Session 3

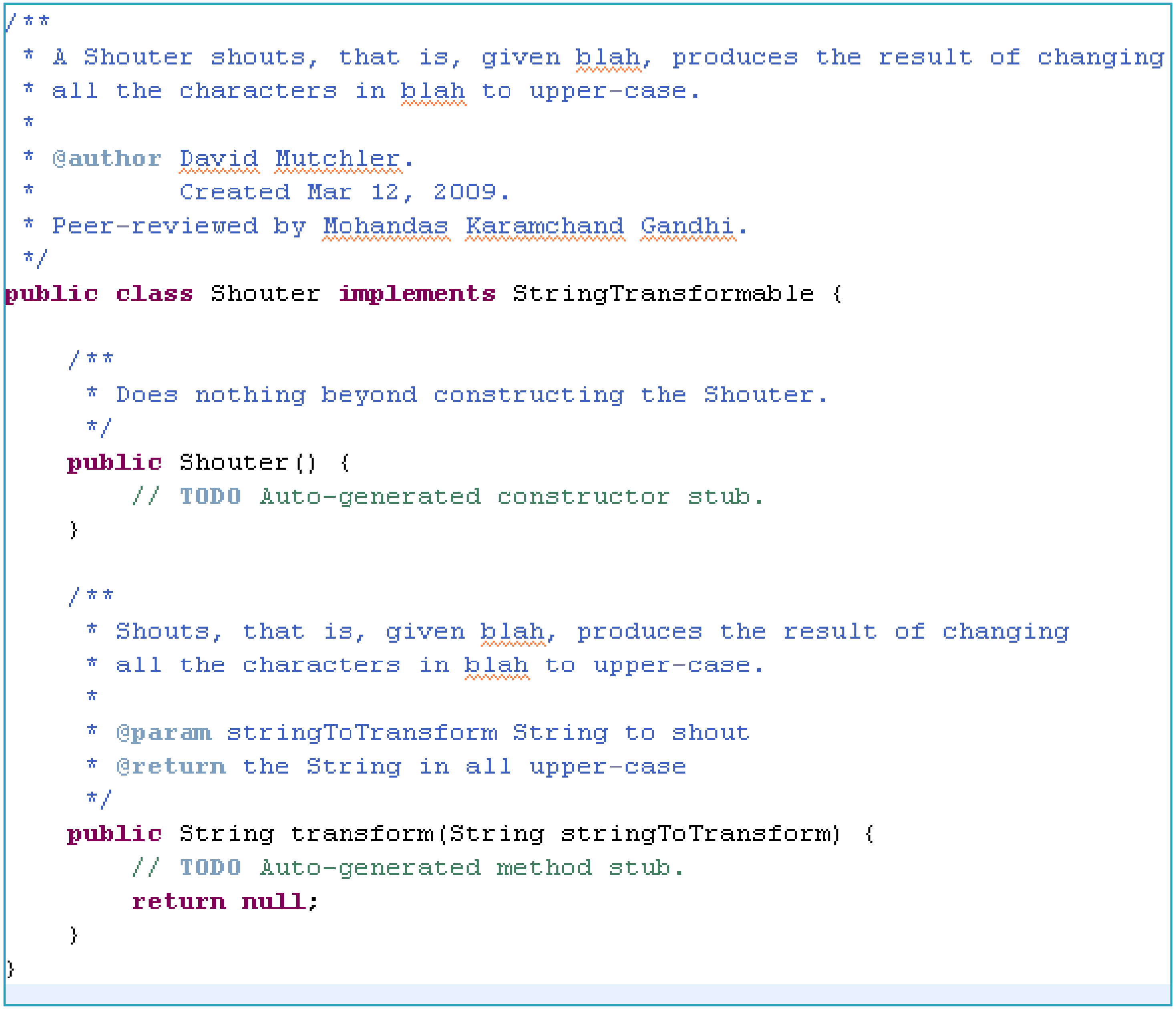
- Implementing a class
  - Implementing an **interface**
  - Using **documented stubs** before coding
  - Writing **JUnit tests** before coding
  - Using **fields**

Shouter

- Step 1: Create the Shouter class with documented stubs

Do you see:

- What is a ***stub***?
- What is a ***documented*** stub?
- What must you document? (Answer: every *public* thing)
- What is the form for a Javadoc?
- What does *"implements StringTransformable"* mean? Why is it important?
- What is the form of this class?
- What is a ***constructor***?



```
/**
 * A Shouter shouts, that is, given blah, produces the result of changing
 * all the characters in blah to upper-case.
 *
 * @author David Mutchler.
 *         Created Mar 12, 2009.
 * Peer-reviewed by Mohandas Karamchand Gandhi.
 */
public class Shouter implements StringTransformable {

    /**
     * Does nothing beyond constructing the Shouter.
     */
    public Shouter() {
        // TODO Auto-generated constructor stub.
    }

    /**
     * Shouts, that is, given blah, produces the result of changing
     * all the characters in blah to upper-case.
     *
     * @param stringToTransform String to shout
     * @return the String in all upper-case
     */
    public String transform(String stringToTransform) {
        // TODO Auto-generated method stub.
        return null;
    }
}
```

Questions on the above?
Did you get yours peer-reviewed?

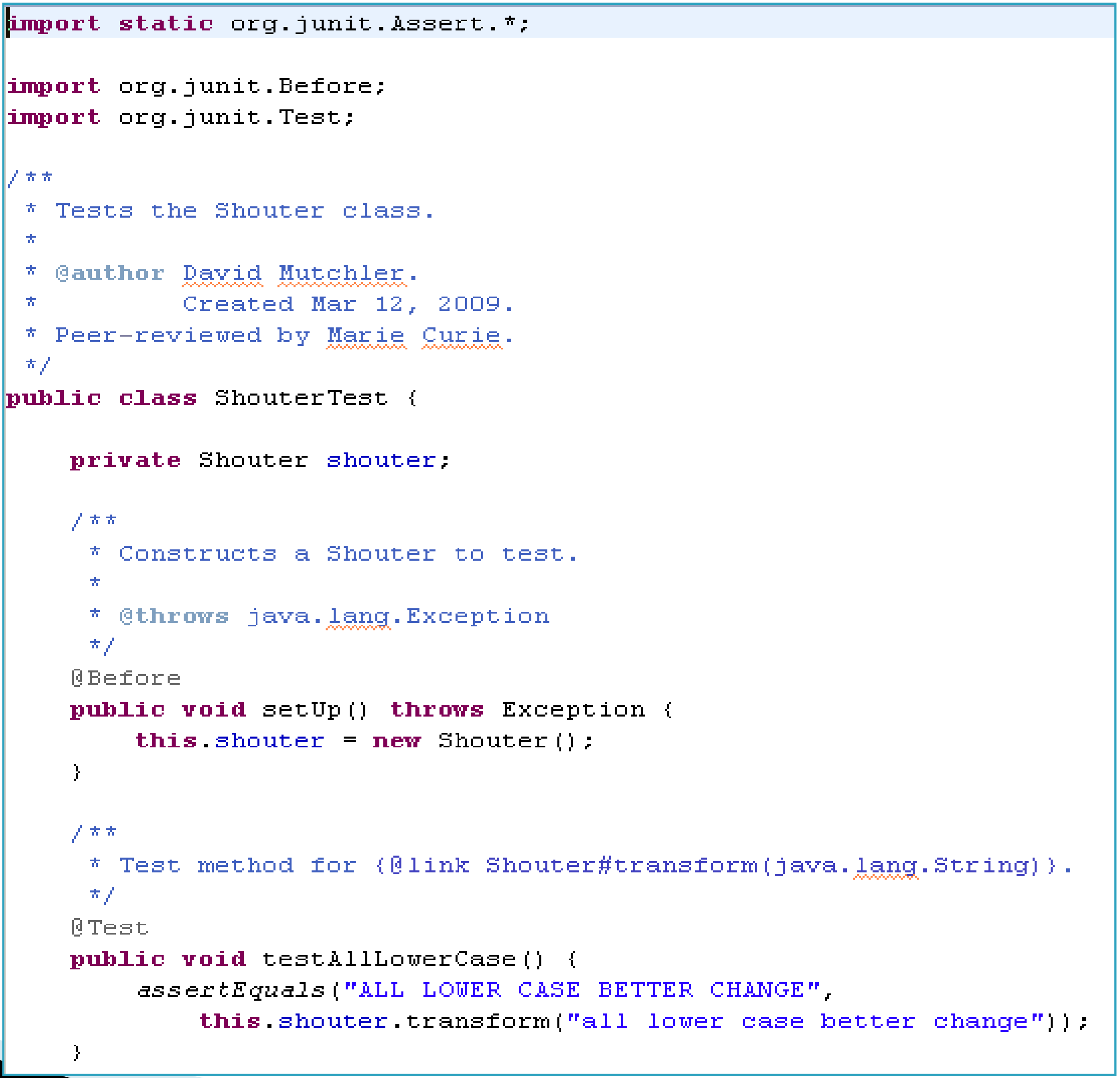
Shouter

Step 2: Write JUnit tests for the constructors and methods of Shouter

Do you see why:

- We test only *translate* here?
- There is a field for a Shouter object?
- How *setup* initializes that field?
- How *assertEquals* works?

Test cases continue on the next slides.



```java
import static org.junit.Assert.*;

import org.junit.Before;
import org.junit.Test;

/**
 * Tests the Shouter class.
 *
 * @author David Mutchler.
 *         Created Mar 12, 2009.
 * Peer-reviewed by Marie Curie.
 */
public class ShouterTest {

    private Shouter shouter;

    /**
     * Constructs a Shouter to test.
     *
     * @throws java.lang.Exception
     */
    @Before
    public void setUp() throws Exception {
        this.shouter = new Shouter();
    }

    /**
     * Test method for {@link Shouter#transform(java.lang.String)}.
     */
    @Test
    public void testAllLowerCase() {
        assertEquals("ALL LOWER CASE BETTER CHANGE",
            this.shouter.transform("all lower case better change"));
    }
```

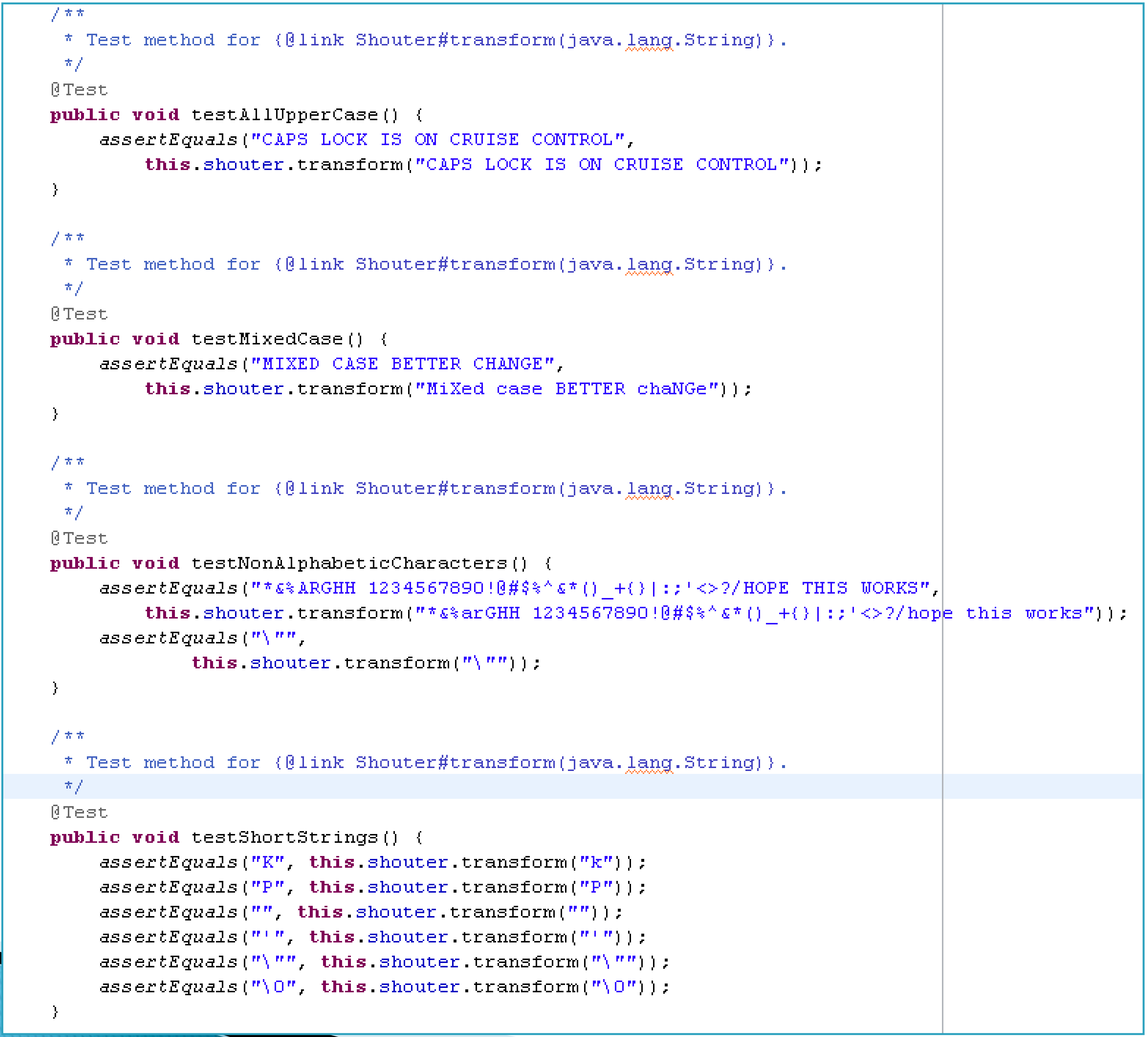
Shouter

Step 2: Write JUnit tests for the constructors and methods of Shouter

Do you see why:

- These are good test cases?

One last test case – testing long Strings – is on the next slide.



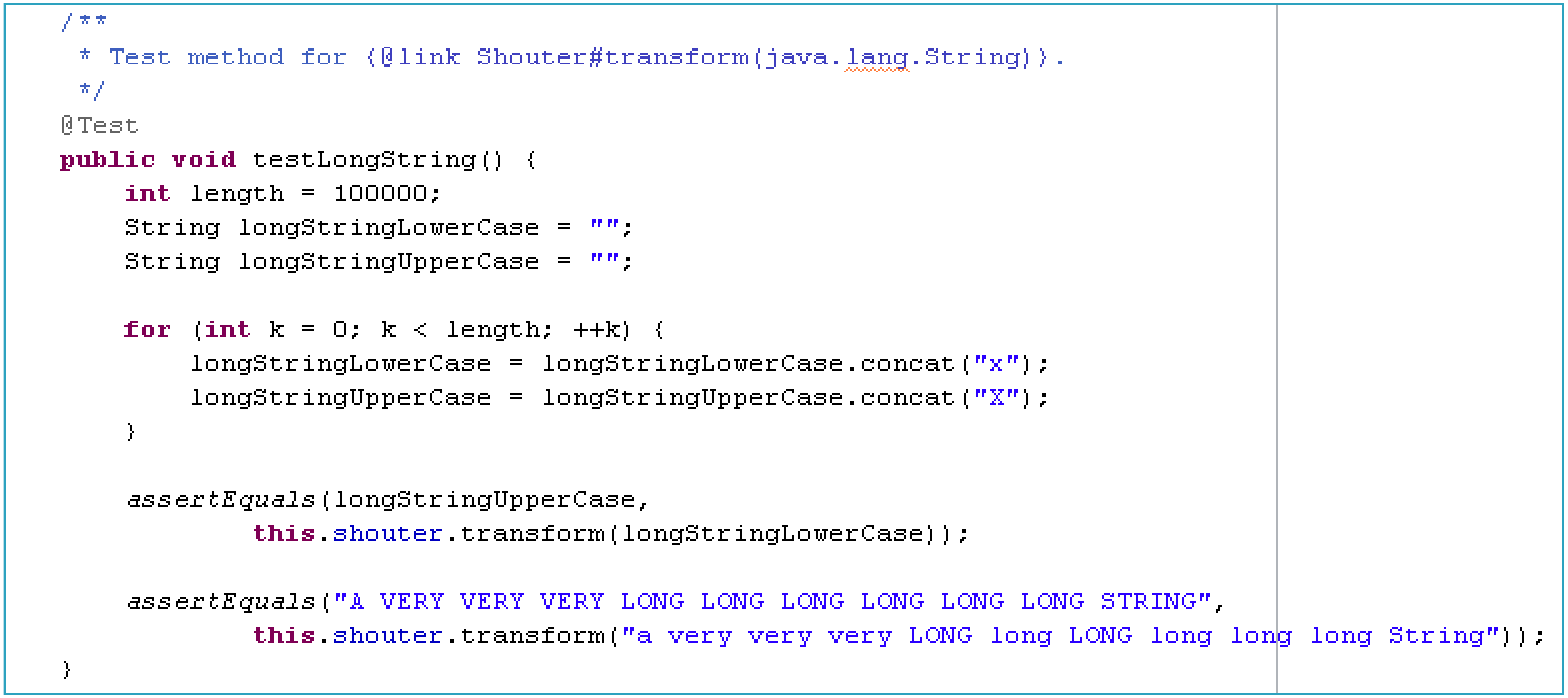
```java
/**
 * Test method for {@link Shouter#transform(java.lang.String)}.
 */
@Test
public void testAllUpperCase() {
    assertEquals("CAPS LOCK IS ON CRUISE CONTROL",
        this.shouter.transform("CAPS LOCK IS ON CRUISE CONTROL"));
}

/**
 * Test method for {@link Shouter#transform(java.lang.String)}.
 */
@Test
public void testMixedCase() {
    assertEquals("MIXED CASE BETTER CHANGE",
        this.shouter.transform("MiXed case BETTER chaNGe"));
}

/**
 * Test method for {@link Shouter#transform(java.lang.String)}.
 */
@Test
public void testNonAlphabeticCharacters() {
    assertEquals("*&%ARGHH 1234567890!@#$%^&*()_+{}|:;'<>?/HOPE THIS WORKS",
        this.shouter.transform("*&%arGHH 1234567890!@#$%^&*()_+{}|:;'<>?/hope this works"));
    assertEquals("\"",
            this.shouter.transform("\""));
}

/**
 * Test method for {@link Shouter#transform(java.lang.String)}.
 */
@Test
public void testShortStrings() {
    assertEquals("K", this.shouter.transform("k"));
    assertEquals("P", this.shouter.transform("P"));
    assertEquals("", this.shouter.transform(""));
    assertEquals("'", this.shouter.transform("'"));
    assertEquals("\"", this.shouter.transform("\""));
    assertEquals("\0", this.shouter.transform("\0"));
}
```

Shouter



```java
/**
 * Test method for {@link Shouter#transform(java.lang.String)}.
 */
@Test
public void testLongString() {
    int length = 100000;
    String longStringLowerCase = "";
    String longStringUpperCase = "";

    for (int k = 0; k < length; ++k) {
        longStringLowerCase = longStringLowerCase.concat("x");
        longStringUpperCase = longStringUpperCase.concat("X");
    }

    assertEquals(longStringUpperCase,
            this.shouter.transform(longStringLowerCase));

    assertEquals("A VERY VERY VERY LONG LONG LONG LONG LONG LONG STRING",
            this.shouter.transform("a very very very LONG long LONG long long long String"));
}
```

Step 2: Write JUnit tests for the constructors and methods of Shouter

Do you see why:

- How this uses *concat* and assignment to build long Strings?
- We should perhaps test *how long* the method takes?
  JUnit can do that, but we won't take the time to do so today.

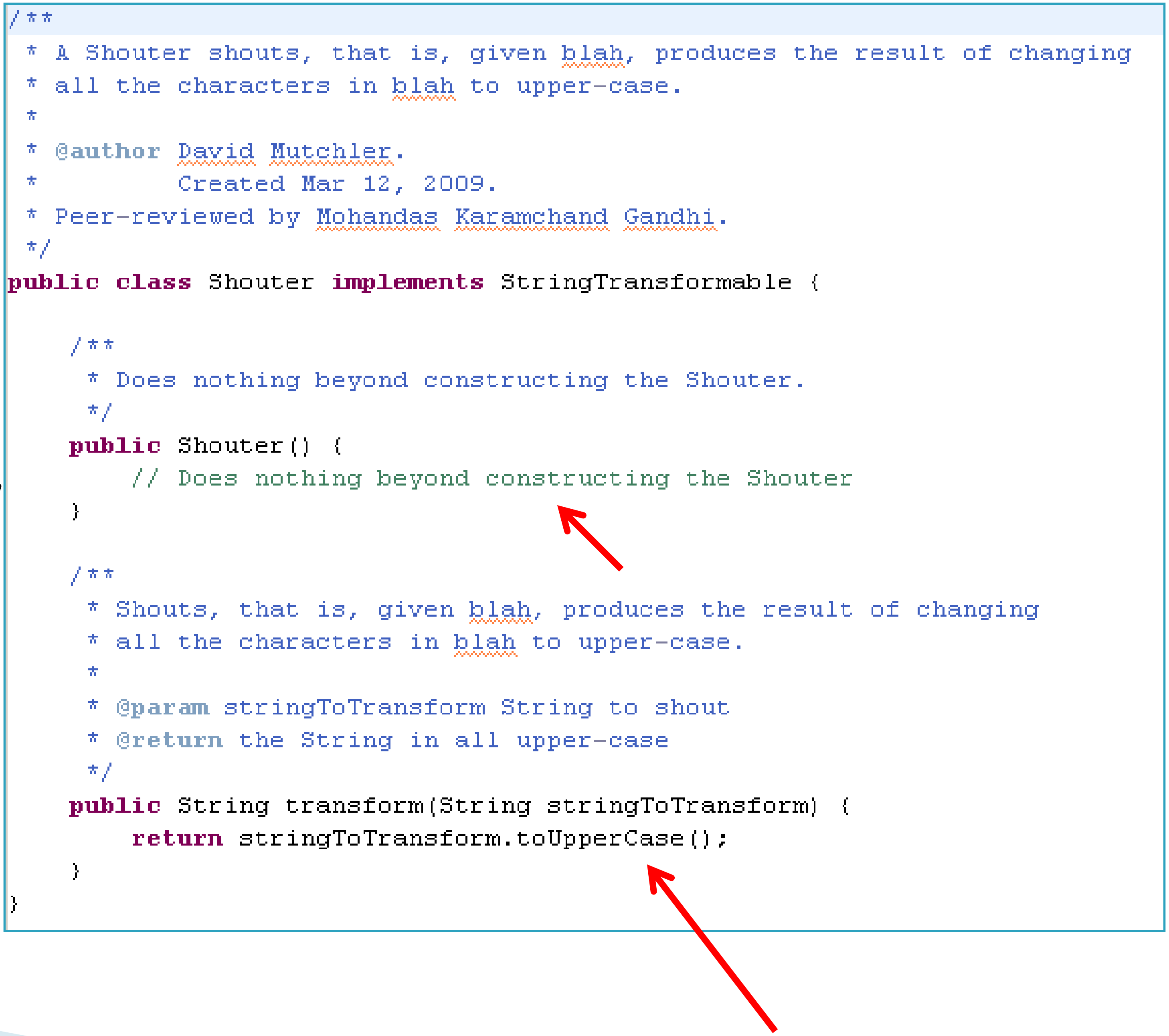
Questions on JUnit tests?

Shouter

Step 3: Implement the class, then test and debug code and/or tests.

Do you see:

- Why implementing *transform*:
  - starts with *stringToTransform*?
  - Continues with a dot?
- How you found the right method (*toUpperCase*) to call?
- Why you don't need a local variable in *transform*?
- Why the constructor does nothing?
- Why you should have the comment that it does nothing?



```java
/**
 * A Shouter shouts, that is, given blah, produces the result of changing
 * all the characters in blah to upper-case.
 *
 * @author David Mutchler.
 *          Created Mar 12, 2009.
 * Peer-reviewed by Mohandas Karamchand Gandhi.
 */
public class Shouter implements StringTransformable {

    /**
     * Does nothing beyond constructing the Shouter.
     */
    public Shouter() {
        // Does nothing beyond constructing the Shouter
    }

    /**
     * Shouts, that is, given blah, produces the result of changing
     * all the characters in blah to upper-case.
     *
     * @param stringToTransform String to shout
     * @return the String in all upper-case
     */
    public String transform(String stringToTransform) {
        return stringToTransform.toUpperCase();
    }
}
```

Fields

- What are *fields* (aka *member variables*) of a class?

- When does a class need fields?

  - Example: TestShouter needs a field.
    - What is the field?
    - Why is it needed?
    - How is it initialized?

  - Example: Censor needs a field.
    - What is the field?
    - Why is it needed?
    - How is it initialized?