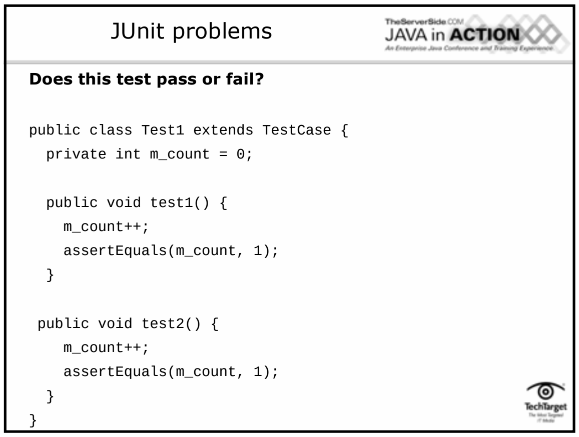
JUnit

- Does it need any introduction?
- Simple Java testing framework based on introspection, test methods, test classes and test suites
- First Java testing framework created and now *de facto* standard in the Java world



JUnit strengths

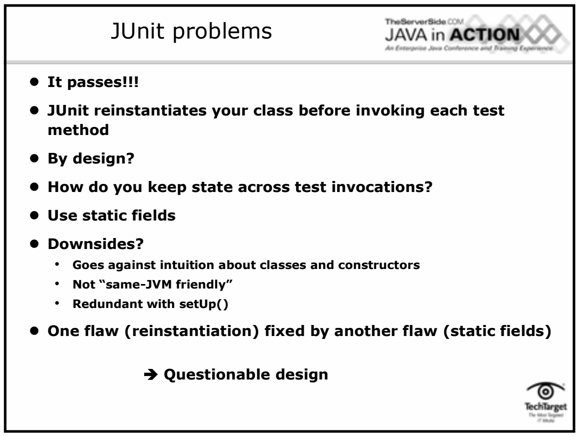
- Simple to understand: test methods, classes suites
- Easy to implement: extend TestCase, write test methods that start with "test"
- Use setUp() and tearDown() methods for initialization and clean up
- Use the TestRunner to get either a text or a graphic result
- Then combine tests in hierarchical suites as you increase test coverage of your code
- Lots of add-ons (ant tasks, report generators, database testing, GUI testing, etc…)



JUnit problems

Does this test pass or fail?

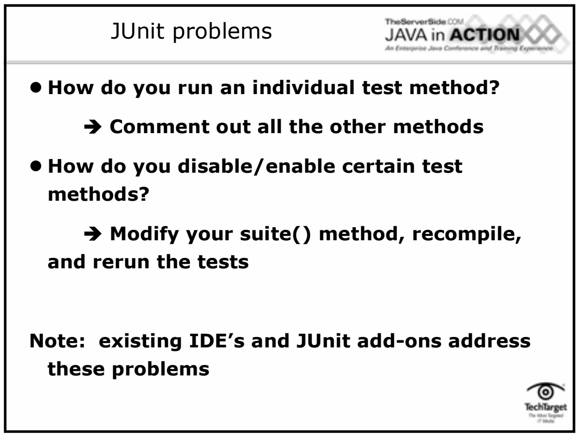
```
public class Test1 extends TestCase {
  private int m_count = 0;

  public void test1() {
    m_count++;
    assertEquals(m_count, 1);
  }

 public void test2() {
    m_count++;
    assertEquals(m_count, 1);
  }
}
```



JUnit problems

- **It passes!!!**
- **JUnit reinstantiates your class before invoking each test method**
- **By design?**
- **How do you keep state across test invocations?**
- **Use static fields**
- **Downsides?**
  - **Goes against intuition about classes and constructors**
  - **Not "same-JVM friendly"**
  - **Redundant with setUp()**
- **One flaw (reinstantiation) fixed by another flaw (static fields)**

➔ Questionable design



JUnit problems

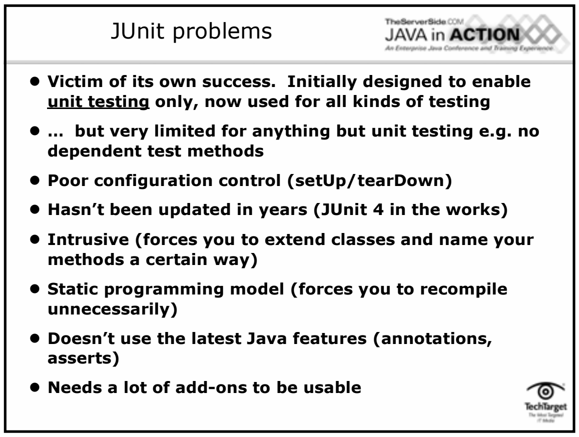
- **How do you run an individual test method?**

  ➔ Comment out all the other methods

- **How do you disable/enable certain test methods?**

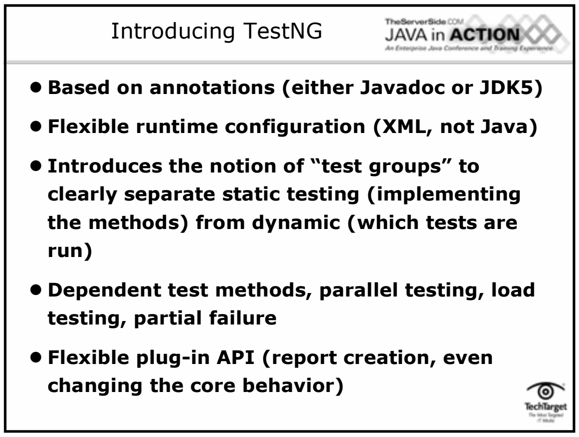
  ➔ Modify your suite() method, recompile, and rerun the tests

Note: existing IDE's and JUnit add-ons address these problems



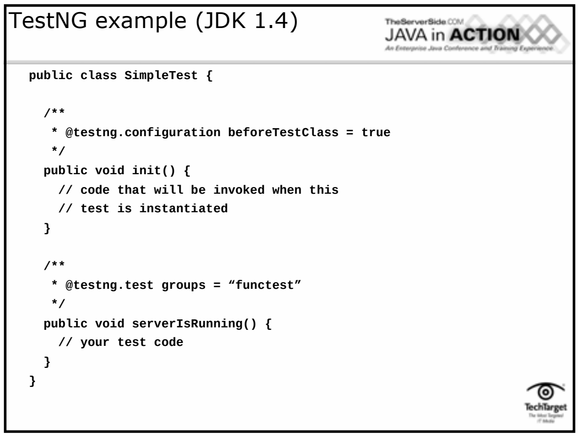
JUnit problems

- **Victim of its own success. Initially designed to enable <u>unit testing</u> only, now used for all kinds of testing**
- **… but very limited for anything but unit testing e.g. no dependent test methods**
- **Poor configuration control (setUp/tearDown)**
- **Hasn't been updated in years (JUnit 4 in the works)**
- **Intrusive (forces you to extend classes and name your methods a certain way)**
- **Static programming model (forces you to recompile unnecessarily)**
- **Doesn't use the latest Java features (annotations, asserts)**
- **Needs a lot of add-ons to be usable**



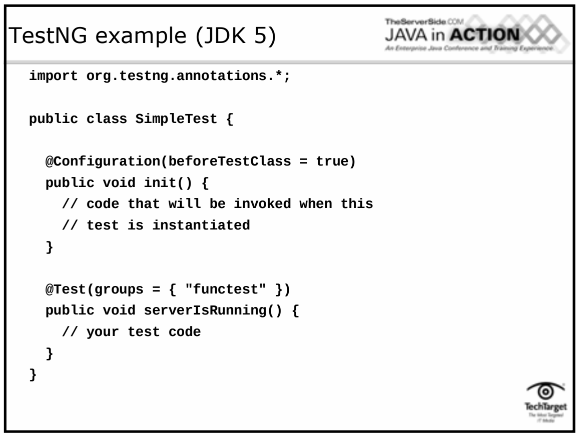
Introducing TestNG

- Based on annotations (either Javadoc or JDK5)
- Flexible runtime configuration (XML, not Java)
- Introduces the notion of “test groups” to clearly separate static testing (implementing the methods) from dynamic (which tests are run)
- Dependent test methods, parallel testing, load testing, partial failure
- Flexible plug-in API (report creation, even changing the core behavior)



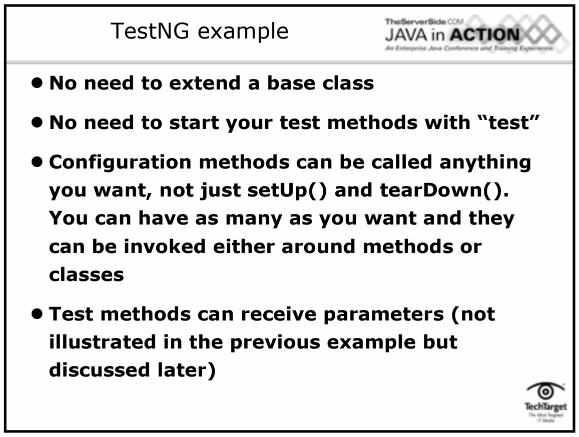
TestNG example (JDK 1.4)

```
public class SimpleTest {

  /**
   * @testng.configuration beforeTestClass = true
   */
  public void init() {
    // code that will be invoked when this
    // test is instantiated
  }

  /**
   * @testng.test groups = “functest”
   */
  public void serverIsRunning() {
    // your test code
  }
}
```



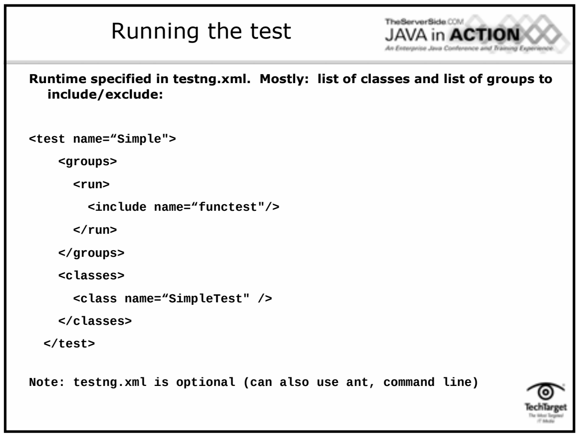
TestNG example (JDK 5)

```
import org.testng.annotations.*;

public class SimpleTest {

  @Configuration(beforeTestClass = true)
  public void init() {
    // code that will be invoked when this
    // test is instantiated
  }

  @Test(groups = { "functest" })
  public void serverIsRunning() {
    // your test code
  }
}
```



TestNG example

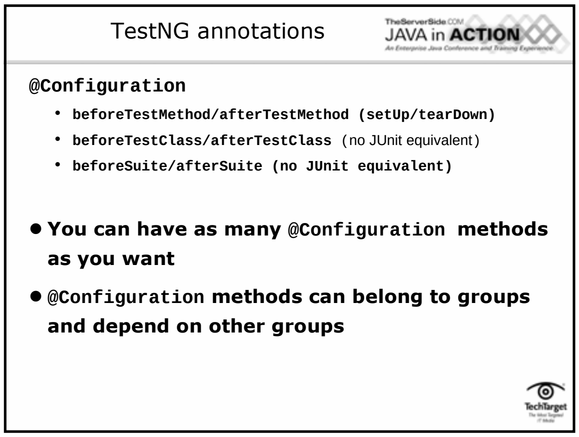
- No need to extend a base class
- No need to start your test methods with "test"
- Configuration methods can be called anything you want, not just setUp() and tearDown(). You can have as many as you want and they can be invoked either around methods or classes
- Test methods can receive parameters (not illustrated in the previous example but discussed later)



Running the test

Runtime specified in testng.xml. Mostly: list of classes and list of groups to include/exclude:

```
<test name="Simple">
    <groups>
      <run>
        <include name="functest"/>
      </run>
    </groups>
    <classes>
      <class name="SimpleTest" />
    </classes>
  </test>

Note: testng.xml is optional (can also use ant, command line)
```

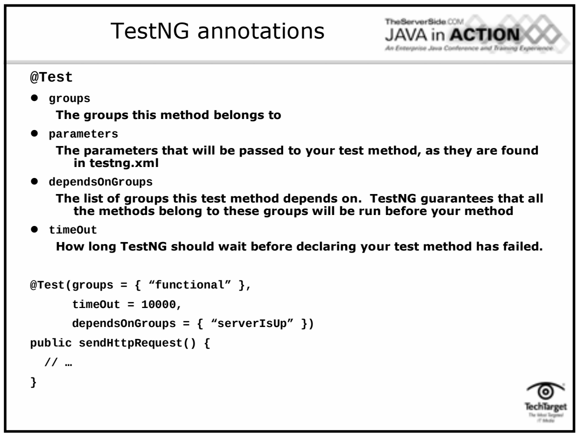


TestNG annotations

`@Configuration`

- `beforeTestMethod/afterTestMethod (setUp/tearDown)`
- `beforeTestClass/afterTestClass` (no JUnit equivalent)
- `beforeSuite/afterSuite (no JUnit equivalent)`

- You can have as many `@Configuration` methods as you want
- `@Configuration` methods can belong to groups and depend on other groups

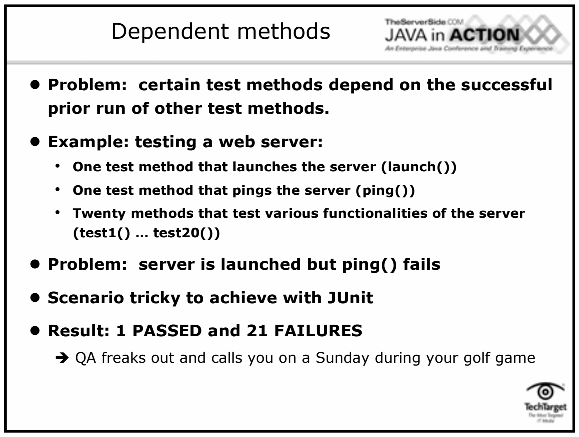


TestNG annotations

@Test

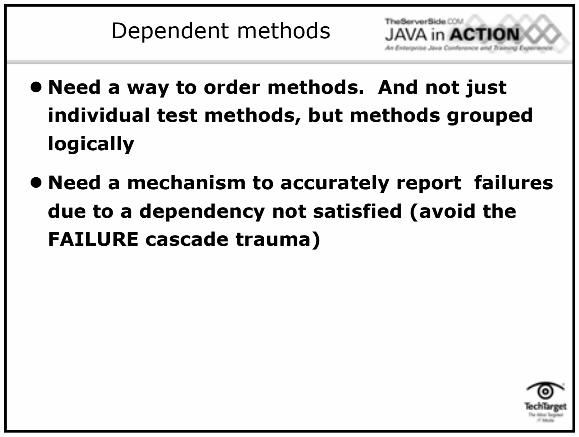
- groups
  The groups this method belongs to
- parameters
  The parameters that will be passed to your test method, as they are found in testng.xml
- dependsOnGroups
  The list of groups this test method depends on. TestNG guarantees that all the methods belong to these groups will be run before your method
- timeOut
  How long TestNG should wait before declaring your test method has failed.

```
@Test(groups = { “functional” },
      timeOut = 10000,
      dependsOnGroups = { “serverIsUp” })
public sendHttpRequest() {
  // …
}
```



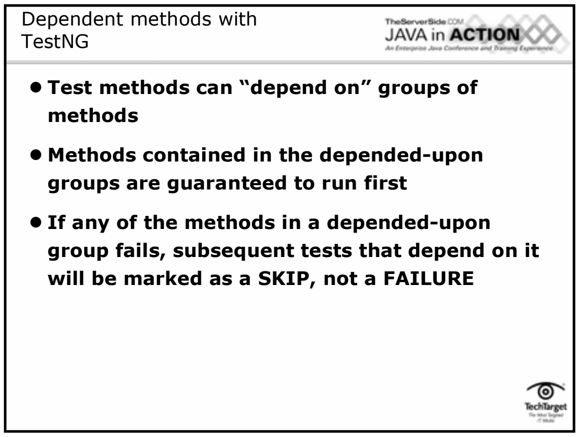
Dependent methods

- **Problem: certain test methods depend on the successful prior run of other test methods.**
- **Example: testing a web server:**
  - **One test method that launches the server (launch())**
  - **One test method that pings the server (ping())**
  - **Twenty methods that test various functionalities of the server (test1() … test20())**
- **Problem: server is launched but ping() fails**
- **Scenario tricky to achieve with JUnit**
- **Result: 1 PASSED and 21 FAILURES**
  ➔ QA freaks out and calls you on a Sunday during your golf game



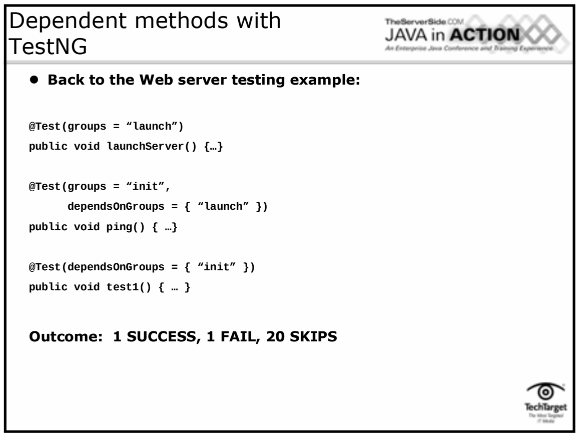
Dependent methods

- **Need a way to order methods. And not just individual test methods, but methods grouped logically**
- **Need a mechanism to accurately report failures due to a dependency not satisfied (avoid the FAILURE cascade trauma)**



Dependent methods with TestNG

- Test methods can “depend on” groups of methods
- Methods contained in the depended-upon groups are guaranteed to run first
- If any of the methods in a depended-upon group fails, subsequent tests that depend on it will be marked as a SKIP, not a FAILURE

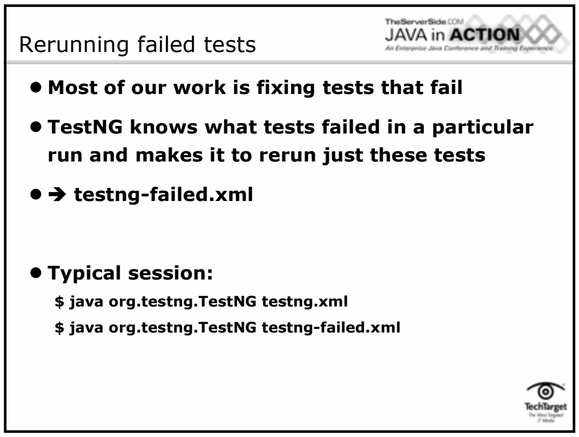


Dependent methods with TestNG

- Back to the Web server testing example:

```
@Test(groups = “launch”)
public void launchServer() {…}

@Test(groups = “init”,
      dependsOnGroups = { “launch” })
public void ping() { …}

@Test(dependsOnGroups = { “init” })
public void test1() { … }
```

Outcome: 1 SUCCESS, 1 FAIL, 20 SKIPS



Rerunning failed tests

- Most of our work is fixing tests that fail
- TestNG knows what tests failed in a particular run and makes it to rerun just these tests
- ➔ testng-failed.xml

- Typical session:

```
$ java org.testng.TestNG testng.xml
$ java org.testng.TestNG testng-failed.xml
```

testng.xml

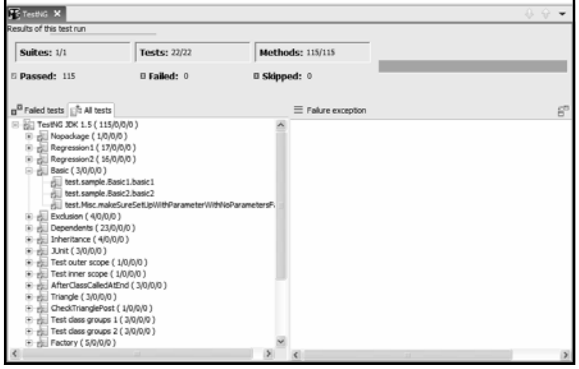
Where all the runtime information goes:

- **The test methods, classes, packages**
- **Which groups should be run (include-groups)**
- **Which groups should not be run (exclude-groups)**
- **Define additional groups ("groups of groups")**
- **Whether the tests should be run in parallel**
- **Parameters**
- **JUnit mode**

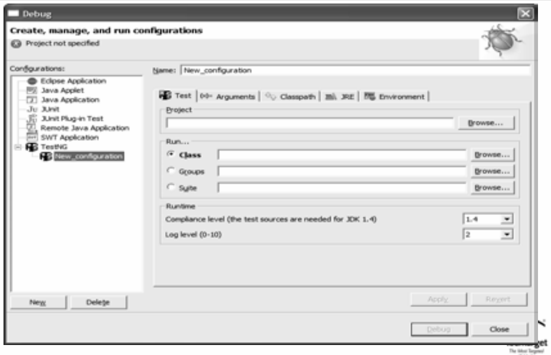
Eclipse and IDEA

TestNG plug-ins exist for both Eclipse and IDEA:

- **Run a test method, test class, group testng.xml**
- **Easy selection of groups and suite files**
- **Show the familiar red/green bar**
- **Directly jump to test failures**
- **Automatically convert from JUnit to TestNG**



```
Suites: 1/1          Tests: 22/22          Methods: 115/115
Passed: 115          Failed: 0             Skipped: 0

TestNG JDK 1.5 ( 115/0/0/0 )
  Nopackage ( 1/0/0/0 )
  Regression1 ( 17/0/0/0 )
  Regression2 ( 16/0/0/0 )
  Basic ( 3/0/0/0 )
    test.sample.Basic1.basic1
    test.sample.Basic2.basic2
    test.Misc.makeSureSetUpWithParameterWithNoParametersF
  Exclusion ( 4/0/0/0 )
  Dependents ( 23/0/0/0 )
  Inheritance ( 4/0/0/0 )
  JUnit ( 3/0/0/0 )
  Test outer scope ( 1/0/0/0 )
  Test inner scope ( 1/0/0/0 )
  AfterClassCalledAtEnd ( 3/0/0/0 )
  Triangle ( 3/0/0/0 )
  CheckTrianglePost ( 1/0/0/0 )
  Test class groups 1 ( 3/0/0/0 )
  Test class groups 2 ( 3/0/0/0 )
  Factory ( 5/0/0/0 )
```



Create, manage, and run configurations

Project not specified

Configurations:

- Eclipse Application
- Java Applet
- Java Application
- JUnit
- JUnit Plug-in Test
- Remote Java Application
- SWT Application
- TestNG
  - New_configuration

Name: New_configuration

Project

Run...

● Class

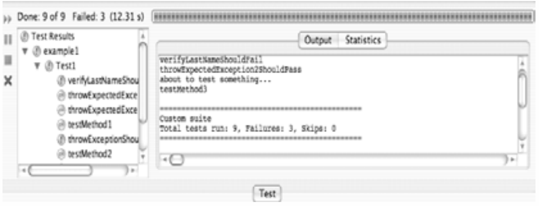
○ Groups

○ Suite

Runtime

Compliance level (the test sources are needed for JDK 1.4): 1.4

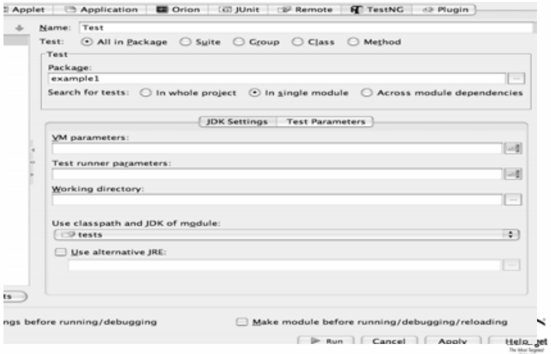
Log level (0-10): 2



Done: 9 of 9 Failed: 3 (12.31 s)

- Test Results
  - example1
    - Test1
      - verifyLastNameShou
      - throwExpectedExce
      - throwExpectedExce
      - testMethod1
      - throwExceptionShou
      - testMethod2

```
verifyLastNameShouldFail
throwExpectedException2ShouldPass
about to test something...
testMethod3

===============================================
Custom suite
Total tests run: 9, Failures: 3, Skips: 0
===============================================
```



Applet | Application | Orion | JUnit | Remote | TestNG | Plugin

Name: Test

Test: ● All in Package ○ Suite ○ Group ○ Class ○ Method

Test

Package: example1

Search for tests: ○ In whole project ● In single module ○ Across module dependencies

JDK Settings | Test Parameters

VM parameters:

Test runner parameters:

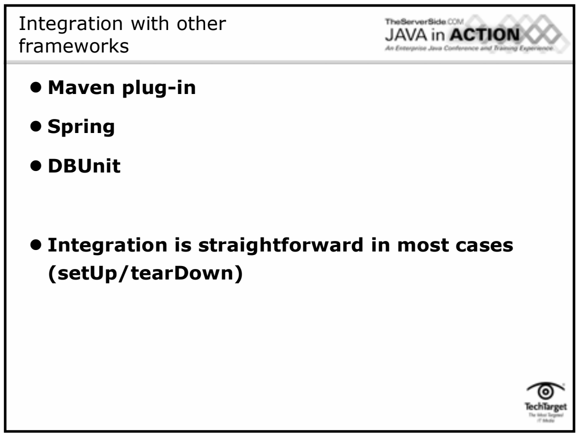
Working directory:

Use classpath and JDK of module: tests

□ Use alternative JRE:

ngs before running/debugging

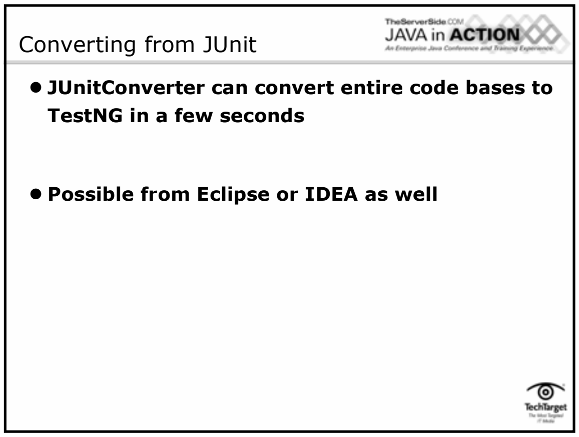
□ Make module before running/debugging/reloading



Integration with other frameworks

- Maven plug-in
- Spring
- DBUnit

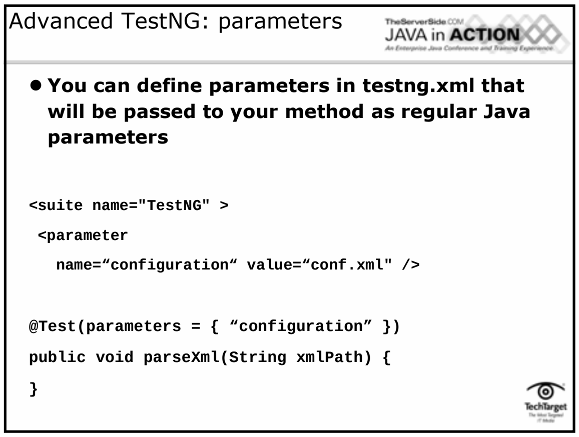
- Integration is straightforward in most cases (setUp/tearDown)



Converting from JUnit

- JUnitConverter can convert entire code bases to TestNG in a few seconds

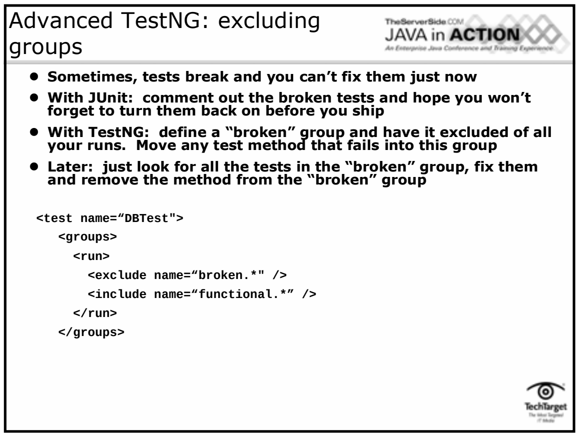
- Possible from Eclipse or IDEA as well



Advanced TestNG: parameters

- You can define parameters in testng.xml that will be passed to your method as regular Java parameters

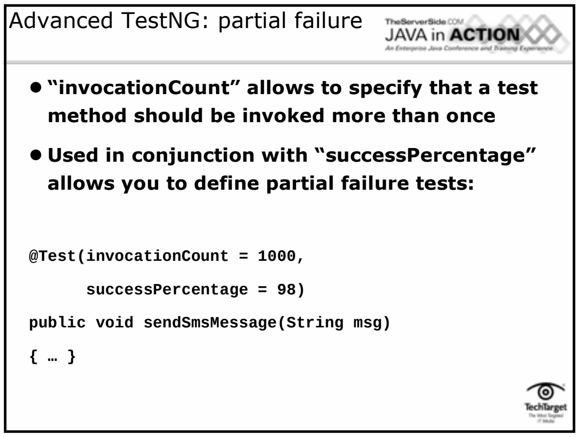
```
<suite name="TestNG" >
 <parameter
   name=“configuration“ value=“conf.xml" />

@Test(parameters = { “configuration” })
public void parseXml(String xmlPath) {
}
```



Advanced TestNG: excluding groups

- Sometimes, tests break and you can't fix them just now
- With JUnit: comment out the broken tests and hope you won't forget to turn them back on before you ship
- With TestNG: define a "broken" group and have it excluded of all your runs. Move any test method that fails into this group
- Later: just look for all the tests in the "broken" group, fix them and remove the method from the "broken" group

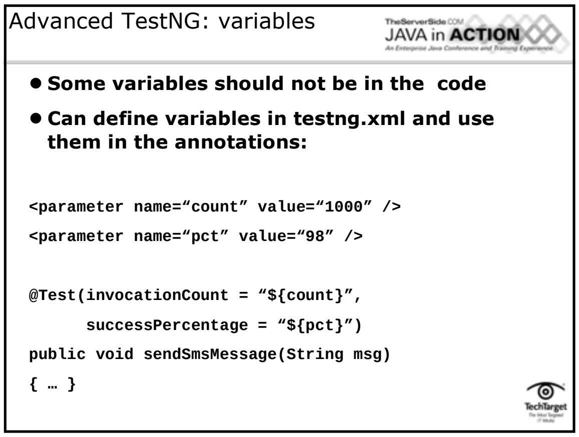
```
<test name="DBTest">
   <groups>
     <run>
       <exclude name="broken.*" />
       <include name="functional.*" />
     </run>
   </groups>
```



Advanced TestNG: partial failure

- "invocationCount" allows to specify that a test method should be invoked more than once
- Used in conjunction with "successPercentage" allows you to define partial failure tests:

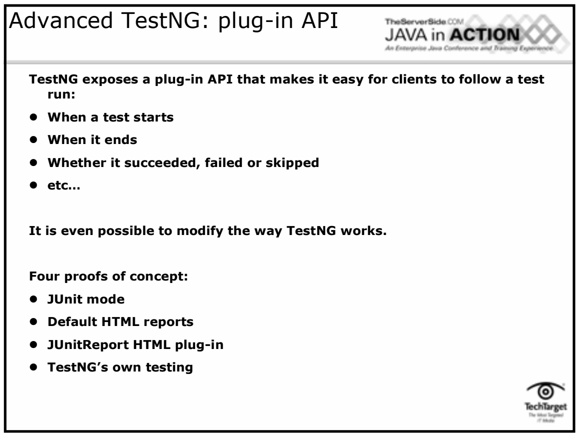
```
@Test(invocationCount = 1000,
      successPercentage = 98)
public void sendSmsMessage(String msg)
{ … }
```



Advanced TestNG: variables

- Some variables should not be in the code
- Can define variables in testng.xml and use them in the annotations:

```
<parameter name="count" value="1000" />
<parameter name="pct" value="98" />

@Test(invocationCount = "${count}",
      successPercentage = "${pct}")
public void sendSmsMessage(String msg)
{ … }
```



Advanced TestNG: plug-in API

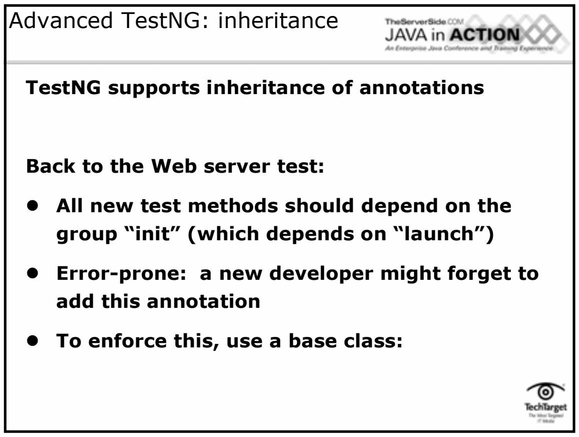
TestNG exposes a plug-in API that makes it easy for clients to follow a test run:

- When a test starts
- When it ends
- Whether it succeeded, failed or skipped
- etc…

It is even possible to modify the way TestNG works.

Four proofs of concept:

- JUnit mode
- Default HTML reports
- JUnitReport HTML plug-in
- TestNG’s own testing

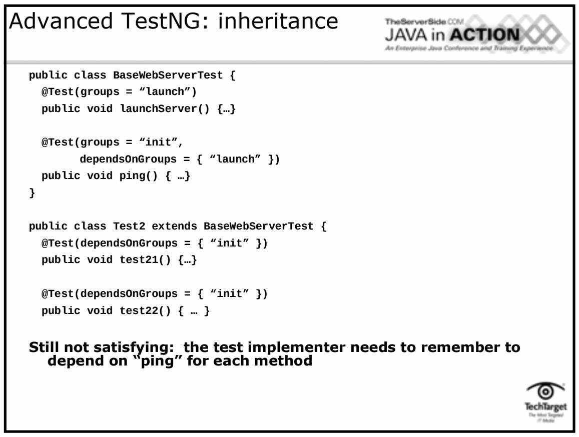


Advanced TestNG: inheritance

TestNG supports inheritance of annotations

Back to the Web server test:

- **All new test methods should depend on the group “init” (which depends on “launch”)**
- **Error-prone: a new developer might forget to add this annotation**
- **To enforce this, use a base class:**



Advanced TestNG: inheritance

```
public class BaseWebServerTest {
  @Test(groups = “launch”)
  public void launchServer() {…}

  @Test(groups = “init”,
        dependsOnGroups = { “launch” })
  public void ping() { …}
}

public class Test2 extends BaseWebServerTest {
  @Test(dependsOnGroups = { “init” })
  public void test21() {…}

  @Test(dependsOnGroups = { “init” })
  public void test22() { … }
```

Still not satisfying: the test implementer needs to remember to depend on “ping” for each method

Advanced TestNG: class scope

Annotations can be scoped: an annotation defined at the class level applies to all public methods of the class.

Solution to the problem: move the annotation at the class level:

```
@Test(dependsOnGroups = { “ping” })
public class Test2 extends BaseWebServerTest {
  public void test21() { …}
  public void test22() { … }
```

Advanced TestNG: annotation inheritance

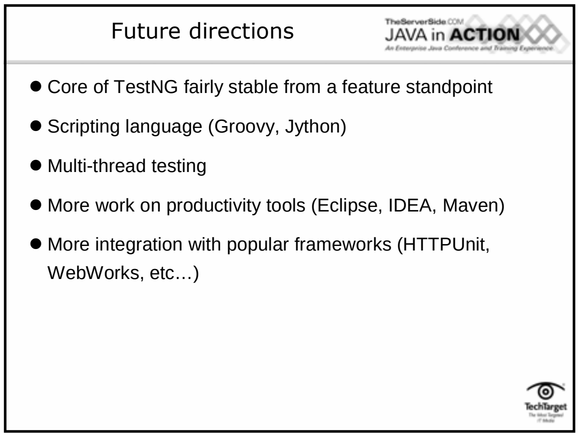
- **Annotations defined on a parent class will apply to subclasses as well.**
- **Example: “making all the test methods of a class belong to the group *win32* automatically”**

```
@Test(groups = { “win32” })
public BaseWin32Test {
}

public Win32Test extends BaseWin32Test {
  public void test1() { … }
}
```

Note how the test class becomes a simple POJO! All the TestNG complexities are hidden.

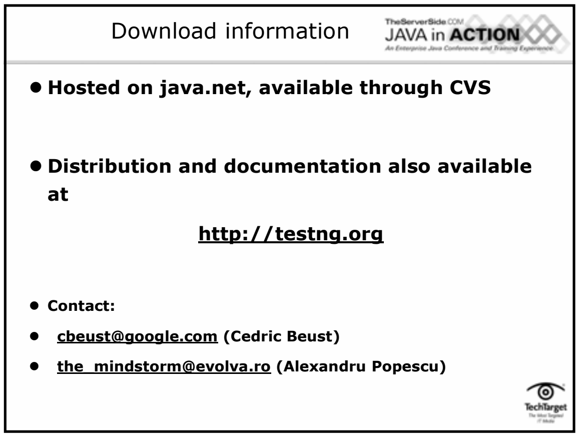
Reporting

- **TestNG issues an HTML report by default**
- **Plug-in API makes it easy to write your own reporter (example: JUnitReport plug-in)**



Future directions

- Core of TestNG fairly stable from a feature standpoint
- Scripting language (Groovy, Jython)
- Multi-thread testing
- More work on productivity tools (Eclipse, IDEA, Maven)
- More integration with popular frameworks (HTTPUnit, WebWorks, etc…)

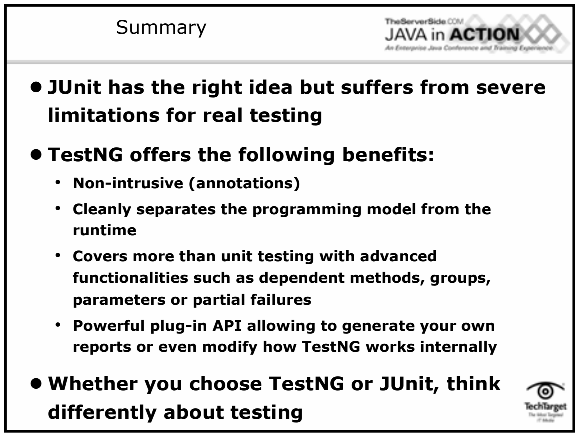


Download information

- **Hosted on java.net, available through CVS**
- **Distribution and documentation also available at**

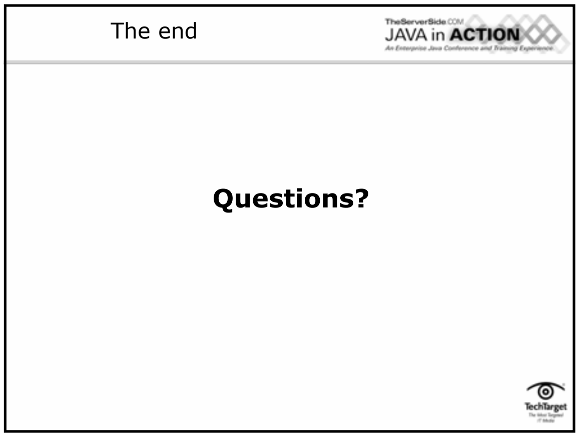
  http://testng.org

- **Contact:**
- **cbeust@google.com (Cedric Beust)**
- **the_mindstorm@evolva.ro (Alexandru Popescu)**



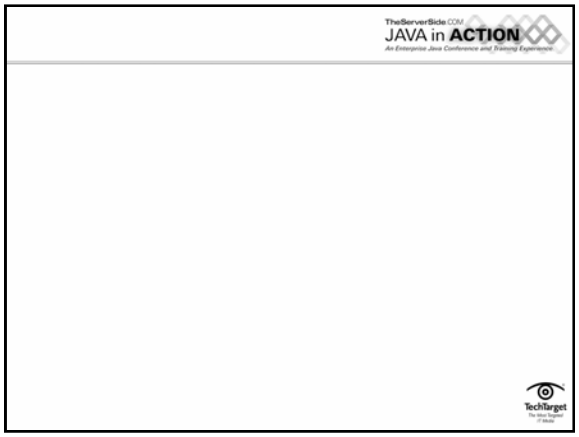
Summary

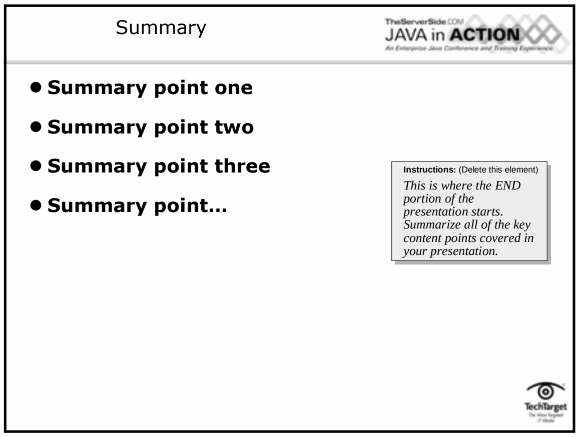
- **JUnit has the right idea but suffers from severe limitations for real testing**
- **TestNG offers the following benefits:**
  - **Non-intrusive (annotations)**
  - **Cleanly separates the programming model from the runtime**
  - **Covers more than unit testing with advanced functionalities such as dependent methods, groups, parameters or partial failures**
  - **Powerful plug-in API allowing to generate your own reports or even modify how TestNG works internally**
- **Whether you choose TestNG or JUnit, think differently about testing**



The end

Questions?

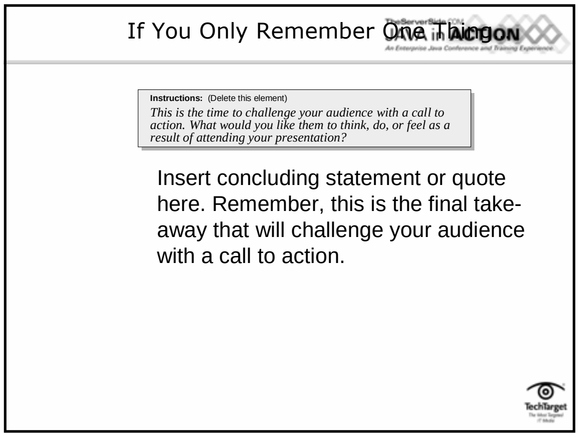




Summary

- **Summary point one**
- **Summary point two**
- **Summary point three**
- **Summary point…**

Instructions: (Delete this element)
This is where the END portion of the presentation starts. Summarize all of the key content points covered in your presentation.

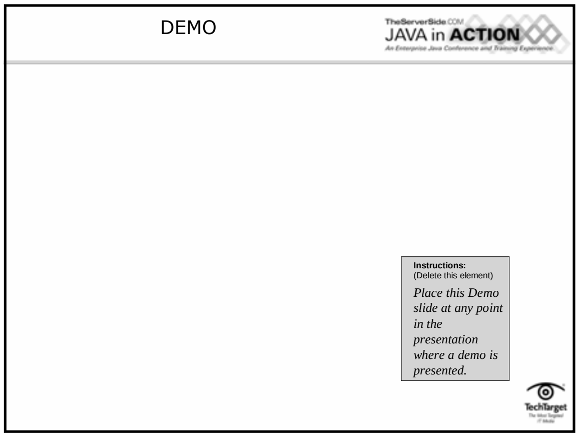


If You Only Remember One Thing…

Instructions: (Delete this element)

This is the time to challenge your audience with a call to action. What would you like them to think, do, or feel as a result of attending your presentation?

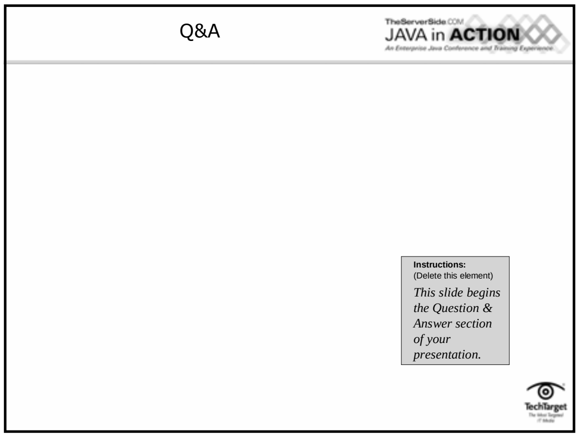
Insert concluding statement or quote here. Remember, this is the final take-away that will challenge your audience with a call to action.



DEMO

Instructions:
(Delete this element)

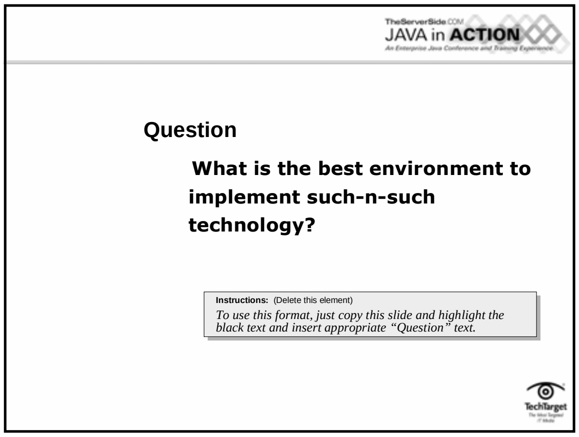
Place this Demo slide at any point in the presentation where a demo is presented.



Q&A

Instructions:
(Delete this element)

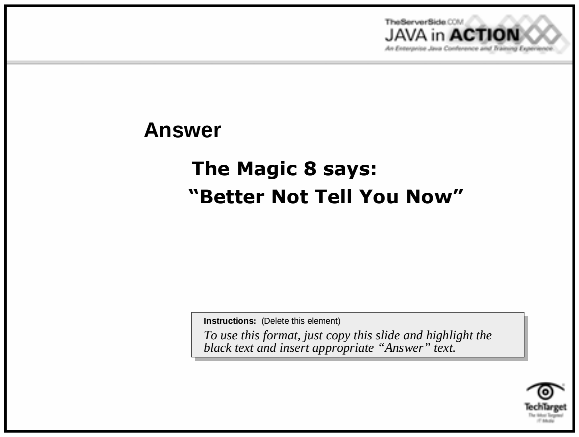
This slide begins the Question & Answer section of your presentation.



Question

What is the best environment to implement such-n-such technology?

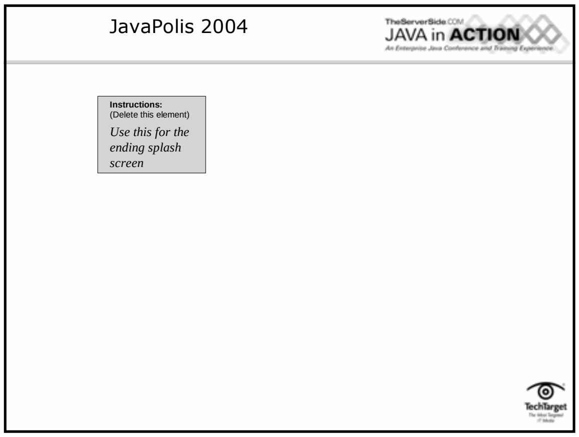
Instructions: (Delete this element)
To use this format, just copy this slide and highlight the black text and insert appropriate "Question" text.



Answer

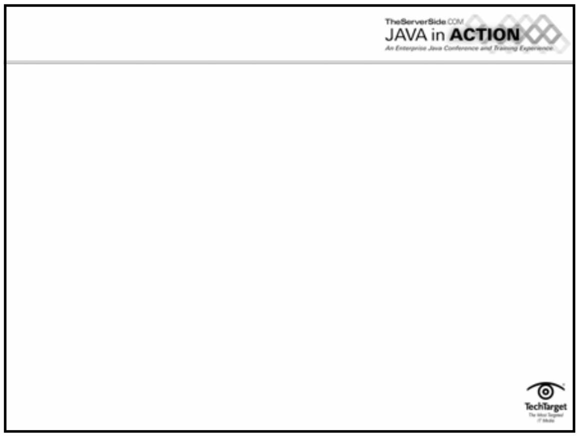
The Magic 8 says:
"Better Not Tell You Now"

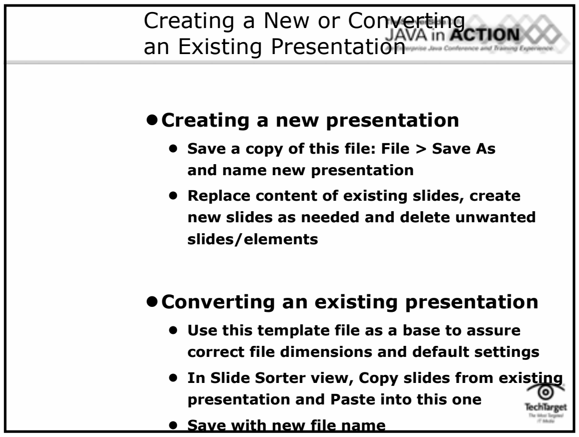
Instructions: (Delete this element)
To use this format, just copy this slide and highlight the black text and insert appropriate "Answer" text.



JavaPolis 2004

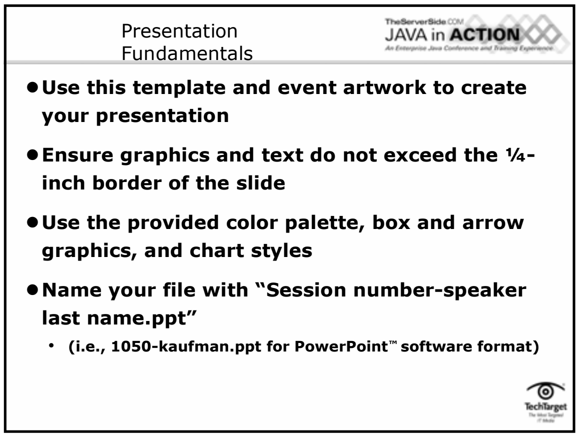
Instructions:
(Delete this element)
Use this for the ending splash screen





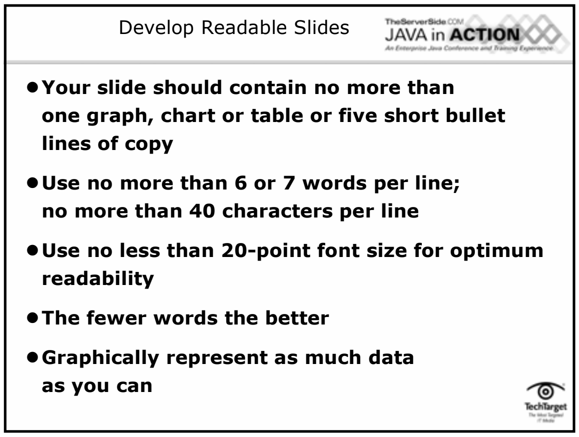
Creating a New or Converting an Existing Presentation

- **Creating a new presentation**
  - **Save a copy of this file: File > Save As and name new presentation**
  - **Replace content of existing slides, create new slides as needed and delete unwanted slides/elements**
- **Converting an existing presentation**
  - **Use this template file as a base to assure correct file dimensions and default settings**
  - **In Slide Sorter view, Copy slides from existing presentation and Paste into this one**
  - **Save with new file name**



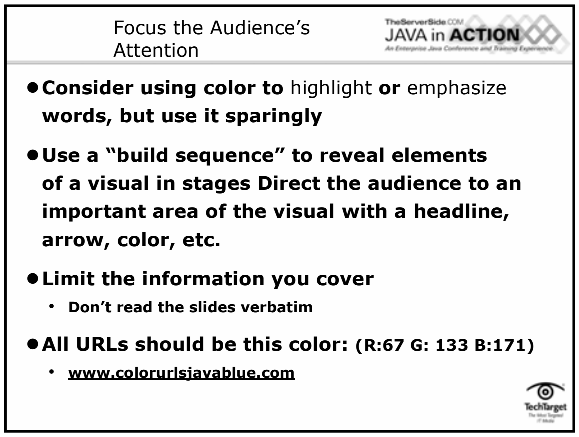
Presentation Fundamentals

- **Use this template and event artwork to create your presentation**
- **Ensure graphics and text do not exceed the ¼-inch border of the slide**
- **Use the provided color palette, box and arrow graphics, and chart styles**
- **Name your file with "Session number-speaker last name.ppt"**
  - **(i.e., 1050-kaufman.ppt for PowerPoint™ software format)**



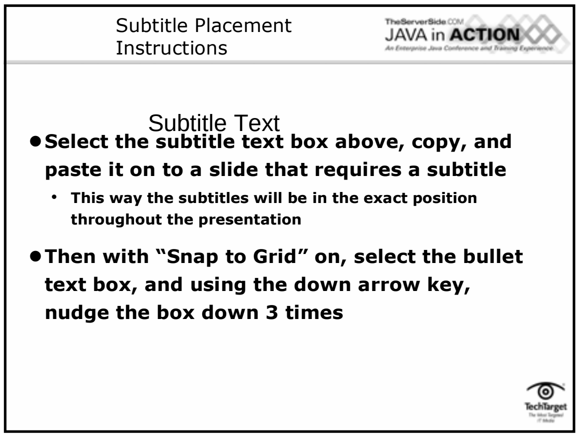
Develop Readable Slides

- Your slide should contain no more than one graph, chart or table or five short bullet lines of copy
- Use no more than 6 or 7 words per line; no more than 40 characters per line
- Use no less than 20-point font size for optimum readability
- The fewer words the better
- Graphically represent as much data as you can



Focus the Audience's Attention

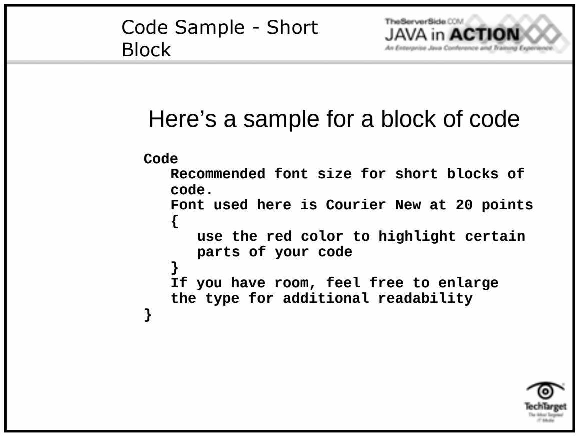
- **Consider using color to** highlight **or** emphasize **words, but use it sparingly**
- **Use a "build sequence" to reveal elements of a visual in stages Direct the audience to an important area of the visual with a headline, arrow, color, etc.**
- **Limit the information you cover**
  - **Don't read the slides verbatim**
- **All URLs should be this color: (R:67 G: 133 B:171)**
  - **www.colorurlsjavablue.com**



Subtitle Placement Instructions

Subtitle Text

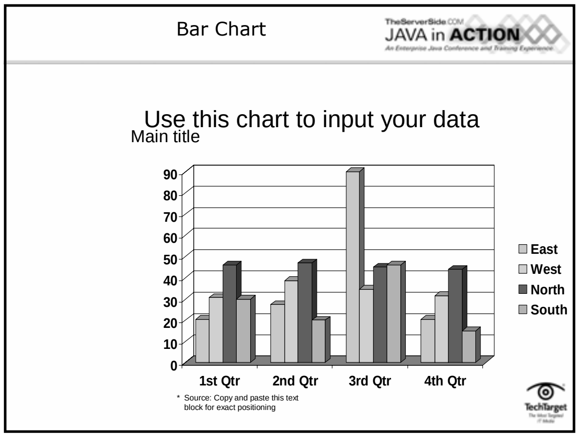
- **Select the subtitle text box above, copy, and paste it on to a slide that requires a subtitle**
  - **This way the subtitles will be in the exact position throughout the presentation**
- **Then with "Snap to Grid" on, select the bullet text box, and using the down arrow key, nudge the box down 3 times**



Code Sample - Short Block

Here's a sample for a block of code

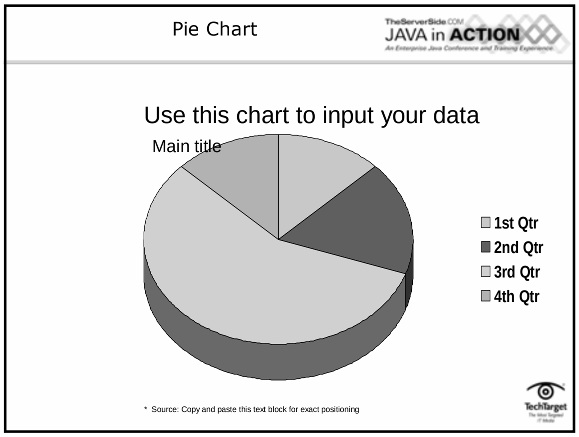
```
Code
   Recommended font size for short blocks of
   code.
   Font used here is Courier New at 20 points
   {
      use the red color to highlight certain
      parts of your code
   }
   If you have room, feel free to enlarge
   the type for additional readability
}
```



Bar Chart

Use this chart to input your data

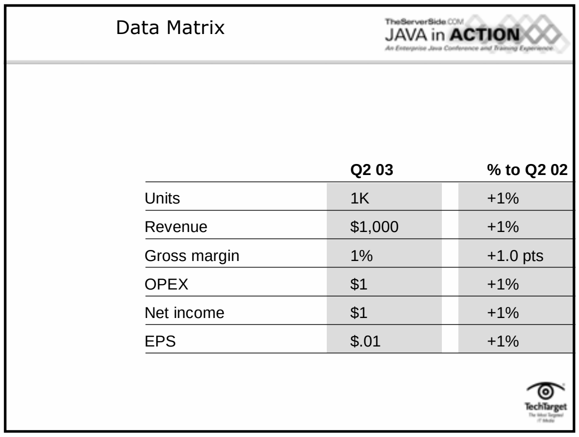
* Source: Copy and paste this text block for exact positioning



Pie Chart

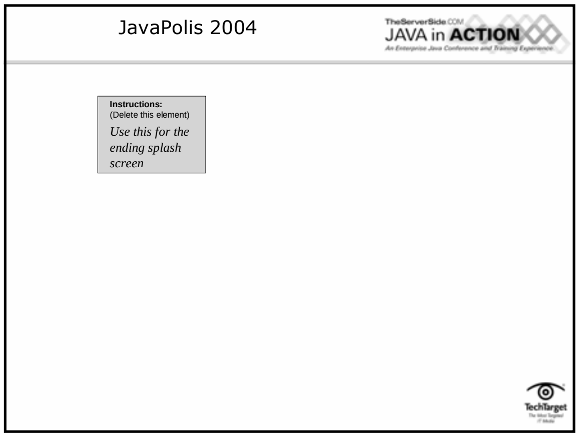
Use this chart to input your data

* Source: Copy and paste this text block for exact positioning



Data Matrix

	Q2 03	% to Q2 02
Units	1K	+1%
Revenue	$1,000	+1%
Gross margin	1%	+1.0 pts
OPEX	$1	+1%
Net income	$1	+1%
EPS	$.01	+1%



JavaPolis 2004

Instructions:
(Delete this element)

Use this for the ending splash screen