Appendix A

Test Cases

In this chapter we reproduce the source files used as test cases in the usability experiment. The files come from many established, well-known open-source projects, giving us a broad variety of coding styles and changes:

- Test case 1 (Sections A.1 and A.2) comes from the Google Collections Library [24];
- Test case 2 (Sections A.3 and A.4) is from Sun's GlassFish Application Server [22]. For brevity's sake, the license header was removed.
- Test case 3 (Sections A.5 and A.6) comes from the Eclipse project [11];
- Test case 4 (Sections A.7 and A.8) is from Jython [31], a Java compiler and interpreter for the Python programming language;
- Test case 5 (Sections A.9 and A.10) comes from the Spring Framework [51];
- Test case 6 (Sections A.11 and A.12) is from JUnit [30], the testing framework.

The files were selected roughly at random to avoid bias. First, we looked for files from about 100 to 200 lines, then we went back into the file revision history until there were about seven to 30 individual changes, with varying degrees of complexity. While browsing the file history for changes, we used the reference tool only.

For the complete source listing, please refer to the electronic version, on-line at:

`http://www.site.uottawa.ca/~damyot/students/lanna/`

A.1 1.old.java

```
1 /*
2  * Copyright (C) 2007 Google Inc.
3  *
4  * Licensed under the Apache License, Version 2.0 (the "License");
5  * you may not use this file except in compliance with the License.
6  * You may obtain a copy of the License at
7  *
8  * http://www.apache.org/licenses/LICENSE-2.0
9  *
10 * Unless required by applicable law or agreed to in writing, software
11 * distributed under the License is distributed on an "AS IS" BASIS,
12 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or ↩
   implied.
13 * See the License for the specific language governing permissions and
14 * limitations under the License.
15 */
16
17 package com.google.common.collect;
18
19 import com.google.common.base.Nullable;
20
21 import java.util.HashMap;
22 import java.util.Map;
23
24 /**
25  * A {@link BiMap} backed by two {@link HashMap} instances. This ↩
   implementation
26  * allows null keys and values.
27  *
28  * @author Mike Bostock
29  */
30 public final class HashBiMap<K, V> extends StandardBiMap<K, V> {
31   /**
32    * Constructs a new empty bimap with the default initial capacity (16) ↩
   and the
33    * default load factor (0.75).
34    */
35   public HashBiMap() {
36     super(new HashMap<K, V>(), new HashMap<V, K>());
37   }
38
39   /**
40    * Constructs a new empty bimap with the specified expected size and the
41    * default load factor (0.75).
42    *
43    * @param expectedSize the expected number of entries
```

```
44     * @throws IllegalArgumentException if the specified expected size is
45     *     negative
46     */
47    public HashBiMap(int expectedSize) {
48      super(new HashMap<K, V>(Maps.capacity(expectedSize)),
49          new HashMap<V, K>(Maps.capacity(expectedSize)));
50    }
51
52    /**
53     * Constructs a new empty bimap with the specified initial capacity ↩
   and load
54     * factor.
55     *
56     * @param initialCapacity the initial capacity
57     * @param loadFactor the load factor
58     * @throws IllegalArgumentException if the initial capacity is ↩
   negative or the
59     *     load factor is nonpositive
60     */
61    public HashBiMap(int initialCapacity, float loadFactor) {
62      super(new HashMap<K, V>(initialCapacity, loadFactor),
63          new HashMap<V, K>(initialCapacity, loadFactor));
64    }
65
66    /**
67     * Constructs a new bimap containing initial values from {@code map}. The
68     * bimap is created with the default load factor (0.75) and an initial
69     * capacity sufficient to hold the mappings in the specified map.
70     */
71    public HashBiMap(Map<? extends K, ? extends V> map) {
72      this(map.size());
73      putAll(map); // careful if we make this class non-final
74    }
75
76    // Override these two methods to show that keys and values may be null
77
78    @Override public V put(@Nullable K key, @Nullable V value) {
79      return super.put(key, value);
80    }
81
82    @Override public V forcePut(@Nullable K key, @Nullable V value) {
83      return super.forcePut(key, value);
84    }
85  }
```

A.2 1.new.java

```
1 /*
2  * Copyright (C) 2007 Google Inc.
3  *
4  * Licensed under the Apache License, Version 2.0 (the "License");
5  * you may not use this file except in compliance with the License.
6  * You may obtain a copy of the License at
7  *
8  * http://www.apache.org/licenses/LICENSE-2.0
9  *
10 * Unless required by applicable law or agreed to in writing, software
11 * distributed under the License is distributed on an "AS IS" BASIS,
12 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or ↩
   implied.
13 * See the License for the specific language governing permissions and
14 * limitations under the License.
15 */
16
17 package com.google.common.collect;
18
19 import com.google.common.base.Nullable;
20
21 import java.io.IOException;
22 import java.io.ObjectInputStream;
23 import java.io.ObjectOutputStream;
24 import java.util.HashMap;
25 import java.util.Map;
26
27 /**
28  * A {@link BiMap} backed by two {@link HashMap} instances. This ↩
   implementation
29  * allows null keys and values. A {@code HashBiMap} and its inverse are ↩
   both
30  * serializable.
31  *
32  * @author Mike Bostock
33  */
34 public final class HashBiMap<K, V> extends StandardBiMap<K, V> {
35   /**
36    * Constructs a new empty bimap with the default initial capacity (16).
37    */
38   public HashBiMap() {
39     super(new HashMap<K, V>(), new HashMap<V, K>());
40   }
41
42   /**
43    * Constructs a new empty bimap with the specified expected size.
```

```
44     *
45     * @param expectedSize the expected number of entries
46     * @throws IllegalArgumentException if the specified expected size is
47     *     negative
48     */
49    public HashBiMap(int expectedSize) {
50      super(new HashMap<K, V>(Maps.capacity(expectedSize)),
51          new HashMap<V, K>(Maps.capacity(expectedSize)));
52    }
53
54    /**
55     * Constructs a new bimap containing initial values from {@code map}. The
56     * bimap is created with an initial capacity sufficient to hold the ↩
   mappings
57     * in the specified map.
58     */
59    public HashBiMap(Map<? extends K, ? extends V> map) {
60      this(map.size());
61      putAll(map); // careful if we make this class non-final
62    }
63
64    // Override these two methods to show that keys and values may be null
65
66    @Override public V put(@Nullable K key, @Nullable V value) {
67      return super.put(key, value);
68    }
69
70    @Override public V forcePut(@Nullable K key, @Nullable V value) {
71      return super.forcePut(key, value);
72    }
73
74    /**
75     * @serialData the number of entries, first key, first value, second key,
76     *     second value, and so on.
77     */
78    private void writeObject(ObjectOutputStream stream) throws IOException {
79      stream.defaultWriteObject();
80      Serialization.writeMap(this, stream);
81    }
83    private void readObject(ObjectInputStream stream)
84        throws IOException, ClassNotFoundException {
85      stream.defaultReadObject();
86      setDelegates(new HashMap<K, V>(), new HashMap<V, K>());
87      Serialization.populateMap(this, stream);
88    }
90    private static final long serialVersionUID = 0;
91  }
```

A.3 2.old.java

```
1 import java.util.*;
2 import java.io.*;
3 import javax.mail.*;
4 import javax.mail.internet.*;
5 import javax.activation.*;
6
7 /**
8  * sendfile will create a multipart message with the second
9  * block of the message being the given file.<p>
10 *
11 * This demonstrates how to use the FileDataSource to send
12 * a file via mail.<p>
13 *
14 * usage: <code>java sendfile <i>to from smtp file true|false</i></code>
15 * where <i>to</i> and <i>from</i> are the destination and
16 * origin email addresses, respectively, and <i>smtp</i>
17 * is the hostname of the machine that has smtp server
18 * running.  <i>file</i> is the file to send. The next parameter
19 * either turns on or turns off debugging during sending.
20 *
21 * @author   Christopher Cotton
22 */
23 public class sendfile {
24
25     public static void main(String[] args) {
26    if (args.length != 5) {
27        System.out.println("usage: java sendfile <to> <from> <smtp> ↩
   <file> true|false");
28        System.exit(1);
29    }
30
31    String to = args[0];
32    String from = args[1];
33    String host = args[2];
34    String filename = args[3];
35    boolean debug = Boolean.valueOf(args[4]).booleanValue();
36    String msgText1 = "Sending a file.\n";
37    String subject = "Sending a file";
38
39    // create some properties and get the default Session
40    Properties props = System.getProperties();
41    props.put("mail.smtp.host", host);
42
43    Session session = Session.getInstance(props, null);
44    session.setDebug(debug);
45
```

```
46      try {
47          // create a message
48          MimeMessage msg = new MimeMessage(session);
49          msg.setFrom(new InternetAddress(from));
50          InternetAddress[] address = {new InternetAddress(to)};
51          msg.setRecipients(Message.RecipientType.TO, address);
52          msg.setSubject(subject);
53
54          // create and fill the first message part
55          MimeBodyPart mbp1 = new MimeBodyPart();
56          mbp1.setText(msgText1);
57
58          // create the second message part
59          MimeBodyPart mbp2 = new MimeBodyPart();
60
61               // attach the file to the message
62             FileDataSource fds = new FileDataSource(filename);
63          mbp2.setDataHandler(new DataHandler(fds));
64          mbp2.setFileName(fds.getName());
65
66          // create the Multipart and add its parts to it
67          Multipart mp = new MimeMultipart();
68          mp.addBodyPart(mbp1);
69          mp.addBodyPart(mbp2);
70
71          // add the Multipart to the message
72          msg.setContent(mp);
73
74          // set the Date: header
75          msg.setSentDate(new Date());
76
77          // send the message
78          Transport.send(msg);
79
80      } catch (MessagingException mex) {
81          mex.printStackTrace();
82          Exception ex = null;
83          if ((ex = mex.getNextException()) != null) {
84             ex.printStackTrace();
85          }
86      }
87       }
88  }
```

A.4 2.new.java

```
1 import java.util.*;
2 import java.io.*;
3 import javax.mail.*;
4 import javax.mail.internet.*;
5 import javax.activation.*;
6
7 /**
8  * sendfile will create a multipart message with the second
9  * block of the message being the given file.<p>
10 *
11 * This demonstrates how to use the FileDataSource to send
12 * a file via mail.<p>
13 *
14 * usage: <code>java sendfile <i>to from smtp file true|false</i></code>
15 * where <i>to</i> and <i>from</i> are the destination and
16 * origin email addresses, respectively, and <i>smtp</i>
17 * is the hostname of the machine that has smtp server
18 * running.  <i>file</i> is the file to send. The next parameter
19 * either turns on or turns off debugging during sending.
20 *
21 * @author   Christopher Cotton
22 */
23 public class sendfile {
24
25     public static void main(String[] args) {
26    if (args.length != 5) {
27        System.out.println("usage: java sendfile <to> <from> <smtp> ↩
  <file> true|false");
28        System.exit(1);
29    }
30
31    String to = args[0];
32    String from = args[1];
33    String host = args[2];
34    String filename = args[3];
35    boolean debug = Boolean.valueOf(args[4]).booleanValue();
36    String msgText1 = "Sending a file.\n";
37    String subject = "Sending a file";
38
39    // create some properties and get the default Session
40    Properties props = System.getProperties();
41    props.put("mail.smtp.host", host);
42
43    Session session = Session.getInstance(props, null);
44    session.setDebug(debug);
45
```

```
46    try {
47        // create a message
48        MimeMessage msg = new MimeMessage(session);
49        msg.setFrom(new InternetAddress(from));
50        InternetAddress[] address = {new InternetAddress(to)};
51        msg.setRecipients(Message.RecipientType.TO, address);
52        msg.setSubject(subject);
53
54        // create and fill the first message part
55        MimeBodyPart mbp1 = new MimeBodyPart();
56        mbp1.setText(msgText1);
57
58        // create the second message part
59        MimeBodyPart mbp2 = new MimeBodyPart();
60
61             // attach the file to the message
62        mbp2.attachFile(filename);
63
64        /*
65         * Use the following approach instead of the above line if
66         * you want to control the MIME type of the attached file.
67         * Normally you should never need to do this.
68         *
69           FileDataSource fds = new FileDataSource(filename) {
70       public String getContentType() {
71           return "application/octet-stream";
72       }
73        };
74        mbp2.setDataHandler(new DataHandler(fds));
75        mbp2.setFileName(fds.getName());
76         */
77
78        // create the Multipart and add its parts to it
79        Multipart mp = new MimeMultipart();
80        mp.addBodyPart(mbp1);
81        mp.addBodyPart(mbp2);
82
83        // add the Multipart to the message
84        msg.setContent(mp);
85
86        // set the Date: header
87        msg.setSentDate(new Date());
88
89        /*
90         * If you want to control the Content-Transfer-Encoding
91         * of the attached file, do the following.  Normally you
92         * should never need to do this.
93         *
```

```
94          msg.saveChanges();
95          mbp2.setHeader("Content-Transfer-Encoding", "base64");
96           */
97
98          // send the message
99          Transport.send(msg);
100
101     } catch (MessagingException mex) {
102         mex.printStackTrace();
103         Exception ex = null;
104         if ((ex = mex.getNextException()) != null) {
105        ex.printStackTrace();
106         }
107     } catch (IOException ioex) {
108         ioex.printStackTrace();
109     }
110      }
111 }
```

A.5 3.old.java

```
1 /*******************************************************************************
  ********
2  * Copyright (c) 2000, 2005 IBM Corporation and others.
3  * All rights reserved. This program and the accompanying materials
4  * are made available under the terms of the Eclipse Public License v1.0
5  * which accompanies this distribution, and is available at
6  * http://www.eclipse.org/legal/epl-v10.html
7  *
8  * Contributors:
9  *     IBM Corporation - initial API and implementation
10 *******************************************************************************
  ********/
11 package org.eclipse.jface.text;
12
13 import java.util.Iterator;
14
15 import org.eclipse.jface.text.source.Annotation;
16 import org.eclipse.jface.text.source.ISourceViewer;
17
18 /**
19  * Standard implementation of {@link org.eclipse.jface.text.ITextHover}.
20  * <p>
21  * XXX: This is work in progress and can change anytime until API for 3.2 is frozen.
22  * </p>
23  *
24  * @since 3.2
25  */
26 public class DefaultTextHover implements ITextHover {
27
28    /** This hover's source viewer */
29    private ISourceViewer fSourceViewer;
30
31    /**
32     * Creates a new annotation hover.
33     *
34     * @param sourceViewer this hover's annotation model
35     */
36    public DefaultTextHover(ISourceViewer sourceViewer)  {
37       Assert.isNotNull(sourceViewer);
38       fSourceViewer= sourceViewer;
39    }
40
41    /*
42     * @see org.eclipse.jface.text.ITextHover#getHoverInfo(org.eclipse.jface.text.ITextViewer, org.eclipse.jface.text.IRegion)
```

```
43     */
44    public String getHoverInfo(ITextViewer textViewer, IRegion ↩
   hoverRegion) {
45
46       Iterator e= fSourceViewer.getAnnotationModel().↩
   getAnnotationIterator();
47       while (e.hasNext()) {
48          Annotation a= (Annotation) e.next();
49          if (isIncluded(a)) {
50             Position p= fSourceViewer.getAnnotationModel().getPosition(a);
51             if (p != null && p.overlapsWith(hoverRegion.getOffset(), ↩
   hoverRegion.getLength())) {
52                String msg= a.getText();
53                if (msg != null && msg.trim().length() > 0)
54                   return msg;
55             }
56          }
57       }
58
59       return null;
60    }
61
62    /*
63     * @see org.eclipse.jface.text.ITextHover#getHoverRegion(org.eclipse.↩
   jface.text.ITextViewer, int)
64     */
65    public IRegion getHoverRegion(ITextViewer textViewer, int offset) {
66       return findWord(textViewer.getDocument(), offset);
67    }
68
69    /**
70     * Tells whether the annotation should be included in
71     * the computation.
72     *
73     * @param annotation the annotation to test
74     * @return <code>true</code> if the annotation is included in the ↩
   computation
75     */
76    protected boolean isIncluded(Annotation annotation) {
77       return true;
78    }
79
80    private IRegion findWord(IDocument document, int offset) {
81       int start= -1;
82       int end= -1;
83
84       try {
85
```

```
86          int pos= offset;
87          char c;
88
89          while (pos >= 0) {
90             c= document.getChar(pos);
91             if (!Character.isUnicodeIdentifierPart(c))
92                break;
93             --pos;
94          }
95
96          start= pos;
97
98          pos= offset;
99          int length= document.getLength();
100
101         while (pos < length) {
102            c= document.getChar(pos);
103            if (!Character.isUnicodeIdentifierPart(c))
104               break;
105            ++pos;
106         }
107
108         end= pos;
109
110      } catch (BadLocationException x) {
111      }
112
113      if (start > -1 && end > -1) {
114         if (start == offset && end == offset)
115            return new Region(offset, 0);
116         else if (start == offset)
117            return new Region(start, end - start);
118         else
119            return new Region(start + 1, end - start - 1);
120      }
121
122      return null;
123   }
124 }
```

A.6 3.new.java

```java
/*******************************************************************************
 * Copyright (c) 2005, 2008 IBM Corporation and others.
 * All rights reserved. This program and the accompanying materials
 * are made available under the terms of the Eclipse Public License v1.0
 * which accompanies this distribution, and is available at
 * http://www.eclipse.org/legal/epl-v10.html
 *
 * Contributors:
 *     IBM Corporation - initial API and implementation
 *******************************************************************************/
package org.eclipse.jface.text;

import java.util.Iterator;

import org.eclipse.core.runtime.Assert;

import org.eclipse.jface.text.source.Annotation;
import org.eclipse.jface.text.source.IAnnotationModel;
import org.eclipse.jface.text.source.ISourceViewer;
import org.eclipse.jface.text.source.ISourceViewerExtension2;

/**
 * Standard implementation of {@link org.eclipse.jface.text.ITextHover}.
 *
 * @since 3.2
 */
public class DefaultTextHover implements ITextHover {

   /** This hover's source viewer */
   private ISourceViewer fSourceViewer;

   /**
    * Creates a new annotation hover.
    *
    * @param sourceViewer this hover's annotation model
    */
   public DefaultTextHover(ISourceViewer sourceViewer)  {
      Assert.isNotNull(sourceViewer);
      fSourceViewer= sourceViewer;
   }

   /**
    * {@inheritDoc}
    *
```

```
45     * @deprecated As of 3.4, replaced by {@link ITextHoverExtension2#↩
   getHoverInfo2(ITextViewer, IRegion)}
46     */
47    public String getHoverInfo(ITextViewer textViewer, IRegion ↩
   hoverRegion) {
48       IAnnotationModel model= getAnnotationModel(fSourceViewer);
49       if (model == null)
50          return null;
51
52       Iterator e= model.getAnnotationIterator();
53       while (e.hasNext()) {
54          Annotation a= (Annotation) e.next();
55          if (isIncluded(a)) {
56             Position p= model.getPosition(a);
57             if (p != null && p.overlapsWith(hoverRegion.getOffset(), ↩
   hoverRegion.getLength())) {
58                String msg= a.getText();
59                if (msg != null && msg.trim().length() > 0)
60                   return msg;
61             }
62          }
63       }
64
65       return null;
66    }
67
68    /*
69     * @see org.eclipse.jface.text.ITextHover#getHoverRegion(org.eclipse.↩
   jface.text.ITextViewer, int)
70     */
71    public IRegion getHoverRegion(ITextViewer textViewer, int offset) {
72       return findWord(textViewer.getDocument(), offset);
73    }
74
75    /**
76     * Tells whether the annotation should be included in
77     * the computation.
78     *
79     * @param annotation the annotation to test
80     * @return <code>true</code> if the annotation is included in the ↩
   computation
81     */
82    protected boolean isIncluded(Annotation annotation) {
83       return true;
84    }
85
86    private IAnnotationModel getAnnotationModel(ISourceViewer viewer) {
87       if (viewer instanceof ISourceViewerExtension2) {
```

```
88          ISourceViewerExtension2 extension= (ISourceViewerExtension2) ↩
   viewer;
89          return extension.getVisualAnnotationModel();
90       }
91       return viewer.getAnnotationModel();
92    }
93
94    private IRegion findWord(IDocument document, int offset) {
95       int start= -2;
96       int end= -1;
97
98       try {
99
100         int pos= offset;
101         char c;
102
103         while (pos >= 0) {
104            c= document.getChar(pos);
105            if (!Character.isUnicodeIdentifierPart(c))
106               break;
107            --pos;
108         }
109
110         start= pos;
111
112         pos= offset;
113         int length= document.getLength();
114
115         while (pos < length) {
116            c= document.getChar(pos);
117            if (!Character.isUnicodeIdentifierPart(c))
118               break;
119            ++pos;
120         }
121
122         end= pos;
123
124      } catch (BadLocationException x) {
125      }
126
127      if (start >= -1 && end > -1) {
128         if (start == offset && end == offset)
129            return new Region(offset, 0);
130         else if (start == offset)
131            return new Region(start, end - start);
132         else
133            return new Region(start + 1, end - start - 1);
134      }
```

```
135
136         return null;
137     }
138 }
```

A.7 4.old.java

```
// Copyright (c) Corporation for National Research Initiatives
package org.python.core;

import java.security.SecureClassLoader;
import java.util.ArrayList;
import java.util.List;
import java.util.Vector;

/**
 * Utility class for loading of compiled python modules and java classes defined in python modules.
 */
public class BytecodeLoader {

    /**
     * Turn the java byte code in data into a java class.
     *
     * @param name
     *            the name of the class
     * @param data
     *            the java byte code.
     * @param referents
     *            superclasses and interfaces that the new class will reference.
     */
    public static Class makeClass(String name, byte[] data, Class... referents) {
        Loader loader = new Loader();
        for (int i = 0; i < referents.length; i++) {
            try {
                ClassLoader cur = referents[i].getClassLoader();
                if (cur != null) {
                    loader.addParent(cur);
                }
            } catch (SecurityException e) {}
        }
        return loader.loadClassFromBytes(name, data);
    }

    /**
     * Turn the java byte code in data into a java class.
     *
     * @param name
     *            the name of the class
     * @param referents
     *            superclasses and interfaces that the new class will
```

```
reference.
     * @param data
     *            the java byte code.
     */
    public static Class makeClass(String name, Vector<Class> referents, byte[] data) {
        if (referents != null) {
            return makeClass(name, data, referents.toArray(new Class[0]));
        }
        return makeClass(name, data);
    }

    /**
     * Turn the java byte code for a compiled python module into a java class.
     *
     * @param name
     *            the name of the class
     * @param data
     *            the java byte code.
     */
    public static PyCode makeCode(String name, byte[] data, String filename) {
        try {
            Class c = makeClass(name, data);
            @SuppressWarnings("unchecked")
            Object o = c.getConstructor(new Class[] {String.class})
                    .newInstance(new Object[] {filename});
            return ((PyRunnable)o).getMain();
        } catch (Exception e) {
            throw Py.JavaError(e);
        }
    }

    public static class Loader extends SecureClassLoader {

        private List<ClassLoader> parents = new ArrayList<ClassLoader>();

        public Loader() {
            parents.add(imp.getSyspathJavaLoader());
        }

        public void addParent(ClassLoader referent) {
            if (!parents.contains(referent)) {
                parents.add(0, referent);
            }
        }

```

```
88          protected Class<?> loadClass(String name, boolean resolve) ↩
    throws ClassNotFoundException {
89              Class c = findLoadedClass(name);
90              if (c != null) {
91                  return c;
92              }
93              for (ClassLoader loader : parents) {
94                  try {
95                      return loader.loadClass(name);
96                  } catch (ClassNotFoundException cnfe) {}
97              }
98              // couldn't find the .class file on sys.path
99              throw new ClassNotFoundException(name);
100         }
101
102         public Class loadClassFromBytes(String name, byte[] data) {
103             Class c = defineClass(name, data, 0, data.length, ↩
    getClass().getProtectionDomain());
104             resolveClass(c);
105             Compiler.compileClass(c);
106             return c;
107         }
108     }
109 }
```

A.8 4.new.java

```
1 // Copyright (c) Corporation for National Research Initiatives
2 package org.python.core;
3
4 import java.security.SecureClassLoader;
5 import java.util.List;
6
7 import org.python.objectweb.asm.ClassReader;
8 import org.python.util.Generic;
9
10 /**
11  * Utility class for loading of compiled python modules and java ↩
  classes defined in python modules.
12  */
13 public class BytecodeLoader {
14
15     /**
16      * Turn the java byte code in data into a java class.
17      *
18      * @param name
19      *            the name of the class
20      * @param data
21      *            the java byte code.
22      * @param referents
23      *            superclasses and interfaces that the new class will ↩
  reference.
24      */
25     public static Class<?> makeClass(String name, byte[] data, ↩
  Class<?>... referents) {
26         Loader loader = new Loader();
27         for (Class<?> referent : referents) {
28             try {
29                 ClassLoader cur = referent.getClassLoader();
30                 if (cur != null) {
31                     loader.addParent(cur);
32                 }
33             } catch (SecurityException e) {}
34         }
35         return loader.loadClassFromBytes(name, data);
36     }
37
38     /**
39      * Turn the java byte code in data into a java class.
40      *
41      * @param name
42      *            the name of the class
43      * @param referents
```

```
44      *            superclasses and interfaces that the new class will ↩
    reference.
45      * @param data
46      *            the java byte code.
47      */
48     public static Class<?> makeClass(String name, List<Class<?>> ↩
    referents, byte[] data) {
49         if (referents != null) {
50             return makeClass(name, data, referents.toArray(new ↩
    Class[referents.size()]));
51         }
52         return makeClass(name, data);
53     }
54 
55     /**
56      * Turn the java byte code for a compiled python module into a java ↩
    class.
57      *
58      * @param name
59      *            the name of the class
60      * @param data
61      *            the java byte code.
62      */
63     public static PyCode makeCode(String name, byte[] data, String ↩
    filename) {
64         try {
65             Class<?> c = makeClass(name, data);
66             Object o = c.getConstructor(new Class[] {String.class})
67                     .newInstance(new Object[] {filename});
68             return ((PyRunnable)o).getMain();
69         } catch (Exception e) {
70             throw Py.JavaError(e);
71         }
72     }
73 
74     public static class Loader extends SecureClassLoader {
75 
76         private List<ClassLoader> parents = Generic.list();
77 
78         public Loader() {
79             parents.add(imp.getSyspathJavaLoader());
80         }
81 
82         public void addParent(ClassLoader referent) {
83             if (!parents.contains(referent)) {
84                 parents.add(0, referent);
85             }
86         }
```

```
87
88          @Override
89          protected Class<?> loadClass(String name, boolean resolve) ↩
    throws ClassNotFoundException {
90              Class<?> c = findLoadedClass(name);
91              if (c != null) {
92                  return c;
93              }
94              for (ClassLoader loader : parents) {
95                  try {
96                      return loader.loadClass(name);
97                  } catch (ClassNotFoundException cnfe) {}
98              }
99              // couldn't find the .class file on sys.path
100             throw new ClassNotFoundException(name);
101         }
102
103         public Class<?> loadClassFromBytes(String name, byte[] data) {
104             if (name.endsWith("$py")) {
105                 try {
106                     // Get the real class name: we might request a 'bar'
107                     // Jython module that was compiled as 'foo.bar', or
108                     // even 'baz.__init__' which is compiled as just 'baz'
109                     ClassReader cr = new ClassReader(data);
110                     name = cr.getClassName().replace('/', '.');
111                 } catch (RuntimeException re) {
112                     // Probably an invalid .class, fallback to the
113                     // specified name
114                 }
115             }
116             Class<?> c = defineClass(name, data, 0, data.length, ↩
    getClass().getProtectionDomain());
117             resolveClass(c);
118             Compiler.compileClass(c);
119             return c;
120         }
121     }
122 }
```

A.9 5.old.java

```
1 package org.springframework.samples.imagedb;
2
3 import java.io.IOException;
4 import java.io.InputStream;
5 import java.io.OutputStream;
6 import java.sql.PreparedStatement;
7 import java.sql.ResultSet;
8 import java.sql.SQLException;
9 import java.util.List;
10
11 import org.springframework.dao.DataAccessException;
12 import org.springframework.dao.IncorrectResultSizeDataAccessException;
13 import org.springframework.jdbc.LobRetrievalFailureException;
14 import org.springframework.jdbc.core.RowMapper;
15 import org.springframework.jdbc.core.support.↩
   AbstractLobCreatingPreparedStatementCallback;
16 import org.springframework.jdbc.core.support.↩
   AbstractLobStreamingResultSetExtractor;
17 import org.springframework.jdbc.core.support.JdbcDaoSupport;
18 import org.springframework.jdbc.support.lob.LobCreator;
19 import org.springframework.jdbc.support.lob.LobHandler;
20 import org.springframework.util.FileCopyUtils;
21
22 /**
23  * Default implementation of the central image database business ↩
   interface.
24  *
25  * <p>Uses JDBC with a LobHandler to retrieve and store image data.
26  * Illustrates direct use of the jdbc.core package, i.e. JdbcTemplate,
27  * rather than operation objects from the jdbc.object package.
28  *
29  * @author Juergen Hoeller
30  * @since 07.01.2004
31  * @see org.springframework.jdbc.core.JdbcTemplate
32  * @see org.springframework.jdbc.support.lob.LobHandler
33  */
34 public class DefaultImageDatabase extends JdbcDaoSupport implements ↩
   ImageDatabase {
35
36    private LobHandler lobHandler;
37
38    /**
39     * Set the LobHandler to use for BLOB/CLOB access.
40     * Could use a DefaultLobHandler instance as default,
41     * but relies on a specified LobHandler here.
42     * @see org.springframework.jdbc.support.lob.DefaultLobHandler
```

```
43     */
44    public void setLobHandler(LobHandler lobHandler) {
45       this.lobHandler = lobHandler;
46    }
47
48    public List getImages() throws DataAccessException {
49       return getJdbcTemplate().query(
50           "SELECT image_name, description FROM imagedb",
51           new RowMapper() {
52              public Object mapRow(ResultSet rs, int rowNum) throws ↩
SQLException {
53                 String name = rs.getString(1);
54                 String description = lobHandler.getClobAsString(rs, 2);
55                 return new ImageDescriptor(name, description);
56              }
57           });
58    }
59
60    public void streamImage(final String name, final OutputStream ↩
contentStream) throws DataAccessException {
61       getJdbcTemplate().query(
62             "SELECT content FROM imagedb WHERE image_name=?", new ↩
Object[] {name},
63             new AbstractLobStreamingResultSetExtractor() {
64                protected void handleNoRowFound() throws ↩
LobRetrievalFailureException {
65                   throw new IncorrectResultSizeDataAccessException(
66                       "Image with name '" + name + "' not found in ↩
database", 1, 0);
67                }
68                public void streamData(ResultSet rs) throws ↩
SQLException, IOException {
69                   InputStream is = lobHandler.getBlobAsBinaryStream(rs, ↩
1);
70                   if (is != null) {
71                      FileCopyUtils.copy(is, contentStream);
72                   }
73                }
74             }
75       );
76    }
77
78    public void storeImage(
79        final String name, final InputStream contentStream, final int ↩
contentLength, final String description)
80        throws DataAccessException {
81       getJdbcTemplate().execute(
82             "INSERT INTO imagedb (image_name, content, description) ↩
```

```
   VALUES (?, ?, ?)",
83             new AbstractLobCreatingPreparedStatementCallback(this.↩
   lobHandler) {
84                protected void setValues(PreparedStatement ps, ↩
   LobCreator lobCreator) throws SQLException {
85                   ps.setString(1, name);
86                   lobCreator.setBlobAsBinaryStream(ps, 2, ↩
   contentStream, contentLength);
87                   lobCreator.setClobAsString(ps, 3, description);
88                }
89             }
90       );
91    }
92
93    public void checkImages() {
94       // could implement consistency check here
95       logger.info("Checking images: not implemented but invoked by ↩
   scheduling");
96    }
97
98    public void clearDatabase() throws DataAccessException {
99       getJdbcTemplate().update("DELETE FROM imagedb");
100   }
101
102 }
```

A.10 5.new.java

```
1 package org.springframework.samples.imagedb;
2
3 import java.io.IOException;
4 import java.io.InputStream;
5 import java.io.OutputStream;
6 import java.sql.PreparedStatement;
7 import java.sql.ResultSet;
8 import java.sql.SQLException;
9 import java.util.List;
10
11 import org.springframework.dao.DataAccessException;
12 import org.springframework.dao.EmptyResultDataAccessException;
13 import org.springframework.jdbc.LobRetrievalFailureException;
14 import org.springframework.jdbc.core.simple.ParameterizedRowMapper;
15 import org.springframework.jdbc.core.simple.SimpleJdbcDaoSupport;
16 import org.springframework.jdbc.core.support.↩
   AbstractLobCreatingPreparedStatementCallback;
17 import org.springframework.jdbc.core.support.↩
   AbstractLobStreamingResultSetExtractor;
18 import org.springframework.jdbc.support.lob.LobCreator;
19 import org.springframework.jdbc.support.lob.LobHandler;
20 import org.springframework.transaction.annotation.Transactional;
21 import org.springframework.util.FileCopyUtils;
22
23 /**
24  * Default implementation of the central image database business ↩
   interface.
25  *
26  * <p>Uses JDBC with a LobHandler to retrieve and store image data.
27  * Illustrates direct use of the <code>jdbc.core</code> package,
28  * i.e. JdbcTemplate, rather than operation objects from the
29  * <code>jdbc.object</code> package.
30  *
31  * @author Juergen Hoeller
32  * @since 07.01.2004
33  * @see org.springframework.jdbc.core.JdbcTemplate
34  * @see org.springframework.jdbc.support.lob.LobHandler
35  */
36 public class DefaultImageDatabase extends SimpleJdbcDaoSupport ↩
   implements ImageDatabase {
37
38    private LobHandler lobHandler;
39
40    /**
41     * Set the LobHandler to use for BLOB/CLOB access.
42     * Could use a DefaultLobHandler instance as default,
```

```
43      * but relies on a specified LobHandler here.
44      * @see org.springframework.jdbc.support.lob.DefaultLobHandler
45      */
46     public void setLobHandler(LobHandler lobHandler) {
47        this.lobHandler = lobHandler;
48     }
49
50     @Transactional(readOnly=true)
51     public List<ImageDescriptor> getImages() throws DataAccessException {
52        return getSimpleJdbcTemplate().query(
53            "SELECT image_name, description FROM imagedb",
54            new ParameterizedRowMapper<ImageDescriptor>() {
55               public ImageDescriptor mapRow(ResultSet rs, int rowNum) ↩
   throws SQLException {
56                  String name = rs.getString(1);
57                  String description = lobHandler.getClobAsString(rs, 2);
58                  return new ImageDescriptor(name, description);
59               }
60            });
61     }
62
63     @Transactional(readOnly=true)
64     public void streamImage(final String name, final OutputStream ↩
   contentStream) throws DataAccessException {
65        getJdbcTemplate().query(
66              "SELECT content FROM imagedb WHERE image_name=?", new ↩
   Object[] {name},
67              new AbstractLobStreamingResultSetExtractor() {
68                 protected void handleNoRowFound() throws ↩
   LobRetrievalFailureException {
69                    throw new EmptyResultDataAccessException(
70                        "Image with name '" + name + "' not found in ↩
   database", 1);
71                 }
72                 public void streamData(ResultSet rs) throws ↩
   SQLException, IOException {
73                    InputStream is = lobHandler.getBlobAsBinaryStream(rs, ↩
   1);
74                    if (is != null) {
75                       FileCopyUtils.copy(is, contentStream);
76                    }
77                 }
78              }
79        );
80     }
81
82     @Transactional
83     public void storeImage(
```

```
84        final String name, final InputStream contentStream, final int ↩
   contentLength, final String description)
85        throws DataAccessException {
86
87       getJdbcTemplate().execute(
88             "INSERT INTO imagedb (image_name, content, description) ↩
   VALUES (?, ?, ?)",
89             new AbstractLobCreatingPreparedStatementCallback(this.↩
   lobHandler) {
90                protected void setValues(PreparedStatement ps, ↩
   LobCreator lobCreator) throws SQLException {
91                   ps.setString(1, name);
92                   lobCreator.setBlobAsBinaryStream(ps, 2, ↩
   contentStream, contentLength);
93                   lobCreator.setClobAsString(ps, 3, description);
94                }
95             }
96       );
97    }
98
99    public void checkImages() {
100      // Could implement consistency check here...
101      logger.info("Checking images: not implemented but invoked by ↩
   scheduling");
102   }
103
104   @Transactional
105   public void clearDatabase() throws DataAccessException {
106      getJdbcTemplate().update("DELETE FROM imagedb");
107   }
108
109 }
```

A.11 6.old.java

```
1 /**
2  *
3  */
4 package org.junit.experimental.theories;
5
6 import java.lang.reflect.Field;
7 import java.lang.reflect.InvocationTargetException;
8 import java.lang.reflect.Modifier;
9 import java.util.ArrayList;
10 import java.util.List;
11
12 import org.junit.Assume;
13 import org.junit.Assume.AssumptionViolatedException;
14 import org.junit.experimental.theories.PotentialAssignment.↩
  CouldNotGenerateValueException;
15 import org.junit.experimental.theories.internal.Assignments;
16 import org.junit.experimental.theories.internal.↩
  ParameterizedAssertionError;
17 import org.junit.internal.runners.InitializationError;
18 import org.junit.internal.runners.JUnit4ClassRunner;
19 import org.junit.internal.runners.links.Statement;
20 import org.junit.internal.runners.model.FrameworkMethod;
21
22 @SuppressWarnings("restriction")
23 public class Theories extends JUnit4ClassRunner {
24    public Theories(Class<?> klass) throws InitializationError {
25       super(klass);
26    }
27
28    @Override
29    protected void collectInitializationErrors(List<Throwable> errors) {
30       Field[] fields= getTestClass().getJavaClass().getDeclaredFields();
31
32       for (Field each : fields)
33          if (each.getAnnotation(DataPoint.class) != null && !Modifier.↩
  isStatic(each.getModifiers()))
34             errors.add(new Error("DataPoint field " + each.getName() + ↩
  " must be static"));
35    }
36
37    @Override
38    protected List<FrameworkMethod> computeTestMethods() {
39       List<FrameworkMethod> testMethods= super.computeTestMethods();
40       List<FrameworkMethod> theoryMethods= getTestClass().↩
  getAnnotatedMethods(Theory.class);
41       testMethods.removeAll(theoryMethods);
```

```
42        testMethods.addAll(theoryMethods);
43        return testMethods;
44     }
45
46     @Override
47     public Statement childBlock(final FrameworkMethod method) {
48        return new TheoryAnchor(method);
49     }
50
51     public class TheoryAnchor extends Statement {
52        private int successes= 0;
53
54        private FrameworkMethod fTestMethod;
55
56        private List<AssumptionViolatedException> fInvalidParameters= new ↩
   ArrayList<AssumptionViolatedException>();
57
58        public TheoryAnchor(FrameworkMethod method) {
59           fTestMethod= method;
60        }
61
62        @Override
63        public void evaluate() throws Throwable {
64           runWithAssignment(Assignments.allUnassigned(
65                 fTestMethod.getMethod(), getTestClass().getJavaClass()));
66
67           if (successes == 0)
68              Assume
69                    .fail("Never found parameters that satisfied method. ↩
   Violated assumptions: "
70                          + fInvalidParameters);
71        }
72
73        protected void runWithAssignment(Assignments parameterAssignment)
74              throws Throwable {
75           if (!parameterAssignment.isComplete()) {
76              runWithIncompleteAssignment(parameterAssignment);
77           } else {
78              runWithCompleteAssignment(parameterAssignment);
79           }
80        }
81
82        protected void runWithIncompleteAssignment(Assignments incomplete)
83              throws InstantiationException, IllegalAccessException,
84              Throwable {
85           for (PotentialAssignment source : incomplete
86                 .potentialsForNextUnassigned()) {
87              runWithAssignment(incomplete.assignNext(source));
```

```
88          }
89       }
90
91       protected void runWithCompleteAssignment(final Assignments complete)
92             throws InstantiationException, IllegalAccessException,
93             InvocationTargetException, NoSuchMethodException, Throwable {
94          new JUnit4ClassRunner(getTestClass().getJavaClass()) {
95             @Override
96             protected void collectInitializationErrors(
97                   List<Throwable> errors) {
98                // do nothing
99             }
100
101            @Override
102            public Statement childBlock(FrameworkMethod method) {
103               final Statement statement= super.childBlock(method);
104               return new Statement() {
105                  @Override
106                  public void evaluate() throws Throwable {
107                     try {
108                        statement.evaluate();
109                        handleDataPointSuccess();
110                     } catch (AssumptionViolatedException e) {
111                        handleAssumptionViolation(e);
112                     } catch (Throwable e) {
113                        reportParameterizedError(e, complete
114                              .getAllArguments(nullsOk()));
115                     }
116                  }
117
118               };
119            }
120
121            @Override
122            protected Statement invoke(FrameworkMethod method, Object ↩
    test) {
123               return methodCompletesWithParameters(method, complete, ↩
    test);
124            }
125
126            @Override
127            public Object createTest() throws Exception {
128               return getTestClass().getConstructor().newInstance(
129                     complete.getConstructorArguments(nullsOk()));
130            }
131         }.childBlock(fTestMethod).evaluate();
132      }
133
```

```
134       private Statement methodCompletesWithParameters(
135             final FrameworkMethod method, final Assignments complete, ↩
    final Object freshInstance) {
136          return new Statement() {
137             @Override
138             public void evaluate() throws Throwable {
139                try {
140                   final Object[] values= complete.getMethodArguments(
141                         nullsOk());
142                   method.invokeExplosively(freshInstance, values);
143                } catch (CouldNotGenerateValueException e) {
144                   // ignore
145                }
146             }
147          };
148       }
149
150       protected void handleAssumptionViolation(↩
    AssumptionViolatedException e) {
151          fInvalidParameters.add(e);
152       }
153
154       protected void reportParameterizedError(Throwable e, Object... ↩
    params)
155             throws Throwable {
156          if (params.length == 0)
157             throw e;
158          throw new ParameterizedAssertionError(e, fTestMethod.getName(),
159                params);
160       }
161
162       private boolean nullsOk() {
163          Theory annotation= fTestMethod.getMethod().getAnnotation(
164                Theory.class);
165          if (annotation == null)
166             return false;
167          return annotation.nullsAccepted();
168       }
169
170       protected void handleDataPointSuccess() {
171          successes++;
172       }
173    }
174 }
```

A.12 6.new.java

```
1 /**
2  *
3  */
4 package org.junit.experimental.theories;
5
6 import java.lang.reflect.Field;
7 import java.lang.reflect.InvocationTargetException;
8 import java.lang.reflect.Modifier;
9 import java.util.ArrayList;
10 import java.util.List;
11
12 import org.junit.Assert;
13 import org.junit.experimental.theories.PotentialAssignment.↩
CouldNotGenerateValueException;
14 import org.junit.experimental.theories.internal.Assignments;
15 import org.junit.experimental.theories.internal.↩
ParameterizedAssertionError;
16 import org.junit.internal.AssumptionViolatedException;
17 import org.junit.runners.BlockJUnit4ClassRunner;
18 import org.junit.runners.model.FrameworkMethod;
19 import org.junit.runners.model.InitializationError;
20 import org.junit.runners.model.Statement;
21
22 public class Theories extends BlockJUnit4ClassRunner {
23    public Theories(Class<?> klass) throws InitializationError {
24       super(klass);
25    }
26
27    @Override
28    protected void collectInitializationErrors(List<Throwable> errors) {
29       super.collectInitializationErrors(errors);
30       validateDataPointFields(errors);
31    }
32
33    private void validateDataPointFields(List<Throwable> errors) {
34       Field[] fields= getTestClass().getJavaClass().getDeclaredFields();
35
36       for (Field each : fields)
37          if (each.getAnnotation(DataPoint.class) != null && !Modifier.↩
isStatic(each.getModifiers()))
38             errors.add(new Error("DataPoint field " + each.getName() + ↩
" must be static"));
39    }
40
41    @Override
42    protected void validateZeroArgConstructor(List<Throwable> errors) {
```

```
43          // constructor can have args
44       }
45
46       @Override
47       protected void validateTestMethods(List<Throwable> errors) {
48          for (FrameworkMethod each : computeTestMethods())
49             each.validatePublicVoid(false, errors);
50       }
51
52       @Override
53       protected List<FrameworkMethod> computeTestMethods() {
54          List<FrameworkMethod> testMethods= super.computeTestMethods();
55          List<FrameworkMethod> theoryMethods= getTestClass().↩
   getAnnotatedMethods(Theory.class);
56          testMethods.removeAll(theoryMethods);
57          testMethods.addAll(theoryMethods);
58          return testMethods;
59       }
60
61       @Override
62       public Statement methodBlock(final FrameworkMethod method) {
63          return new TheoryAnchor(method);
64       }
65
66       public class TheoryAnchor extends Statement {
67          private int successes= 0;
68
69          private FrameworkMethod fTestMethod;
70
71          private List<AssumptionViolatedException> fInvalidParameters= new ↩
   ArrayList<AssumptionViolatedException>();
72
73          public TheoryAnchor(FrameworkMethod method) {
74             fTestMethod= method;
75          }
76
77          @Override
78          public void evaluate() throws Throwable {
79             runWithAssignment(Assignments.allUnassigned(
80                   fTestMethod.getMethod(), getTestClass()));
81
82             if (successes == 0)
83                Assert
84                      .fail("Never found parameters that satisfied method ↩
   assumptions.  Violated assumptions: "
85                            + fInvalidParameters);
86          }
87
```

```
88      protected void runWithAssignment(Assignments parameterAssignment)
89            throws Throwable {
90         if (!parameterAssignment.isComplete()) {
91            runWithIncompleteAssignment(parameterAssignment);
92         } else {
93            runWithCompleteAssignment(parameterAssignment);
94         }
95      }
96
97      protected void runWithIncompleteAssignment(Assignments incomplete)
98            throws InstantiationException, IllegalAccessException,
99            Throwable {
100        for (PotentialAssignment source : incomplete
101              .potentialsForNextUnassigned()) {
102           runWithAssignment(incomplete.assignNext(source));
103        }
104     }
105
106     protected void runWithCompleteAssignment(final Assignments complete)
107           throws InstantiationException, IllegalAccessException,
108           InvocationTargetException, NoSuchMethodException, Throwable {
109        new BlockJUnit4ClassRunner(getTestClass().getJavaClass()) {
110           @Override
111           protected void collectInitializationErrors(
112                 List<Throwable> errors) {
113              // do nothing
114           }
115
116           @Override
117           public Statement methodBlock(FrameworkMethod method) {
118              final Statement statement= super.methodBlock(method);
119              return new Statement() {
120                 @Override
121                 public void evaluate() throws Throwable {
122                    try {
123                       statement.evaluate();
124                       handleDataPointSuccess();
125                    } catch (AssumptionViolatedException e) {
126                       handleAssumptionViolation(e);
127                    } catch (Throwable e) {
128                       reportParameterizedError(e, complete
129                             .getArgumentStrings(nullsOk()));
130                    }
131                 }
132
133              };
134           }
135
```

```
136             @Override
137             protected Statement methodInvoker(FrameworkMethod method, ↩
    Object test) {
138                return methodCompletesWithParameters(method, complete, ↩
    test);
139             }
140
141             @Override
142             public Object createTest() throws Exception {
143                return getTestClass().getOnlyConstructor().newInstance(
144                      complete.getConstructorArguments(nullsOk()));
145             }
146          }.methodBlock(fTestMethod).evaluate();
147       }
148
149       private Statement methodCompletesWithParameters(
150             final FrameworkMethod method, final Assignments complete, ↩
    final Object freshInstance) {
151          return new Statement() {
152             @Override
153             public void evaluate() throws Throwable {
154                try {
155                   final Object[] values= complete.getMethodArguments(
156                         nullsOk());
157                   method.invokeExplosively(freshInstance, values);
158                } catch (CouldNotGenerateValueException e) {
159                   // ignore
160                }
161             }
162          };
163       }
164
165       protected void handleAssumptionViolation(↩
    AssumptionViolatedException e) {
166          fInvalidParameters.add(e);
167       }
168
169       protected void reportParameterizedError(Throwable e, Object... ↩
    params)
170             throws Throwable {
171          if (params.length == 0)
172             throw e;
173          throw new ParameterizedAssertionError(e, fTestMethod.getName(),
174                params);
175       }
176
177       private boolean nullsOk() {
178          Theory annotation= fTestMethod.getMethod().getAnnotation(
```

```
179                 Theory.class);
180           if (annotation == null)
181              return false;
182           return annotation.nullsAccepted();
183        }
184
185        protected void handleDataPointSuccess() {
186           successes++;
187        }
188     }
189  }
```