Object Oriented Programming

Week 8 Part 2
Developing Console Input and Output

Lecture

- Input Output Design
- Unit Testing I/O

Adding input and output to a program

Adding Input and Output to a Program

- The last discussion describes console I/O
- There is more to adding I/O to an object oriented program
  - How do you test I/O?
  - How can you connect I/O to appropriate streams?
  - Which class is responsible for I/O?
  - Etc.

Animals yet again

- How do we add I/O for wolves and wolf packs.
  - Wolf is an object with fields
  - Wolf Pack is an ListArray<Wolf>
- We will do this in two steps:
  1) We will produce and InputOutput class that prints and read Wolf and Pack.
  2) We will refactor the program so that Wolf and Pack can print and read themselves.

Step 1: Add Input/Output Object

- We want to create an object that will handle the input and output for us.
- We want to be able to test this object using JUnit.
  - We can see the output, and could output the input to see it, be we want to automate the tests
  - With automation we test quickly giving so new code doesn't break old code

Testing I/O

- The key to testing I/O is replacing the streams associated with the monitor or the keyboard, with streams associated with strings.
  - If you print to a string stream, it produces a string with everything printed to it.
  - If you read from a string stream, it produces characters from a string with which it is initialized

Putting Output into a String

String Output

- System.In is a PrintStream
- A PrintStream can be constructed from a OutputStream
  - Constructor: PrintStream(OutputStream out)
- A ByteStreamOutputStream, that puts the output into a string
  - Constructor: ByteStreamOutputString()
  - To retrieve output string: toString()

String Output Calls

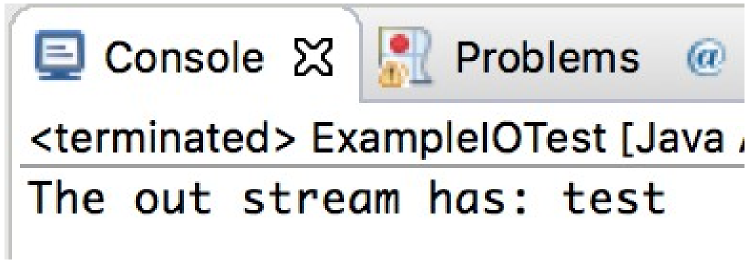
- Create a ByteArrayOutputStream
  - ByteArrayOutputStream outStream =

    new ByteArrayOutputStream();
- Create a PrintStream from the StringWriter
  - PrintStream out = new PrintStream(outStream);
- Print “test” to a string
  - out.print(“test”);
- Test the string from the stream
  - assertEquals(“test”, outStream.toString);

Example: String Output

```
package oop.example;

import java.io.ByteArrayOutputStream;
import java.io.PrintStream;

public class ExampleIOTest {

    public static void main(String[] args) {
        ByteArrayOutputStream outStream = new ByteArrayOutputStream();
        PrintStream out = new PrintStream(outStream);
        out.print("test");
        System.out.println("The out stream has: " + outStream.toString());
    }
}
```

Output



```
<terminated> ExampleIOTest [Java
The out stream has: test
```

Getting Input from a String

String Input Calls

- Create a Buffered Reader from a StringReader
  - BufferedReader in =

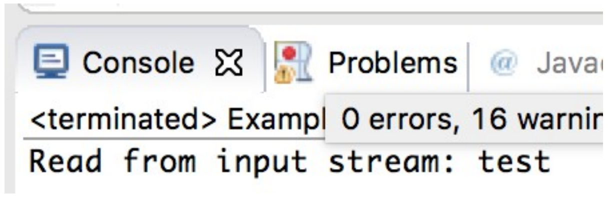
                new BufferedReader (

                          new StringReader (“test\n”));
- Reader from the BufferedReader
  - String s = in.readLine()
- Test the string read
  - assertEquals(“test”, s);

Example: String Input

```java
package oop.example;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.StringReader;

public class ExampleIOTest {

    public static void main(String[] args) {
        BufferedReader in = new BufferedReader(new StringReader("test\n"));
        String s = "uninitialized";
        try {
            s = in.readLine();
        } catch (IOException e) {
            e.printStackTrace();
        }
        System.out.println("Read from input stream: " + s);
    }
}
```

Output



```
<terminated> Exampl 0 errors, 16 warnir
Read from input stream: test
```

Adding I/O to Animals

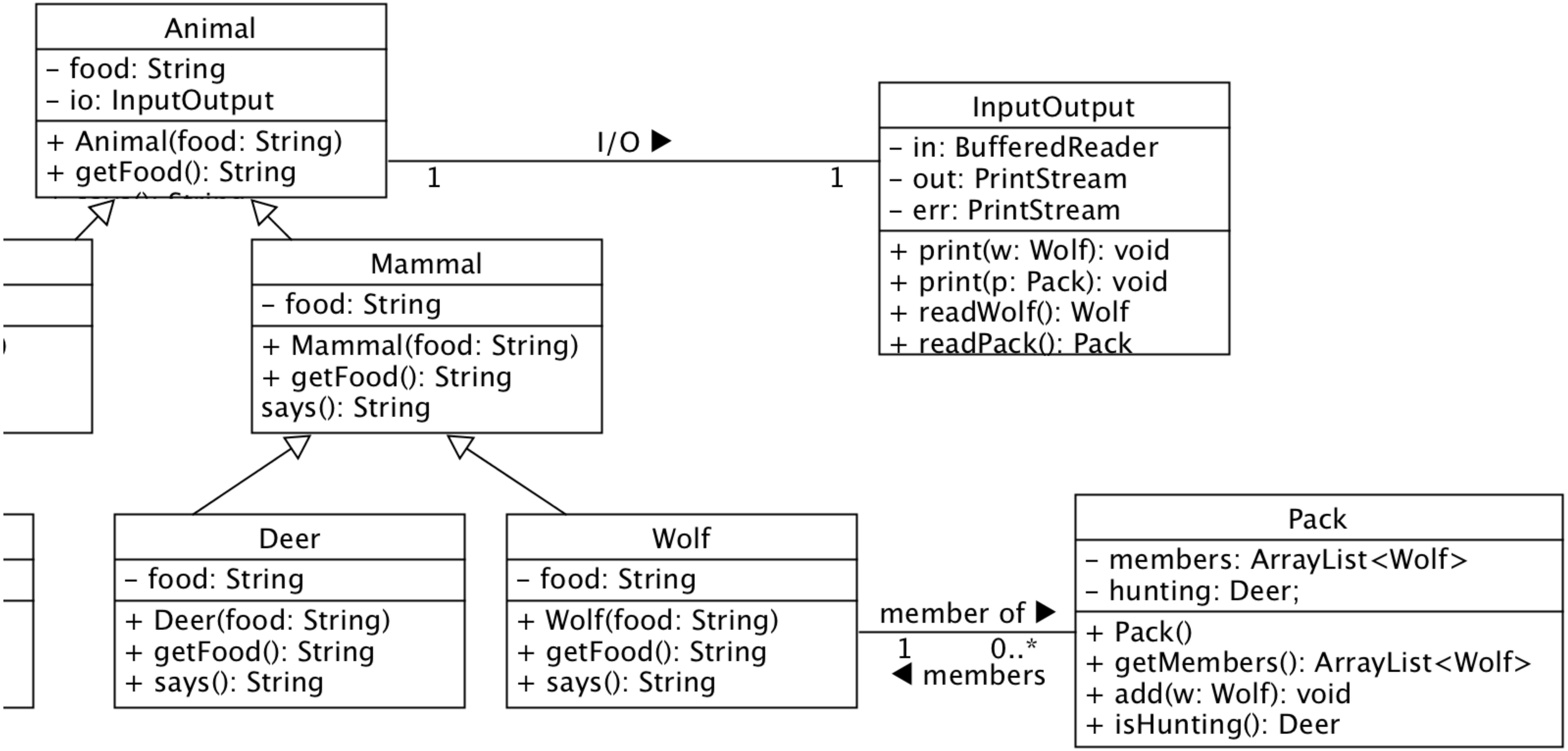
Adding I/O to a Program

- We want to encapsulate program I/O to
  - Allow testing by substituting a fake input or output stream for the real input or output stream
  - Redirect I/O for different UIs
    - For example, in a GUI we may want to print to a window in the program rather than to the console

Adding I/O to Animals

- We will add an I/O object to animals
  - It will contain three streams a fields: in, out, and err
  - It will define the print commands for Wolf and Pack
  - It will define the read commands for Wolf and Pack

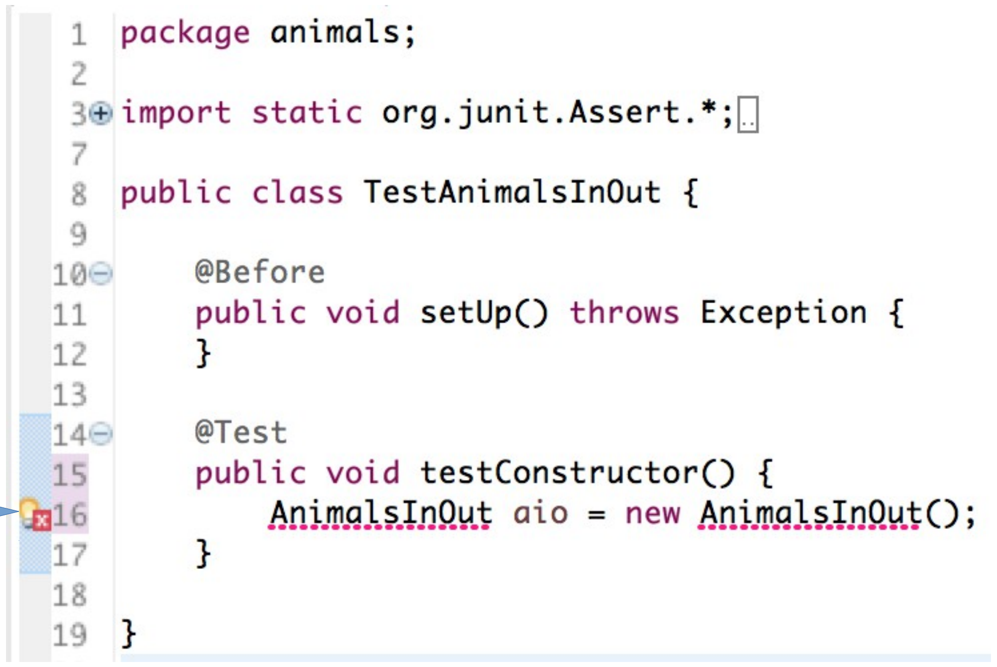
Partial UML for I/O for Animals



Writing the InputOutput class

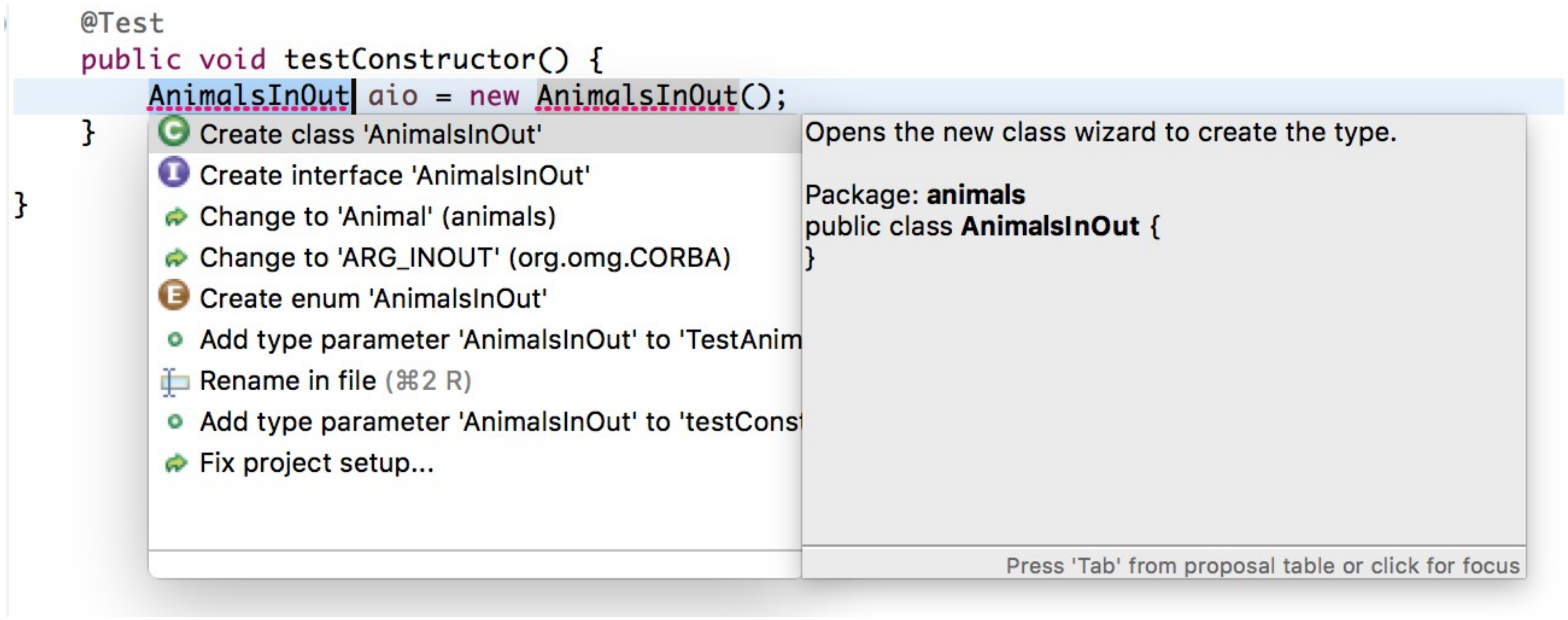
Create a Test

Error: AnimalInOut not defined



```
package animals;

import static org.junit.Assert.*;

public class TestAnimalsInOut {

    @Before
    public void setUp() throws Exception {
    }

    @Test
    public void testConstructor() {
        AnimalsInOut aio = new AnimalsInOut();
    }

}
```

Suggestions: Select Create Class



```
    @Test
    public void testConstructor() {
        AnimalsInOut aio = new AnimalsInOut();
    }

}
```

Create class 'AnimalsInOut'
Create interface 'AnimalsInOut'
Change to 'Animal' (animals)
Change to 'ARG_INOUT' (org.omg.CORBA)
Create enum 'AnimalsInOut'
Add type parameter 'AnimalsInOut' to 'TestAnim
Rename in file (⌘2 R)
Add type parameter 'AnimalsInOut' to 'testCons
Fix project setup...

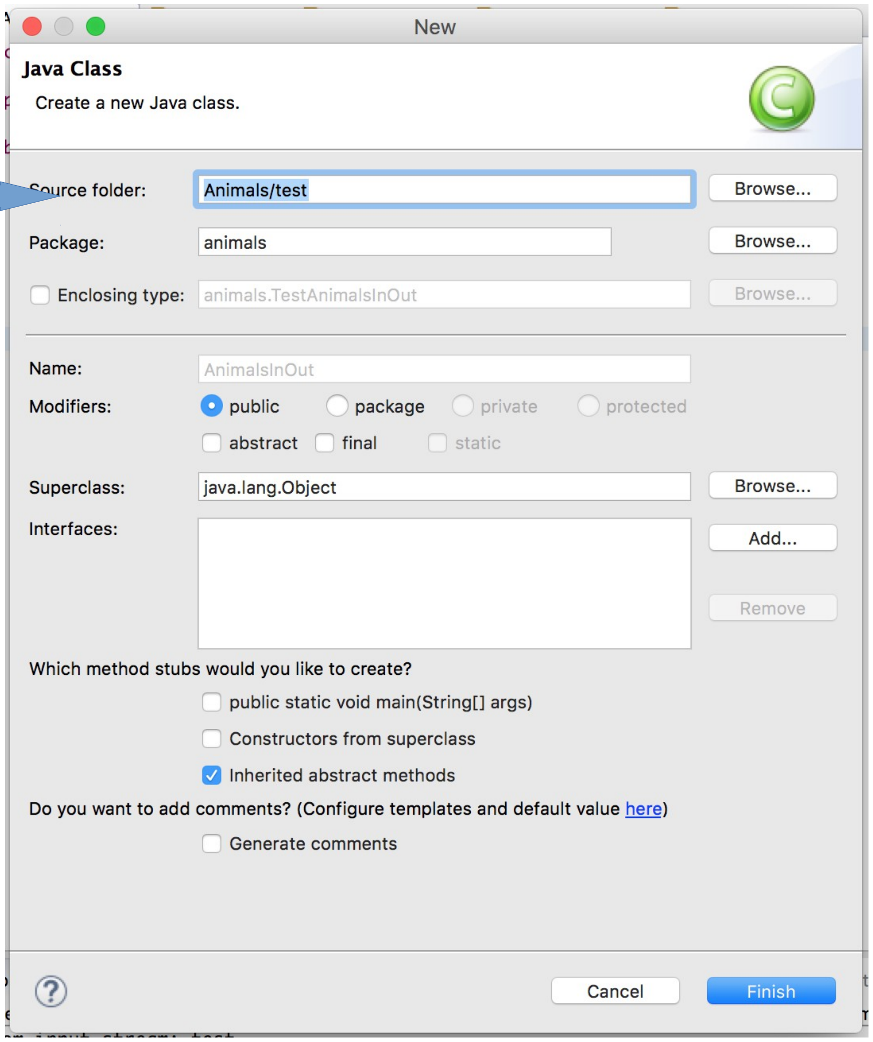
Opens the new class wizard to create the type.

Package: **animals**
public class **AnimalsInOut** {
}

Press 'Tab' from proposal table or click for focus

Create Class in Animals/src

Change Source Folder to Animals/src



New

Java Class

Create a new Java class.

Source folder: Animals/test

Package: animals

[ ] Enclosing type: animals.TestAnimalsInOut

Name: AnimalsInOut

Modifiers: [✓ public] package private protected

[ ] abstract [ ] final [ ] static

Superclass: java.lang.Object

Interfaces:

Which method stubs would you like to create?

[ ] public static void main(String[] args)

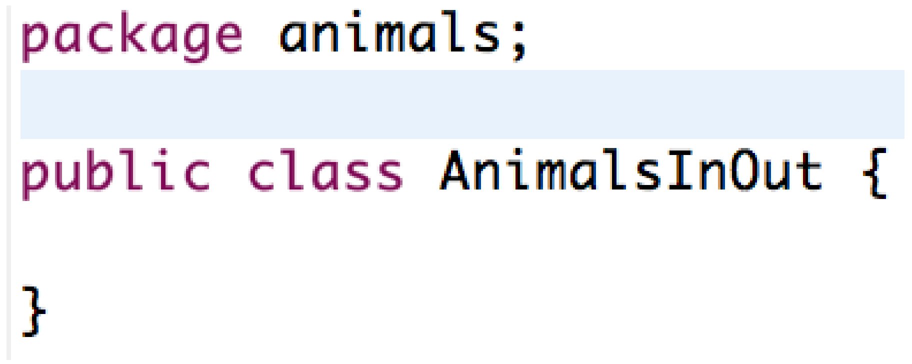
[ ] Constructors from superclass

[✓] Inherited abstract methods

Do you want to add comments? (Configure templates and default value here)

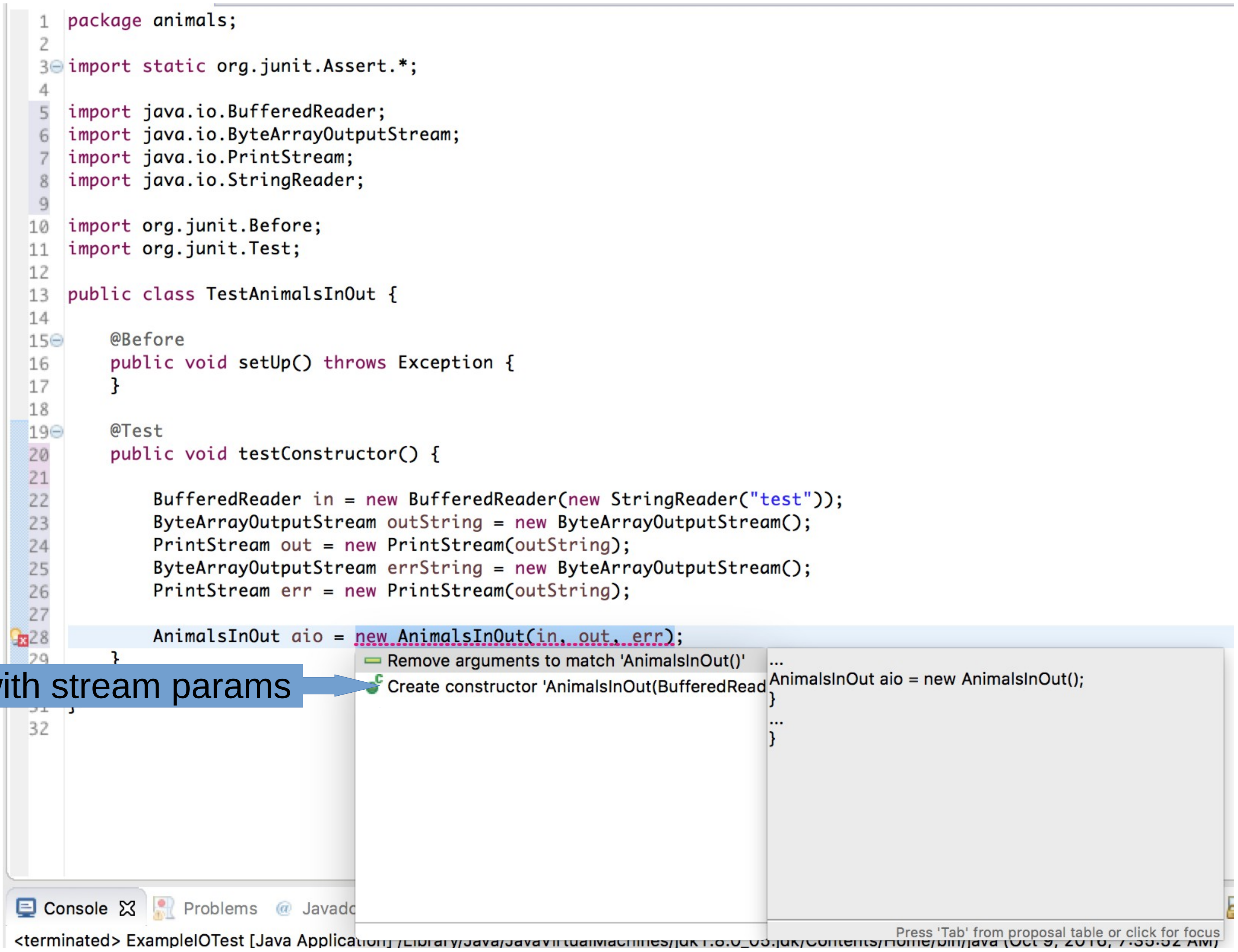
[ ] Generate comments

Creates Template Class



```
package animals;

public class AnimalsInOut {

}
```

Create a new Constructor



```
package animals;

import static org.junit.Assert.*;

import java.io.BufferedReader;
import java.io.ByteArrayOutputStream;
import java.io.PrintStream;
import java.io.StringReader;

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

public class TestAnimalsInOut {

    @Before
    public void setUp() throws Exception {
    }

    @Test
    public void testConstructor() {

        BufferedReader in = new BufferedReader(new StringReader("test"));
        ByteArrayOutputStream outString = new ByteArrayOutputStream();
        PrintStream out = new PrintStream(outString);
        ByteArrayOutputStream errString = new ByteArrayOutputStream();
        PrintStream err = new PrintStream(outString);

        AnimalsInOut aio = new AnimalsInOut(in, out, err);
    }
```

Remove arguments to match 'AnimalsInOut()'
Create constructor 'AnimalsInOut(BufferedRead
Console
Problems
Javadc
Press 'Tab' from proposal table or click for focus

```
...
AnimalsInOut aio = new AnimalsInOut();
}
...
}
```

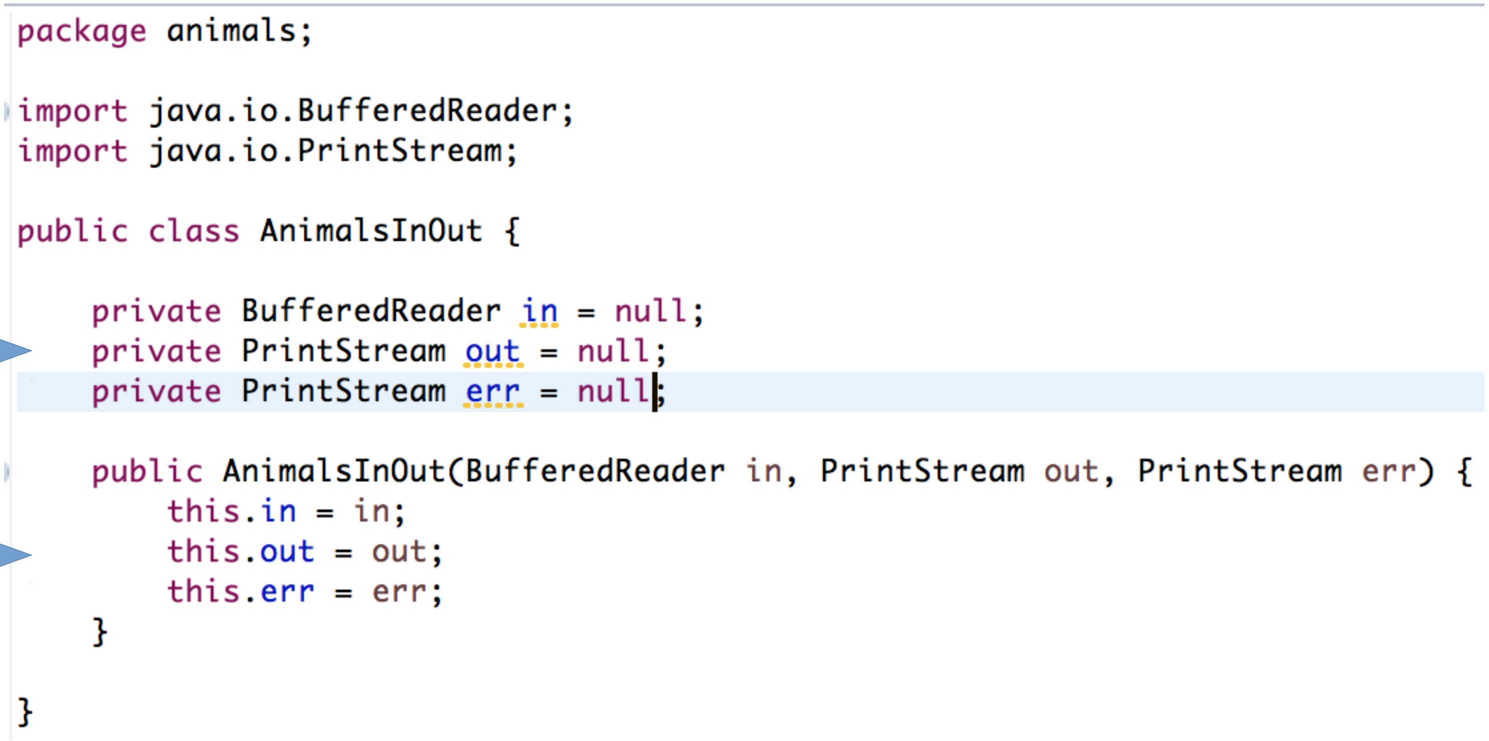
<terminated> ExampleIOTest [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_05.jdk/Contents/Home/bin/java (Oct 9, 2016, 7:35:32 AM)

New Constructor with stream params

Fill in new constructor

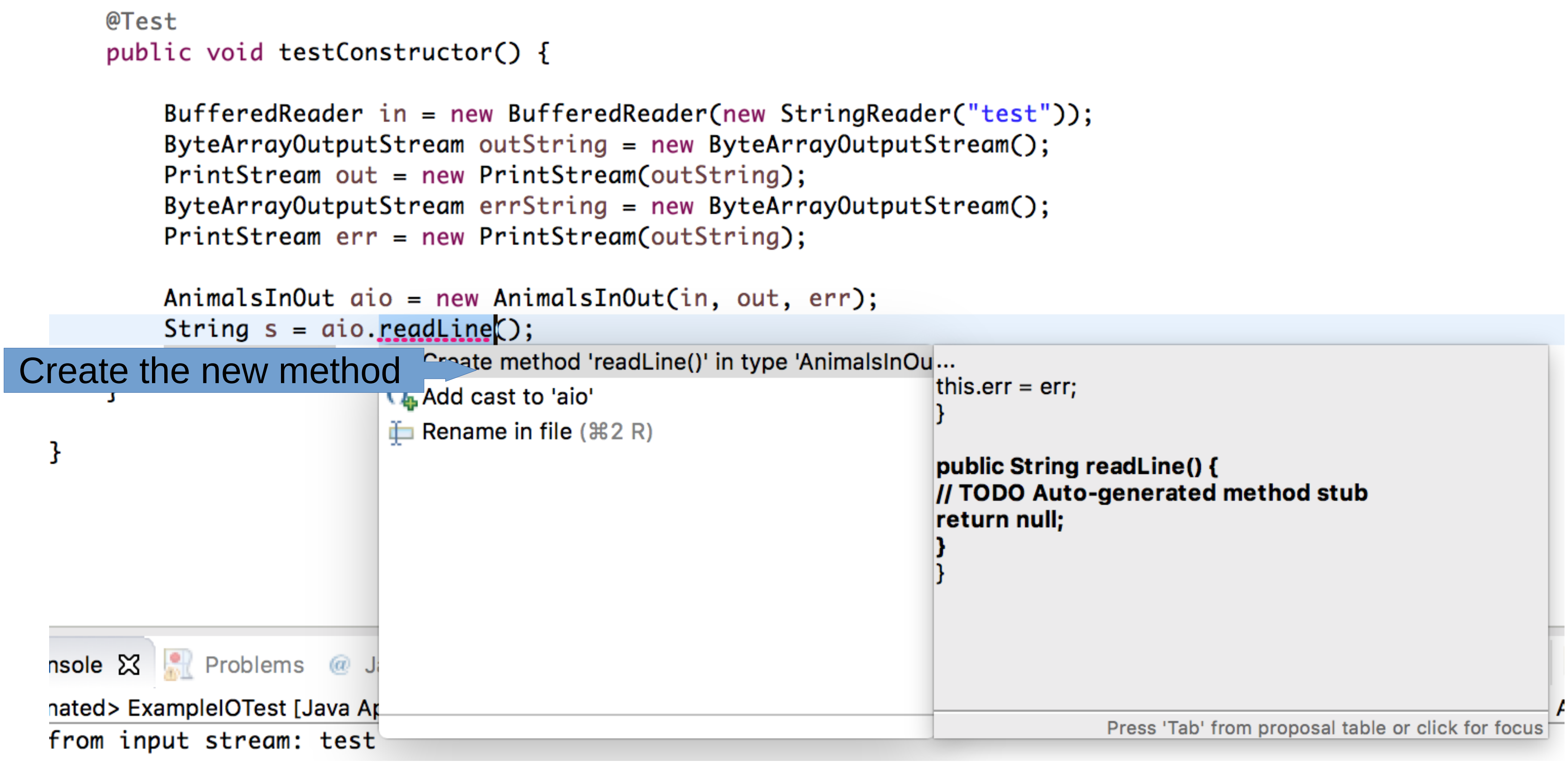
Stream fields

Initialize stream fields



```java
package animals;

import java.io.BufferedReader;
import java.io.PrintStream;

public class AnimalsInOut {

    private BufferedReader in = null;
    private PrintStream out = null;
    private PrintStream err = null;

    public AnimalsInOut(BufferedReader in, PrintStream out, PrintStream err) {
        this.in = in;
        this.out = out;
        this.err = err;
    }

}
```

Add aio.readline test



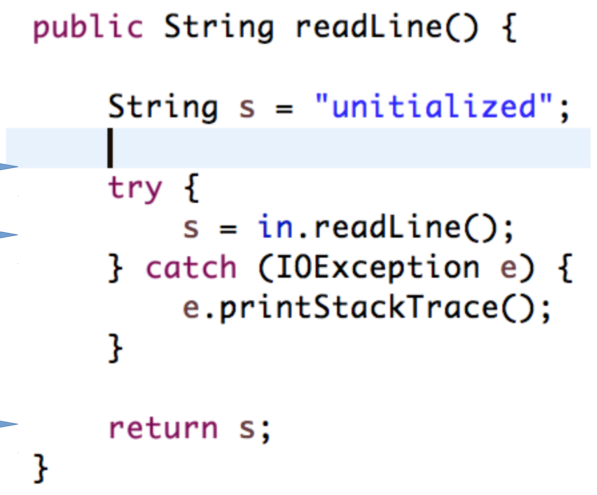
```
@Test
public void testConstructor() {

    BufferedReader in = new BufferedReader(new StringReader("test"));
    ByteArrayOutputStream outString = new ByteArrayOutputStream();
    PrintStream out = new PrintStream(outString);
    ByteArrayOutputStream errString = new ByteArrayOutputStream();
    PrintStream err = new PrintStream(outString);

    AnimalsInOut aio = new AnimalsInOut(in, out, err);
    String s = aio.readLine();
}

}
```

Create the new method

```
this.err = err;
}

public String readLine() {
// TODO Auto-generated method stub
return null;
}
}
```

```
nated> ExampleIOTest [Java Ap
from input stream: test
```

Write new method



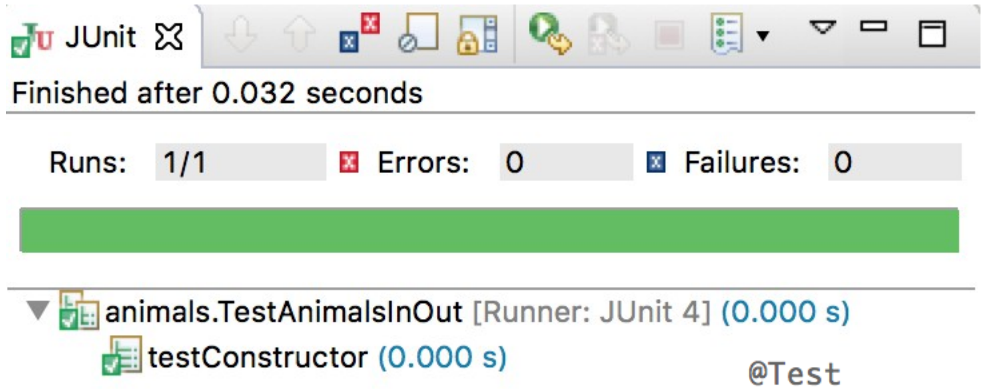
```
public String readLine() {

    String s = "unitialized";

    try {
        s = in.readLine();
    } catch (IOException e) {
        e.printStackTrace();
    }

    return s;
}
```

The readLine() method throws an exception

Read a string from the in stream

Return string read

Run the test

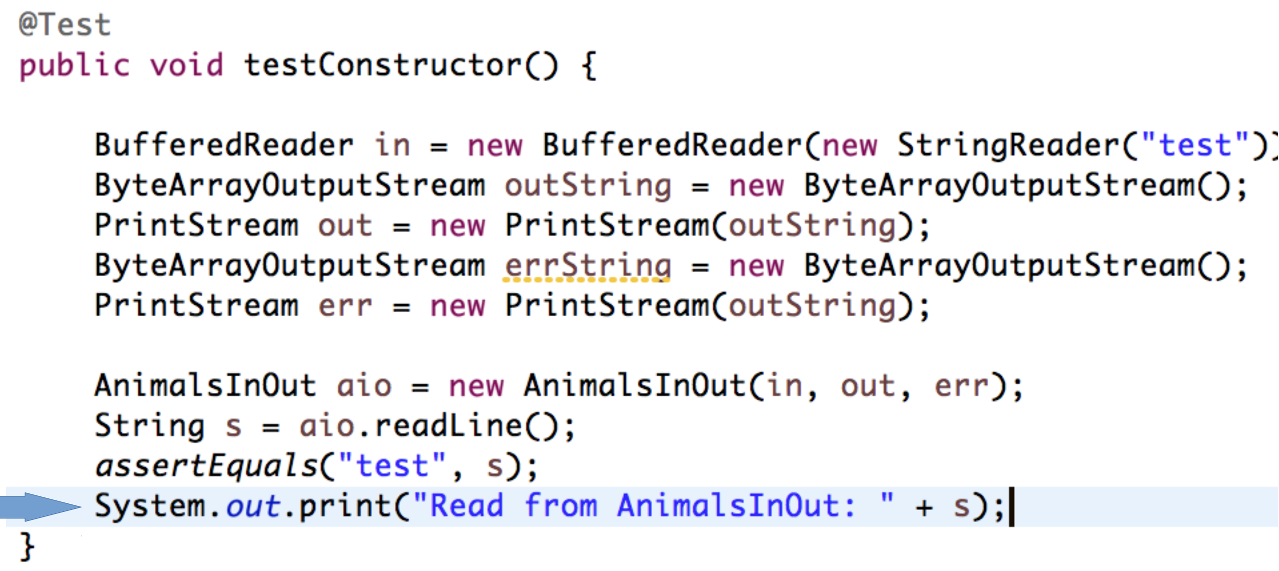

JUnit

Finished after 0.032 seconds

Runs: 1/1 Errors: 0 Failures: 0

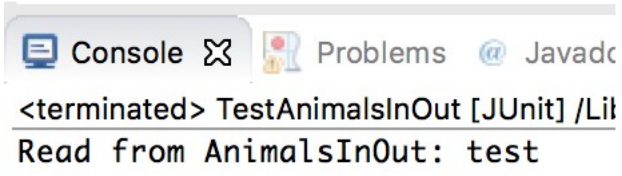
animals.TestAnimalsInOut [Runner: JUnit 4] (0.000 s)

testConstructor (0.000 s)



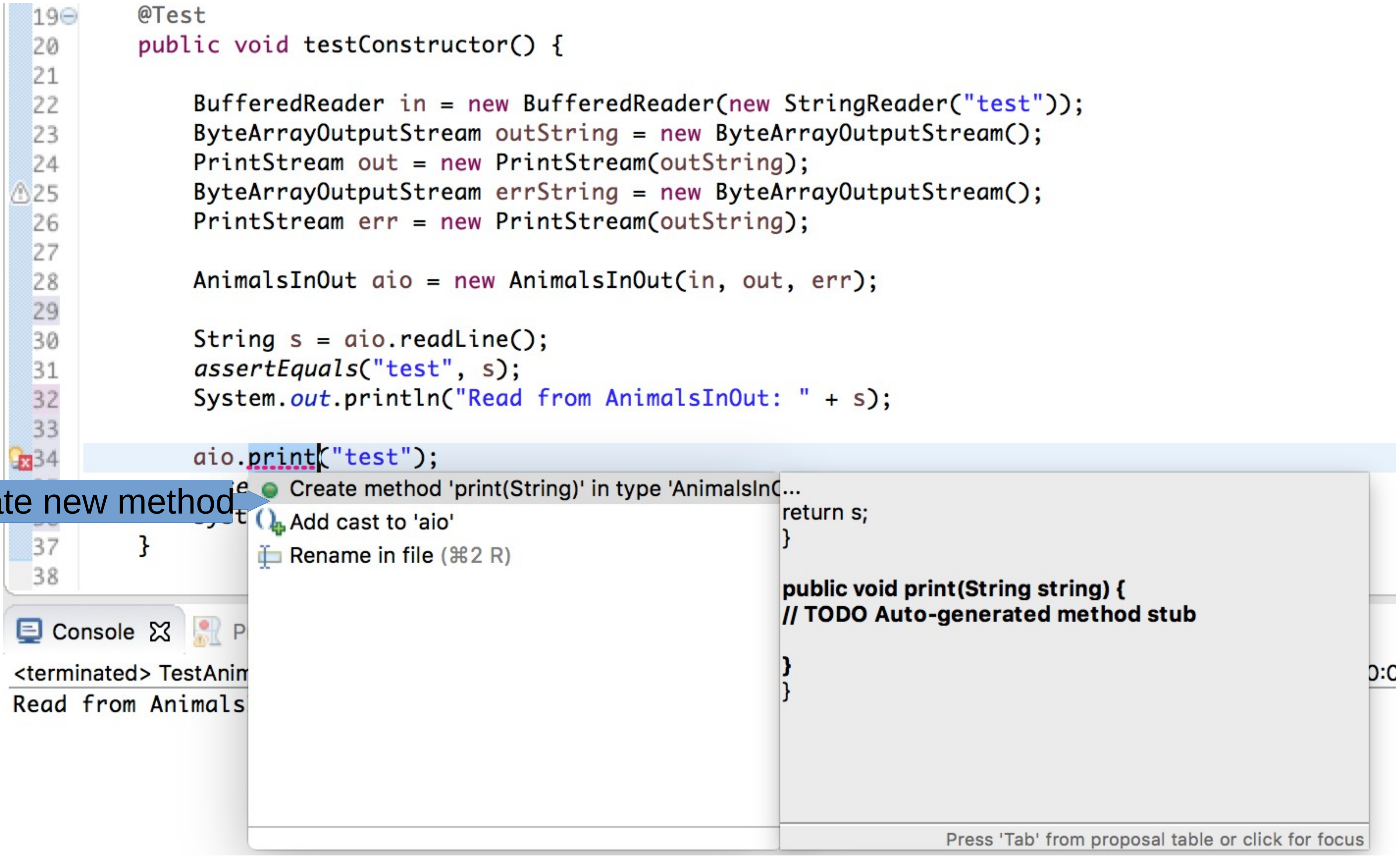
```
@Test
public void testConstructor() {

    BufferedReader in = new BufferedReader(new StringReader("test"))
    ByteArrayOutputStream outString = new ByteArrayOutputStream();
    PrintStream out = new PrintStream(outString);
    ByteArrayOutputStream errString = new ByteArrayOutputStream();
    PrintStream err = new PrintStream(outString);

    AnimalsInOut aio = new AnimalsInOut(in, out, err);
    String s = aio.readLine();
    assertEquals("test", s);
    System.out.print("Read from AnimalsInOut: " + s);
}
```

Add output to test


Console Problems Javadc

```
<terminated> TestAnimalsInOut [JUnit] /Lil
Read from AnimalsInOut: test
```

Extend the test



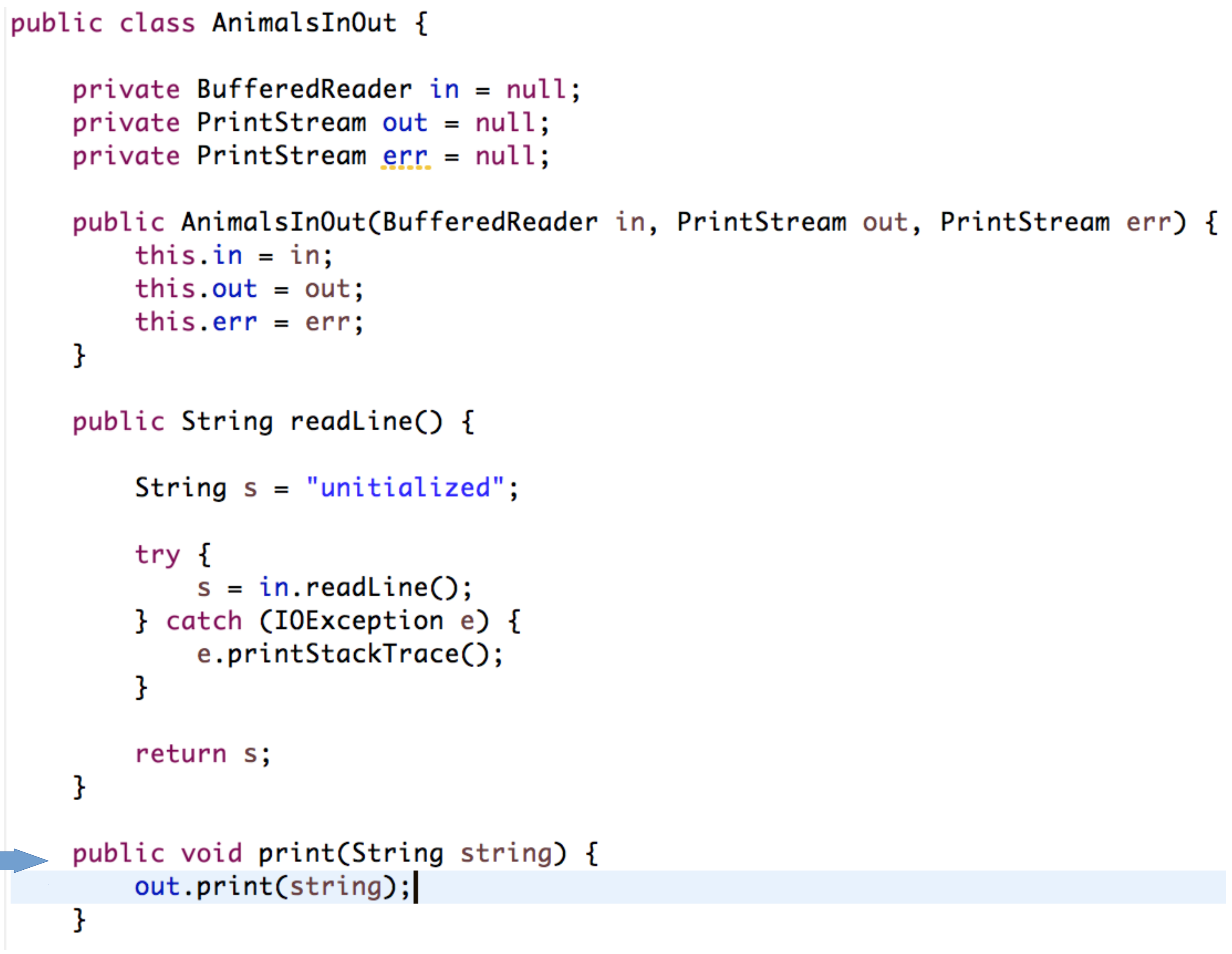
```
    @Test
    public void testConstructor() {

        BufferedReader in = new BufferedReader(new StringReader("test"));
        ByteArrayOutputStream outString = new ByteArrayOutputStream();
        PrintStream out = new PrintStream(outString);
        ByteArrayOutputStream errString = new ByteArrayOutputStream();
        PrintStream err = new PrintStream(outString);

        AnimalsInOut aio = new AnimalsInOut(in, out, err);

        String s = aio.readLine();
        assertEquals("test", s);
        System.out.println("Read from AnimalsInOut: " + s);

        aio.print("test");
```

```
    }
```

Create new method
Create method 'print(String)' in type 'AnimalsInO...
Add cast to 'aio'
Rename in file (⌘2 R)
Console
<terminated> TestAnim
Read from Animals
Press 'Tab' from proposal table or click for focus

```
return s;
}

public void print(String string) {
// TODO Auto-generated method stub

}
}
```

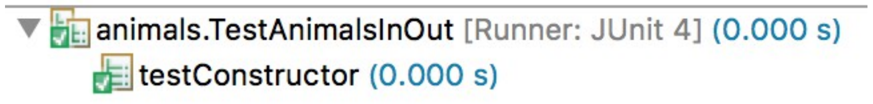
Write new method



```
public class AnimalsInOut {

    private BufferedReader in = null;
    private PrintStream out = null;
    private PrintStream err = null;

    public AnimalsInOut(BufferedReader in, PrintStream out, PrintStream err) {
        this.in = in;
        this.out = out;
        this.err = err;
    }

    public String readLine() {

        String s = "unitialized";

        try {
            s = in.readLine();
        } catch (IOException e) {
            e.printStackTrace();
        }

        return s;
    }

    public void print(String string) {
        out.print(string);
    }
```

New method

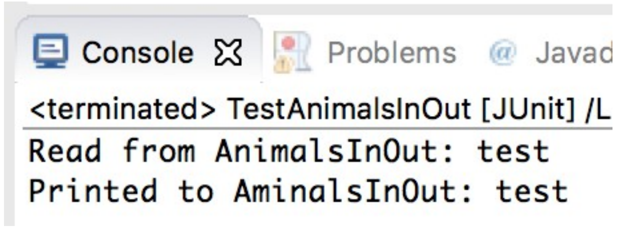
Run Extended Test



```
animals.TestAnimalsInOut [Runner: JUnit 4] (0.000 s)
    testConstructor (0.000 s)
```

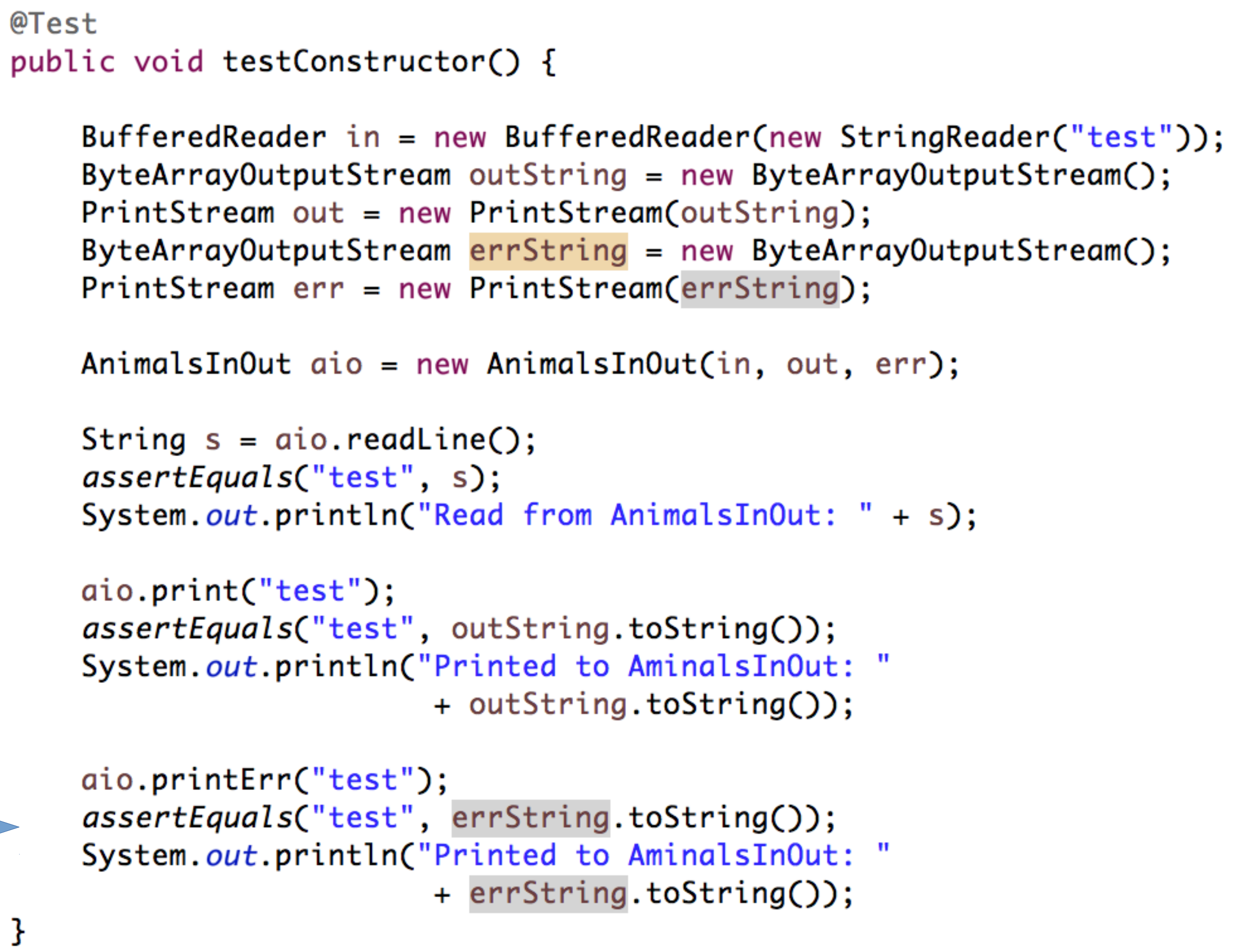
```
@Test
public void testConstructor() {

    BufferedReader in = new BufferedReader(new StringReader("test"));
    ByteArrayOutputStream outString = new ByteArrayOutputStream();
    PrintStream out = new PrintStream(outString);
    ByteArrayOutputStream errString = new ByteArrayOutputStream();
    PrintStream err = new PrintStream(outString);

    AnimalsInOut aio = new AnimalsInOut(in, out, err);

    String s = aio.readLine();
    assertEquals("test", s);
    System.out.println("Read from AnimalsInOut: " + s);

    aio.print("test");
    assertEquals("test", outString.toString());
    System.out.println("Printed to AminalsInOut: "
                      + outString.toString());
}
```

Print Output



```
<terminated> TestAnimalsInOut [JUnit] /L
Read from AnimalsInOut: test
Printed to AminalsInOut: test
```

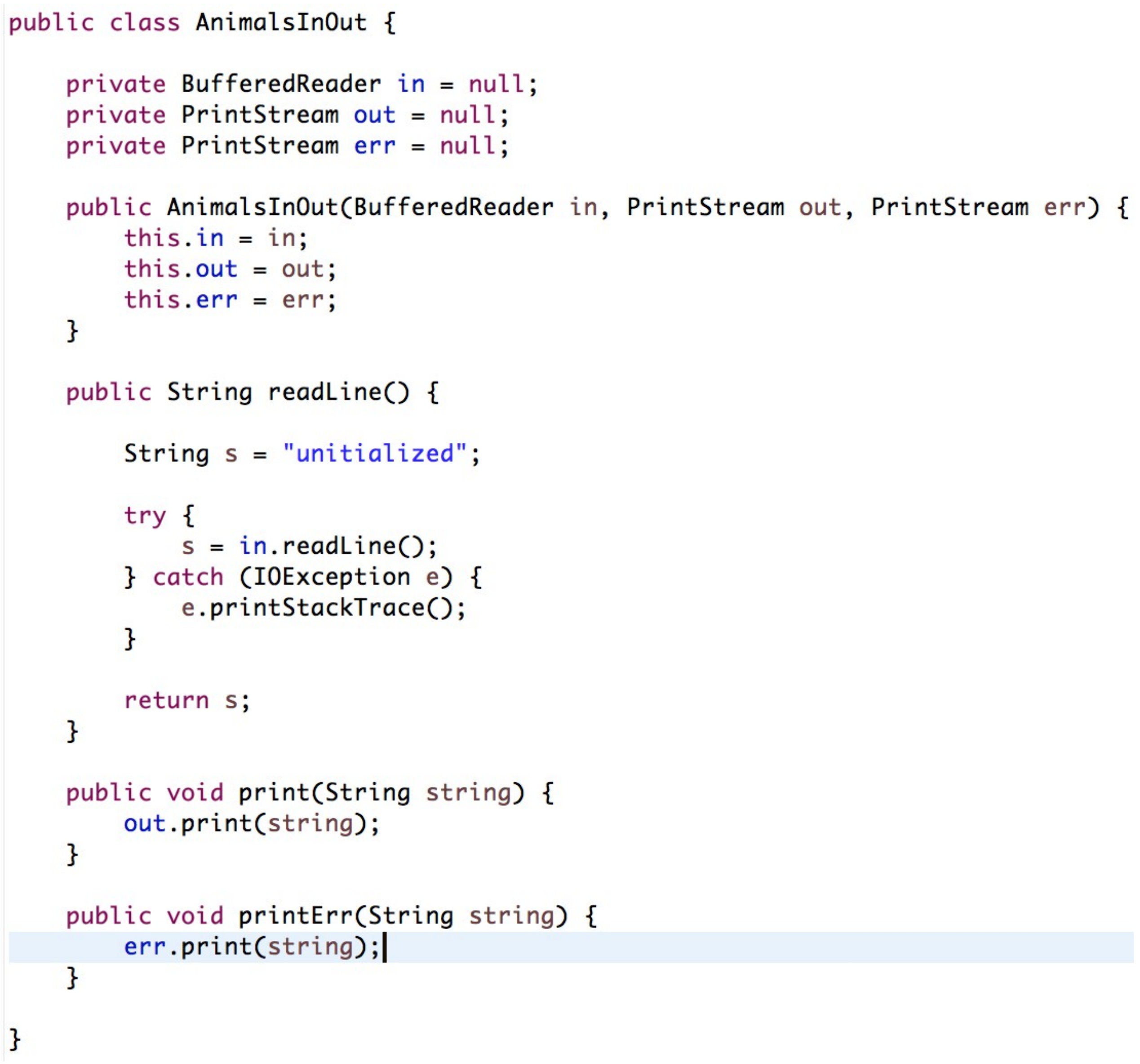
Test error stream



```
@Test
public void testConstructor() {

    BufferedReader in = new BufferedReader(new StringReader("test"));
    ByteArrayOutputStream outString = new ByteArrayOutputStream();
    PrintStream out = new PrintStream(outString);
    ByteArrayOutputStream errString = new ByteArrayOutputStream();
    PrintStream err = new PrintStream(errString);

    AnimalsInOut aio = new AnimalsInOut(in, out, err);

    String s = aio.readLine();
    assertEquals("test", s);
    System.out.println("Read from AnimalsInOut: " + s);

    aio.print("test");
    assertEquals("test", outString.toString());
    System.out.println("Printed to AminalsInOut: "
                         + outString.toString());

    aio.printErr("test");
    assertEquals("test", errString.toString());
    System.out.println("Printed to AminalsInOut: "
                         + errString.toString());
}
```

New Test

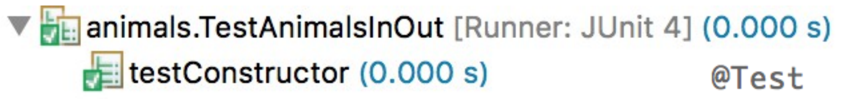
Add error stream method



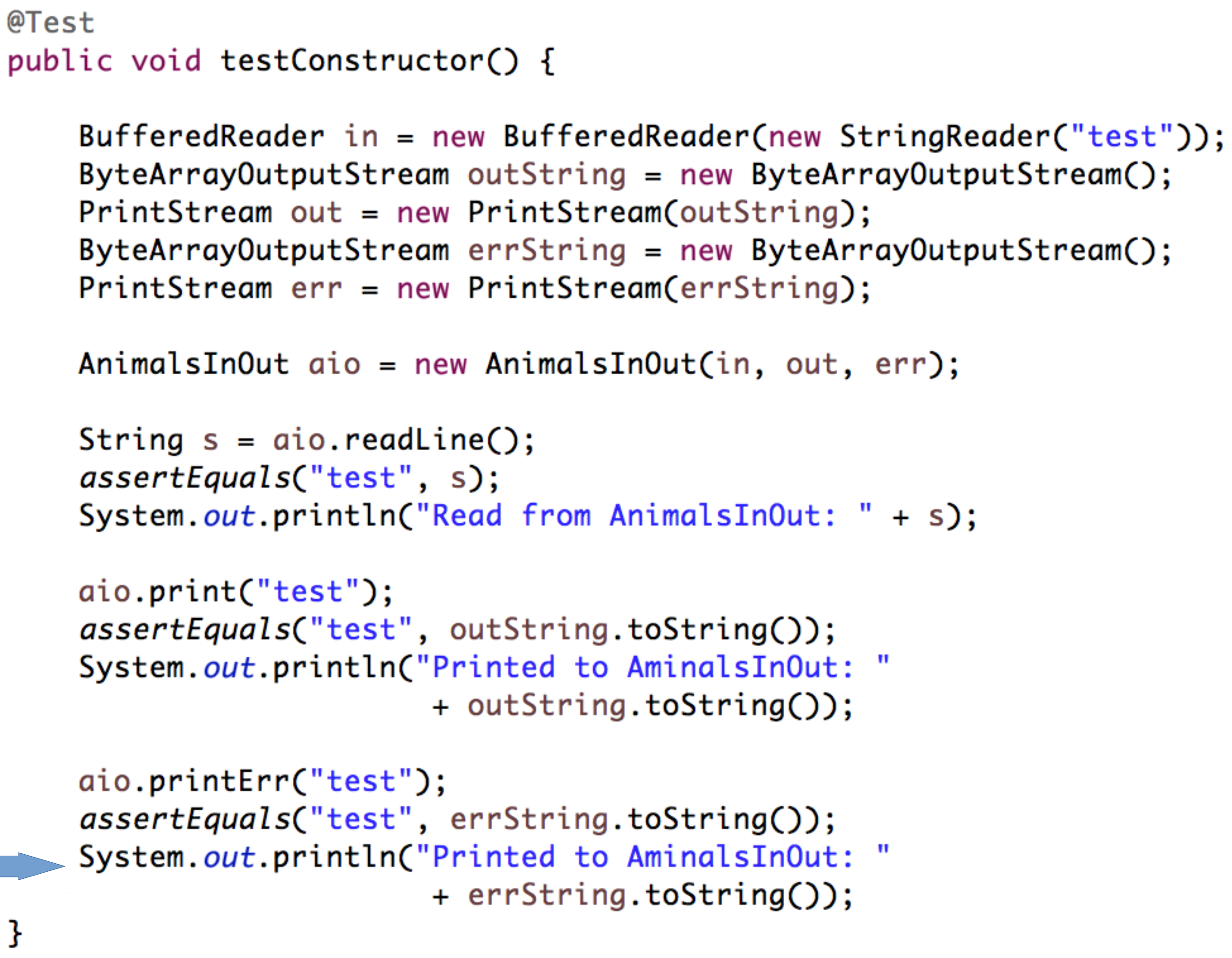
```
public class AnimalsInOut {

    private BufferedReader in = null;
    private PrintStream out = null;
    private PrintStream err = null;

    public AnimalsInOut(BufferedReader in, PrintStream out, PrintStream err) {
        this.in = in;
        this.out = out;
        this.err = err;
    }

    public String readLine() {

        String s = "unitialized";

        try {
            s = in.readLine();
        } catch (IOException e) {
            e.printStackTrace();
        }

        return s;
    }

    public void print(String string) {
        out.print(string);
    }

    public void printErr(String string) {
        err.print(string);
    }

}
```

New method

Run test again

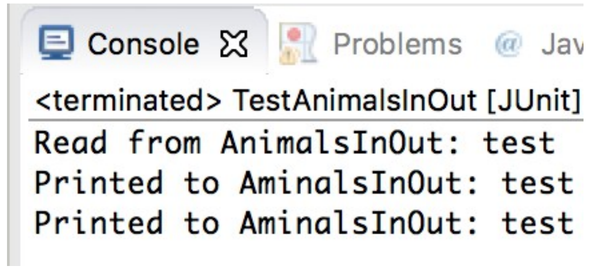


```
animals.TestAnimalsInOut [Runner: JUnit 4] (0.000 s)
    testConstructor (0.000 s)
```



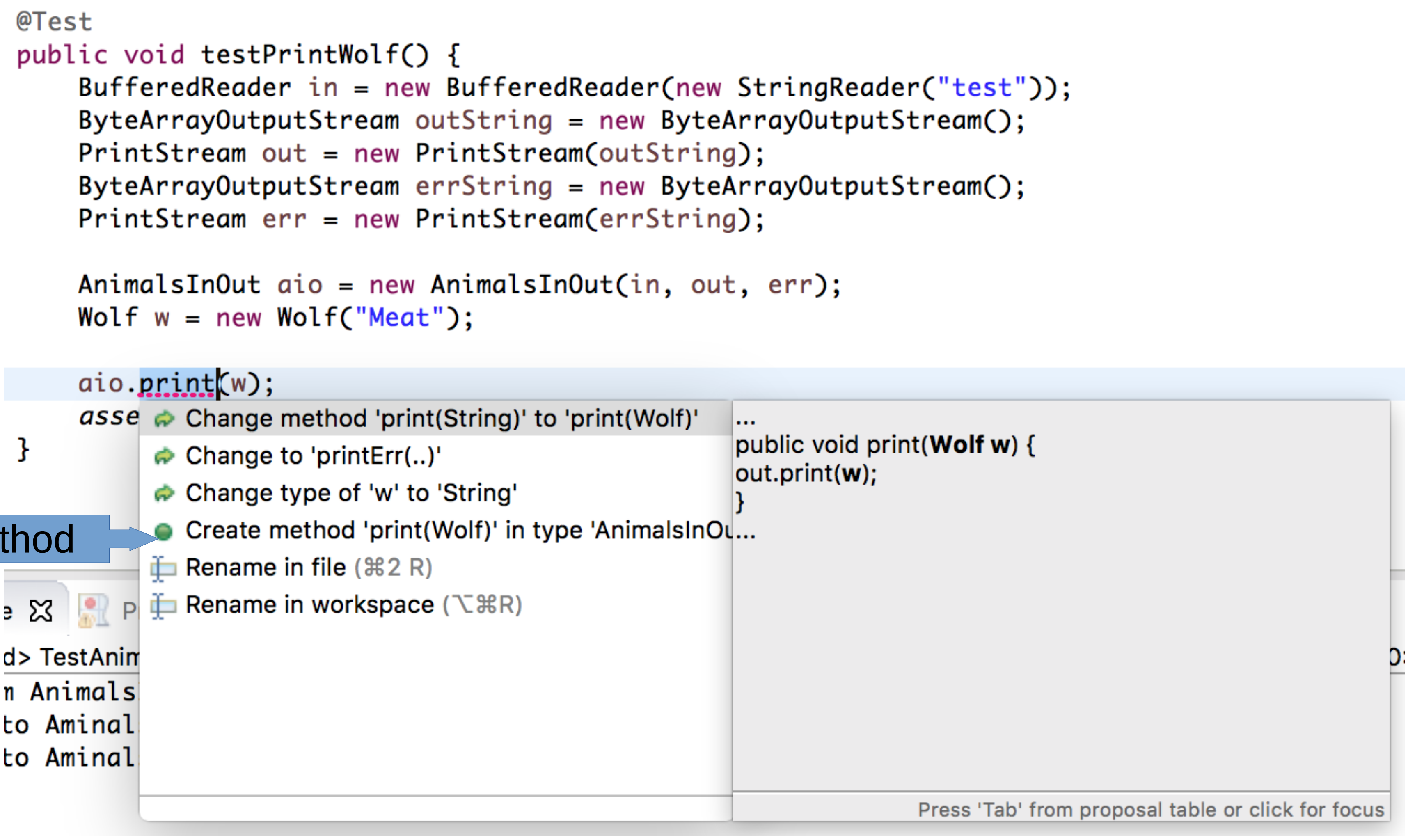
```
@Test
public void testConstructor() {

    BufferedReader in = new BufferedReader(new StringReader("test"));
    ByteArrayOutputStream outString = new ByteArrayOutputStream();
    PrintStream out = new PrintStream(outString);
    ByteArrayOutputStream errString = new ByteArrayOutputStream();
    PrintStream err = new PrintStream(errString);

    AnimalsInOut aio = new AnimalsInOut(in, out, err);

    String s = aio.readLine();
    assertEquals("test", s);
    System.out.println("Read from AnimalsInOut: " + s);

    aio.print("test");
    assertEquals("test", outString.toString());
    System.out.println("Printed to AminalsInOut: "
                       + outString.toString());

    aio.printErr("test");
    assertEquals("test", errString.toString());
    System.out.println("Printed to AminalsInOut: "
                       + errString.toString());
}
```

Print Output



```
<terminated> TestAnimalsInOut [JUnit]
Read from AnimalsInOut: test
Printed to AminalsInOut: test
Printed to AminalsInOut: test
```

Add Print Wolf Test



```
@Test
public void testPrintWolf() {
    BufferedReader in = new BufferedReader(new StringReader("test"));
    ByteArrayOutputStream outString = new ByteArrayOutputStream();
    PrintStream out = new PrintStream(outString);
    ByteArrayOutputStream errString = new ByteArrayOutputStream();
    PrintStream err = new PrintStream(errString);

    AnimalsInOut aio = new AnimalsInOut(in, out, err);
    Wolf w = new Wolf("Meat");

    aio.print(w);
    asse
}
```

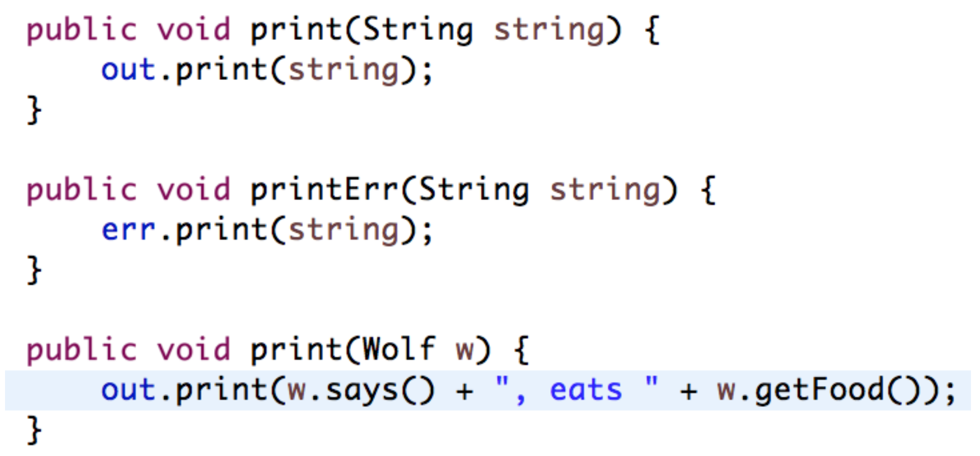
Change method 'print(String)' to 'print(Wolf)'
Change to 'printErr(..)'
Change type of 'w' to 'String'
Create method 'print(Wolf)' in type 'AnimalsInOu...
Rename in file (⌘2 R)
Rename in workspace (⌥⌘R)
Press 'Tab' from proposal table or click for focus

```
...
public void print(Wolf w) {
out.print(w);
}
```

Create new method

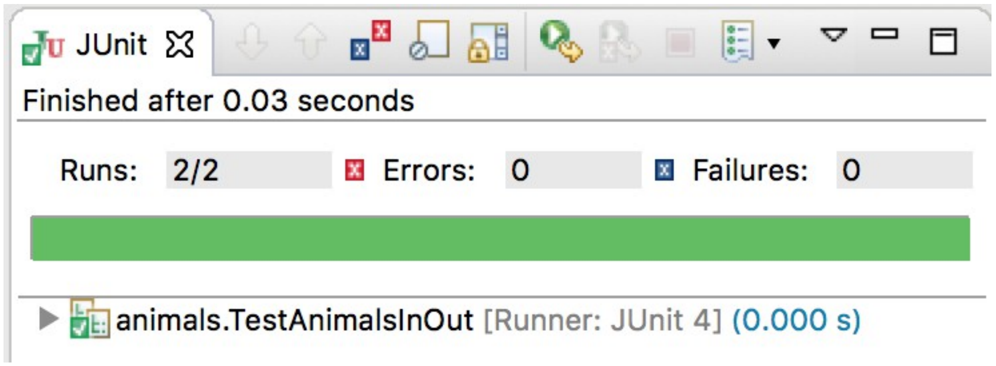
Add print(Wolf) method

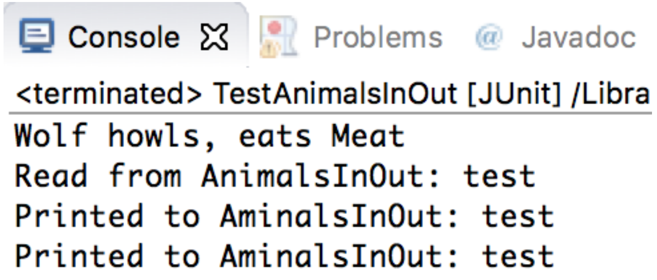
New method



```java
public void print(String string) {
    out.print(string);
}

public void printErr(String string) {
    err.print(string);
}

public void print(Wolf w) {
    out.print(w.says() + ", eats " + w.getFood());
}
```

Run Test


JUnit
Finished after 0.03 seconds
Runs: 2/2
Errors: 0
Failures: 0
animals.TestAnimalsInOut [Runner: JUnit 4] (0.000 s)



```
<terminated> TestAnimalsInOut [JUnit] /Libra
Wolf howls, eats Meat
Read from AnimalsInOut: test
Printed to AminalsInOut: test
Printed to AminalsInOut: test
```

```java
@Test
public void testConstructor() {
    in = new BufferedReader(new StringReader("test"));
    outString = new ByteArrayOutputStream();
    out = new PrintStream(outString);
    errString = new ByteArrayOutputStream();
    err = new PrintStream(errString);

    aio = new AnimalsInOut(in, out, err);
    String s = aio.readLine();
    assertEquals("test", s);
    System.out.println("Read from AnimalsInOut: " + s);

    aio.print("test");
    assertEquals("test", outString.toString());
    System.out.println("Printed to AminalsInOut: "
                    + outString.toString());

    aio.printErr("test");
    assertEquals("test", errString.toString());
    System.out.println("Printed to AminalsInOut: "
                    + errString.toString());
}

@Test
public void testPrintWolf() {
    in = new BufferedReader(new StringReader("test"));
    outString = new ByteArrayOutputStream();
    out = new PrintStream(outString);
    errString = new ByteArrayOutputStream();
    err = new PrintStream(errString);

    aio = new AnimalsInOut(in, out, err);
    Wolf w = new Wolf("Meat");

    aio.print(w);
    assertEquals("Wolf howls, eats Meat", outString.toString());
    System.out.println(outString.toString());
}
```

Refactor Tests: Extract Constructor

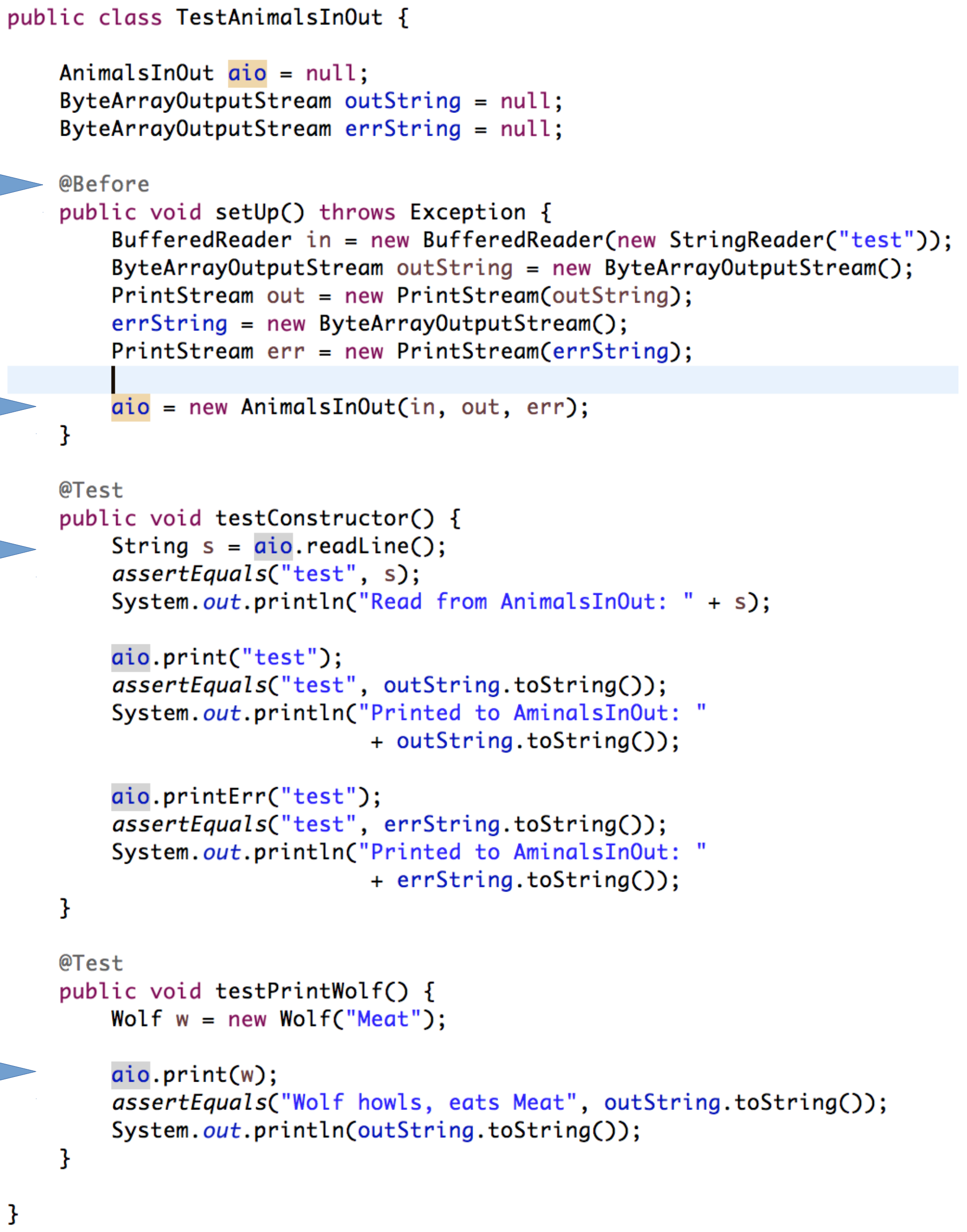
Fields used in tests

@Before means run before each test

New AnimalsInOut before each test

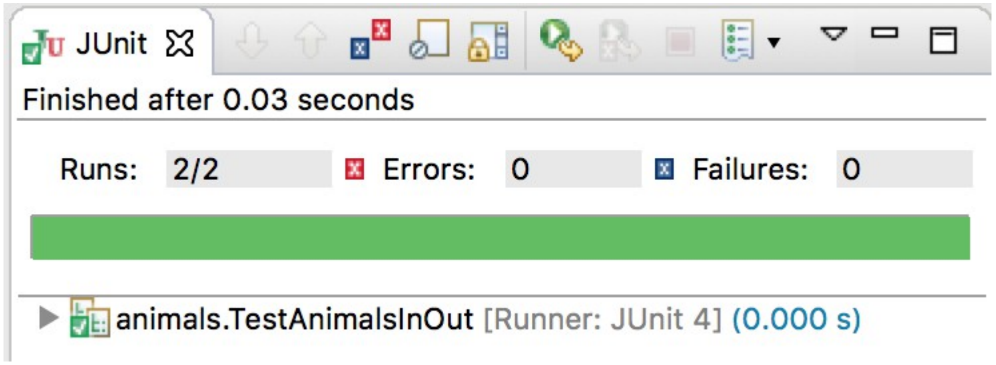
AnimalsInOut created before test

AnimalsInOut created before test



```
public class TestAnimalsInOut {

    AnimalsInOut aio = null;
    ByteArrayOutputStream outString = null;
    ByteArrayOutputStream errString = null;

    @Before
    public void setUp() throws Exception {
        BufferedReader in = new BufferedReader(new StringReader("test"));
        ByteArrayOutputStream outString = new ByteArrayOutputStream();
        PrintStream out = new PrintStream(outString);
        errString = new ByteArrayOutputStream();
        PrintStream err = new PrintStream(errString);

        aio = new AnimalsInOut(in, out, err);
    }

    @Test
    public void testConstructor() {
        String s = aio.readLine();
        assertEquals("test", s);
        System.out.println("Read from AnimalsInOut: " + s);

        aio.print("test");
        assertEquals("test", outString.toString());
        System.out.println("Printed to AminalsInOut: "
                            + outString.toString());

        aio.printErr("test");
        assertEquals("test", errString.toString());
        System.out.println("Printed to AminalsInOut: "
                            + errString.toString());
    }

    @Test
    public void testPrintWolf() {
        Wolf w = new Wolf("Meat");

        aio.print(w);
        assertEquals("Wolf howls, eats Meat", outString.toString());
        System.out.println(outString.toString());
    }

}
```

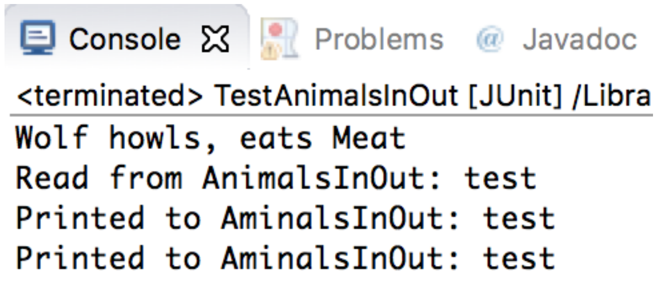
Run Refactored Test



Finished after 0.03 seconds

Runs: 2/2   Errors: 0   Failures: 0

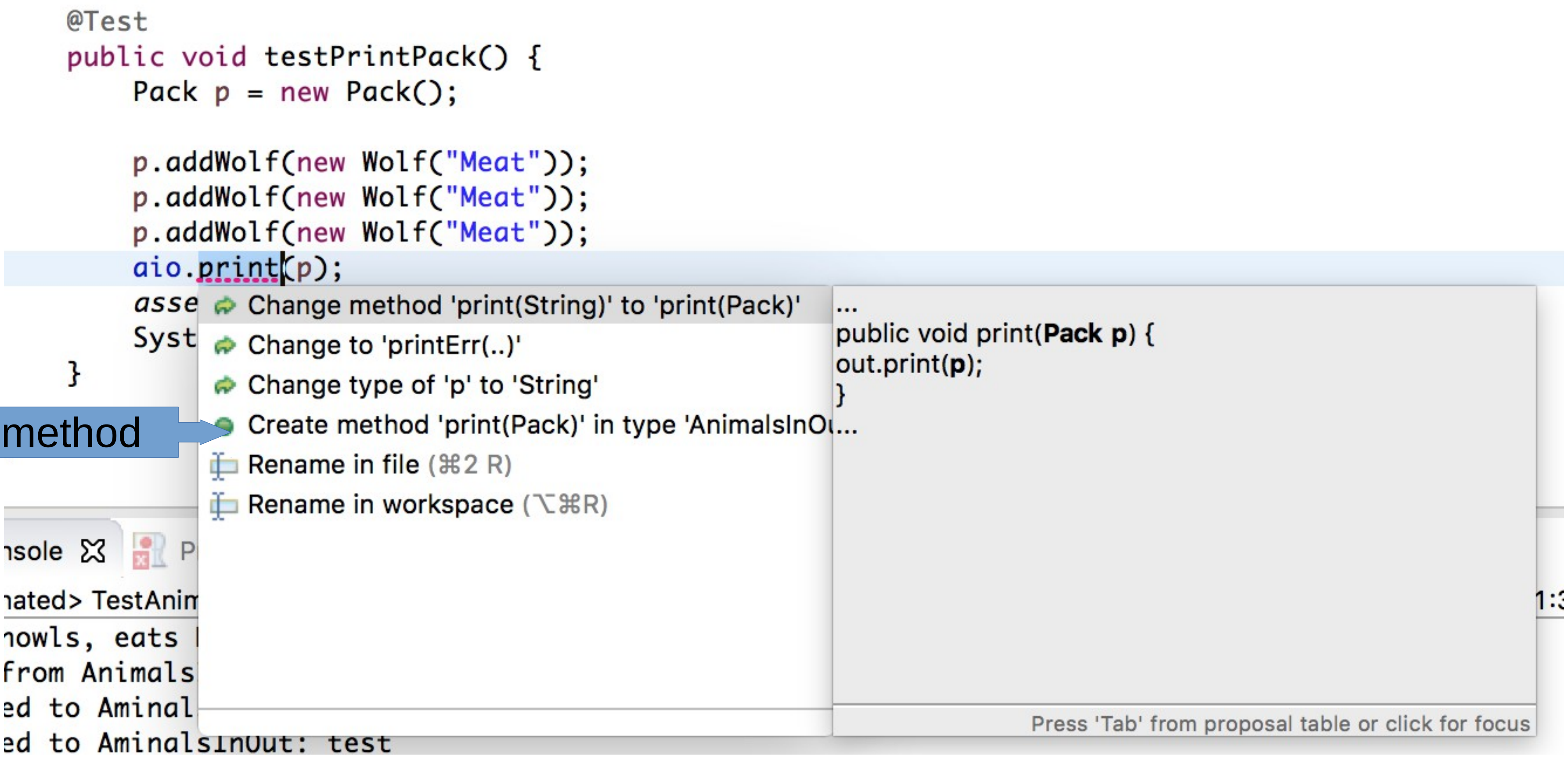
animals.TestAnimalsInOut [Runner: JUnit 4] (0.000 s)



```
<terminated> TestAnimalsInOut [JUnit] /Libra
Wolf howls, eats Meat
Read from AnimalsInOut: test
Printed to AminalsInOut: test
Printed to AminalsInOut: test
```

```java
public class TestAnimalsInOut {

    AnimalsInOut aio = null;
    ByteArrayOutputStream outString = null;
    ByteArrayOutputStream errString = null;

    @Before
    public void setUp() throws Exception {
        BufferedReader in = new BufferedReader(new StringReader("test"));
        outString = new ByteArrayOutputStream();
        PrintStream out = new PrintStream(outString);
        errString = new ByteArrayOutputStream();
        PrintStream err = new PrintStream(errString);

        aio = new AnimalsInOut(in, out, err);
    }

    @Test
    public void testConstructor() {
        String s = aio.readLine();
        assertEquals("test", s);
        System.out.println("Read from AnimalsInOut: " + s);

        aio.print("test");
        assertEquals("test", outString.toString());
        System.out.println("Printed to AminalsInOut: "
                    + outString.toString());

        aio.printErr("test");
        assertEquals("test", errString.toString());
        System.out.println("Printed to AminalsInOut: "
                    + errString.toString());
    }

    @Test
    public void testPrintWolf() {
        Wolf w = new Wolf("Meat");

        aio.print(w);
        assertEquals("Wolf howls, eats Meat", outString.toString());
        System.out.println(outString.toString());
    }

}
```

Add Print Pack Test



```
@Test
public void testPrintPack() {
    Pack p = new Pack();

    p.addWolf(new Wolf("Meat"));
    p.addWolf(new Wolf("Meat"));
    p.addWolf(new Wolf("Meat"));
    aio.print(p);
    asse
    Syst
}
```

Change method 'print(String)' to 'print(Pack)'
Change to 'printErr(..)'
Change type of 'p' to 'String'
Create method 'print(Pack)' in type 'AnimalsInOu...
Rename in file (⌘2 R)
Rename in workspace (⌥⌘R)

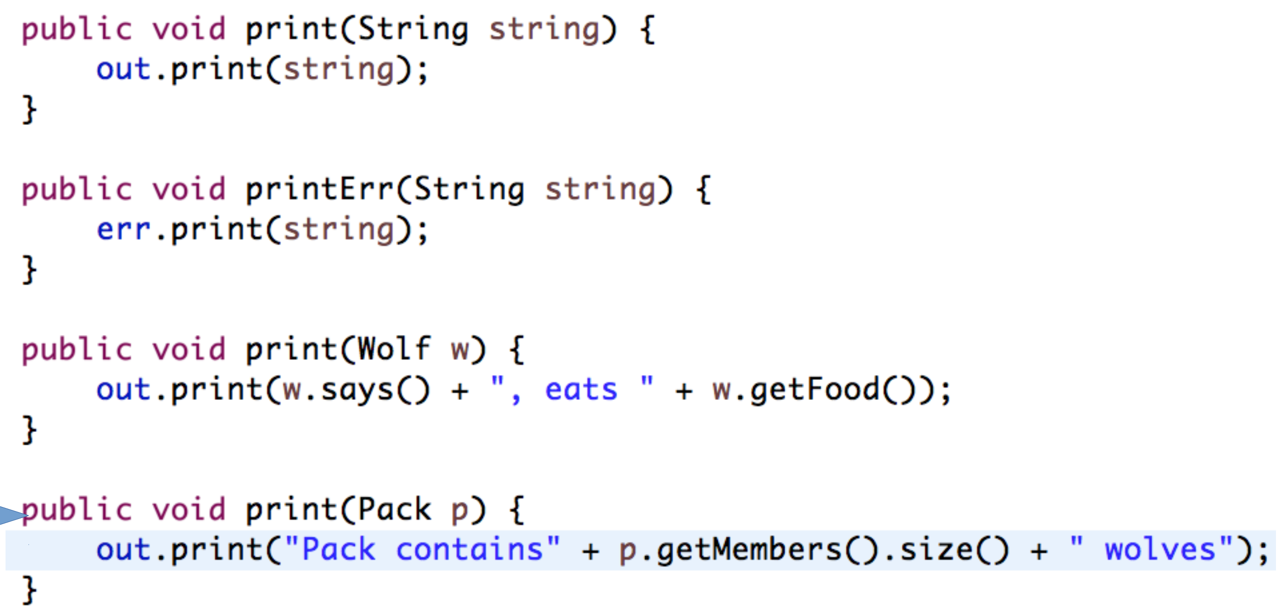
```
...
public void print(Pack p) {
out.print(p);
}
```

Press 'Tab' from proposal table or click for focus
nsole
P
nated> TestAnim
1:3

```
nowls, eats
from Animals
ed to Aminal
ed to AminalsInOut: test
```

Create new method

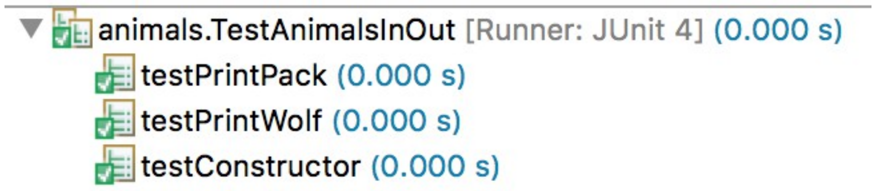
Add print(Pack) method



```
public void print(String string) {
    out.print(string);
}

public void printErr(String string) {
    err.print(string);
}

public void print(Wolf w) {
    out.print(w.says() + ", eats " + w.getFood());
}

public void print(Pack p) {
    out.print("Pack contains" + p.getMembers().size() + " wolves");
}
```

New method

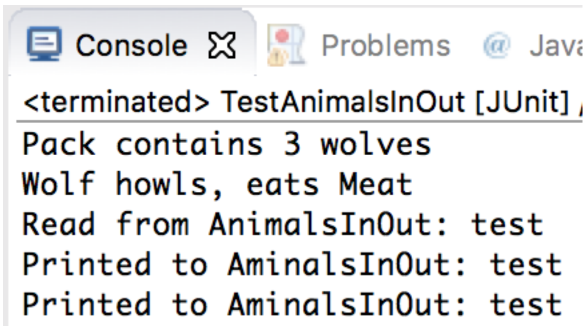
Run Tests



```
animals.TestAnimalsInOut [Runner: JUnit 4] (0.000 s)
    testPrintPack (0.000 s)
    testPrintWolf (0.000 s)
    testConstructor (0.000 s)
```

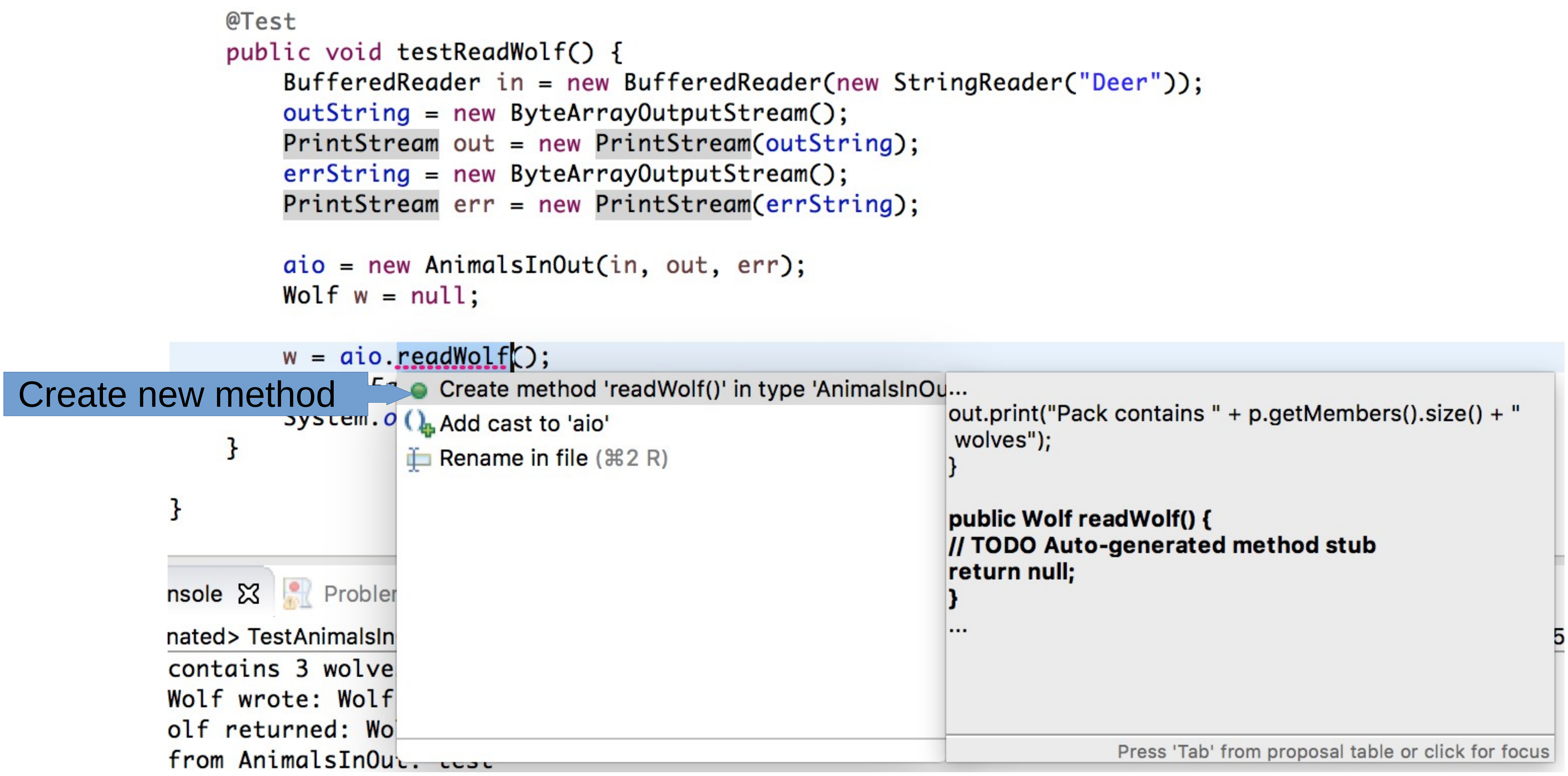
```
@Test
public void testPrintPack() {
    Pack p = new Pack();

    p.addWolf(new Wolf("Meat"));
    p.addWolf(new Wolf("Meat"));
    p.addWolf(new Wolf("Meat"));
    aio.print(p);
    assertEquals("Pack contains 3 wolves", outString.toString());
    System.out.println(outString.toString());
}
```

Print Output



```
<terminated> TestAnimalsInOut [JUnit]
Pack contains 3 wolves
Wolf howls, eats Meat
Read from AnimalsInOut: test
Printed to AminalsInOut: test
Printed to AminalsInOut: test
```

Add Read Wolf Test



```
@Test
public void testReadWolf() {
    BufferedReader in = new BufferedReader(new StringReader("Deer"));
    outString = new ByteArrayOutputStream();
    PrintStream out = new PrintStream(outString);
    errString = new ByteArrayOutputStream();
    PrintStream err = new PrintStream(errString);

    aio = new AnimalsInOut(in, out, err);
    Wolf w = null;

    w = aio.readWolf();
    System.o
}

}
```

Create new method

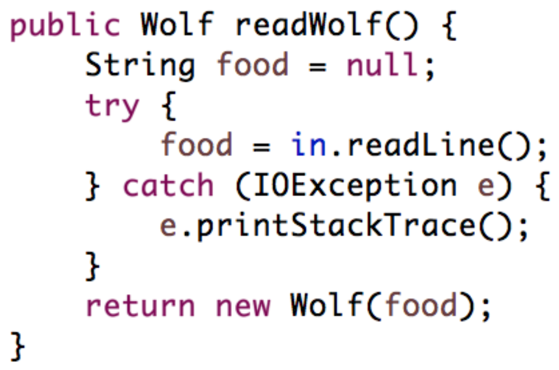
```
out.print("Pack contains " + p.getMembers().size() + "
 wolves");
}

public Wolf readWolf() {
// TODO Auto-generated method stub
return null;
}
...
```

```
nated> TestAnimalsIn
contains 3 wolve
Wolf wrote: Wolf
olf returned: Wo
from AnimalsInOut.
```

Add readWolf() method

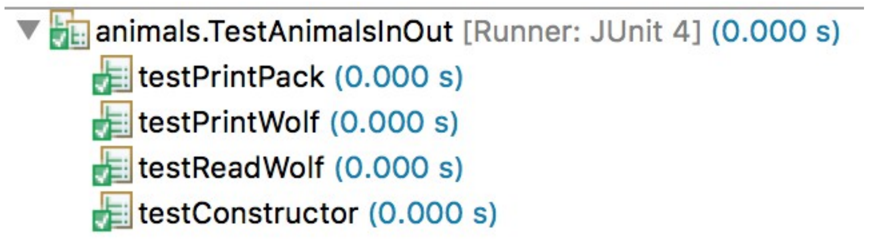
readLine() throws an error

Create a new wolf to return

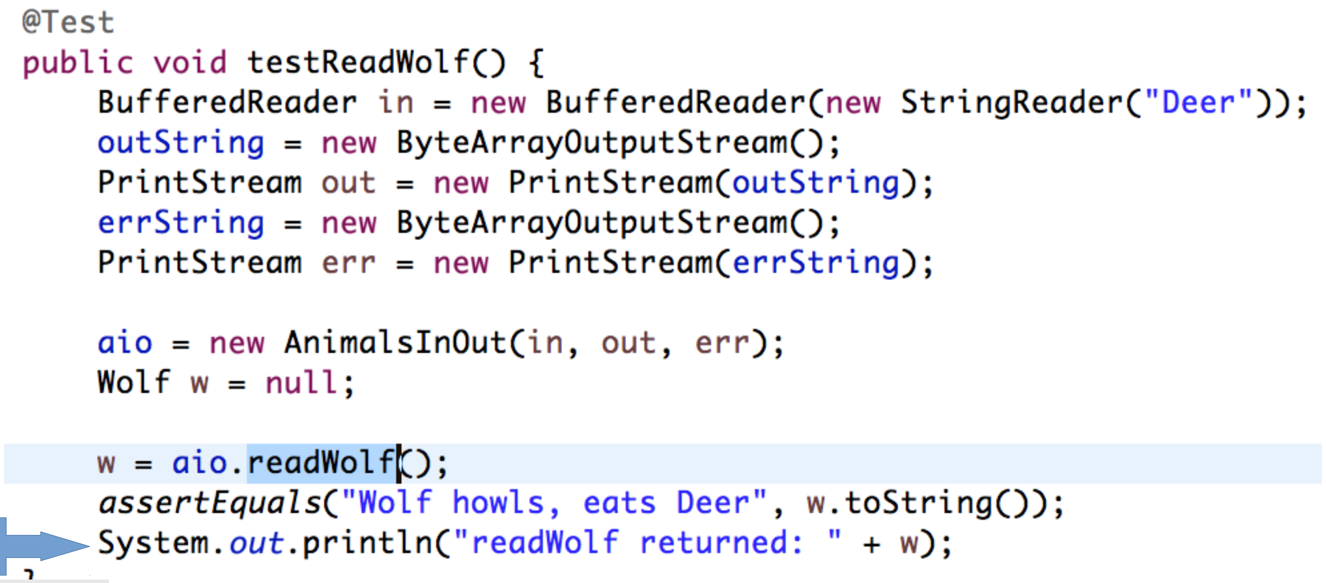


```
public Wolf readWolf() {
    String food = null;
    try {
        food = in.readLine();
    } catch (IOException e) {
        e.printStackTrace();
    }
    return new Wolf(food);
}
```

Run Tests

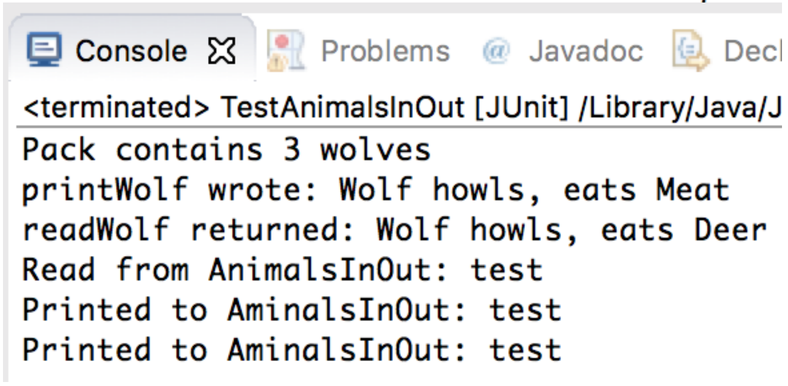


```
animals.TestAnimalsInOut [Runner: JUnit 4] (0.000 s)
    testPrintPack (0.000 s)
    testPrintWolf (0.000 s)
    testReadWolf (0.000 s)
    testConstructor (0.000 s)
```



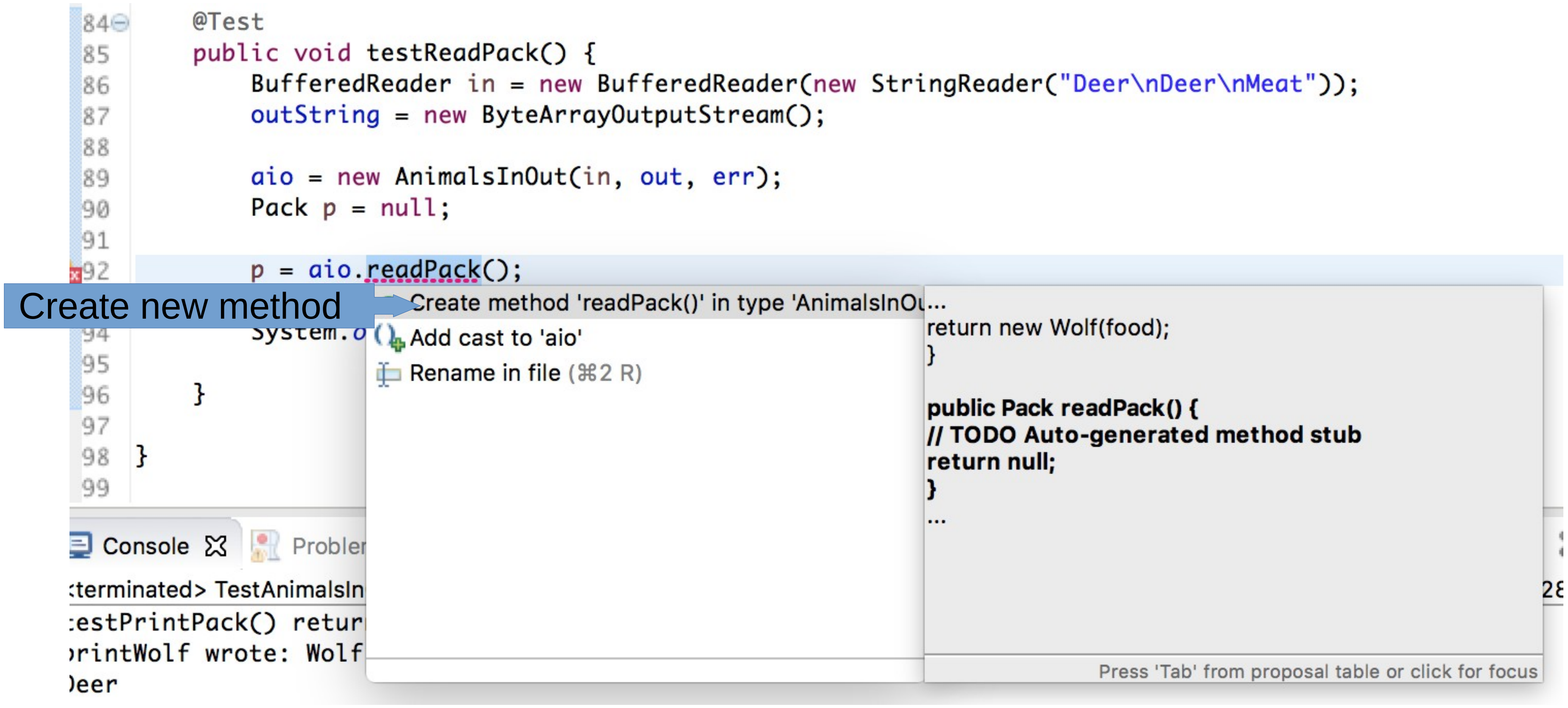
```java
@Test
public void testReadWolf() {
    BufferedReader in = new BufferedReader(new StringReader("Deer"));
    outString = new ByteArrayOutputStream();
    PrintStream out = new PrintStream(outString);
    errString = new ByteArrayOutputStream();
    PrintStream err = new PrintStream(errString);

    aio = new AnimalsInOut(in, out, err);
    Wolf w = null;

    w = aio.readWolf();
    assertEquals("Wolf howls, eats Deer", w.toString());
    System.out.println("readWolf returned: " + w);
```

Print Output



```
<terminated> TestAnimalsInOut [JUnit] /Library/Java/J
Pack contains 3 wolves
printWolf wrote: Wolf howls, eats Meat
readWolf returned: Wolf howls, eats Deer
Read from AnimalsInOut: test
Printed to AminalsInOut: test
Printed to AminalsInOut: test
```

Add Read Pack Test



```
    @Test
    public void testReadPack() {
        BufferedReader in = new BufferedReader(new StringReader("Deer\nDeer\nMeat"));
        outString = new ByteArrayOutputStream();

        aio = new AnimalsInOut(in, out, err);
        Pack p = null;

        p = aio.readPack();
        System.o
        
    }

}
```

Create new method
Create method 'readPack()' in type 'AnimalsInOu...
Add cast to 'aio'
Rename in file (⌘2 R)
Press 'Tab' from proposal table or click for focus
Console
Probler

```
return new Wolf(food);
}

public Pack readPack() {
// TODO Auto-generated method stub
return null;
}
...
```

```
<terminated> TestAnimalsIn
testPrintPack() retur
printWolf wrote: Wolf
Deer
```

Add readPack() method

Create a Scanner called s

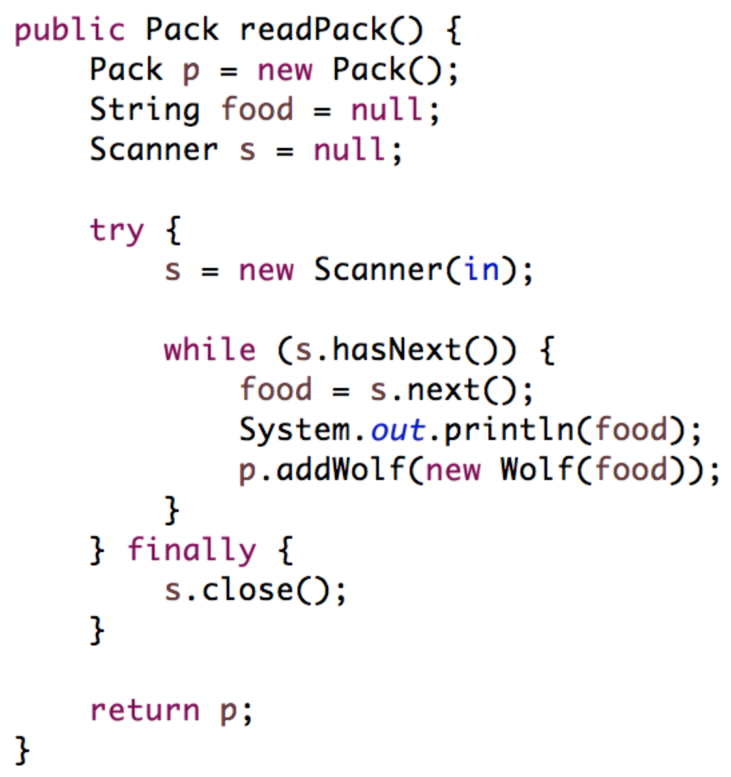
Initialize it with the input BufferedReader

Keep reading it until there is nothing left

Read the wolf's food from the stream

Create a new wolf and add it to the pack

Return the pack

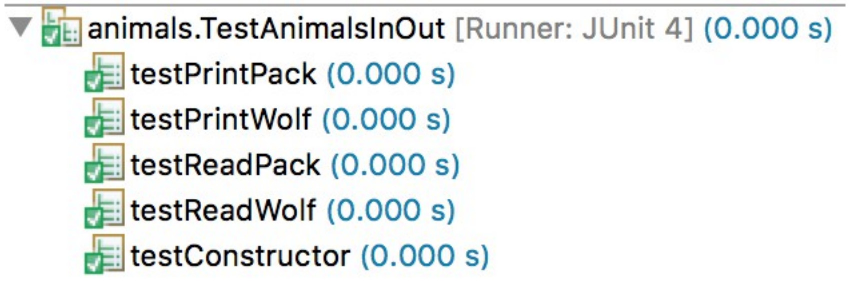


```
public Pack readPack() {
    Pack p = new Pack();
    String food = null;
    Scanner s = null;

    try {
        s = new Scanner(in);

        while (s.hasNext()) {
            food = s.next();
            System.out.println(food);
            p.addWolf(new Wolf(food));
        }
    } finally {
        s.close();
    }

    return p;
}
```

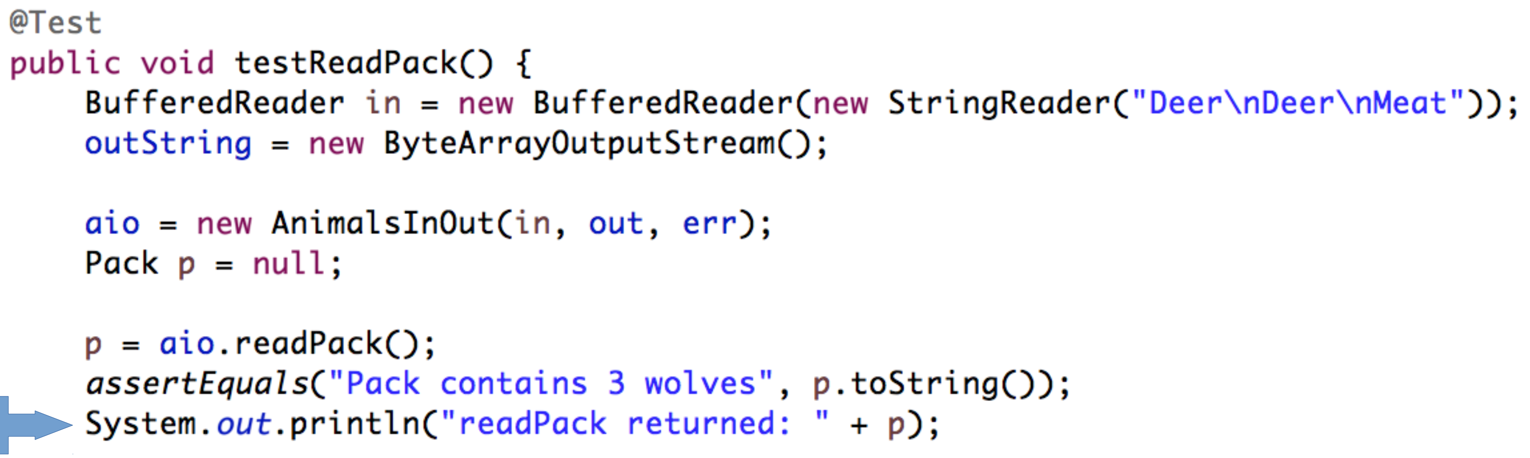
Scanner

- A Scanner object parses a stream
- It breaks the stream into *tokens*, which are strings separated by white space
  - White space is a space, tab or newline
- Given a Scanner s,
  - s.hasNext() returns true if there is an additional token; false otherwise
  - s.next() returns the next token

Run Tests

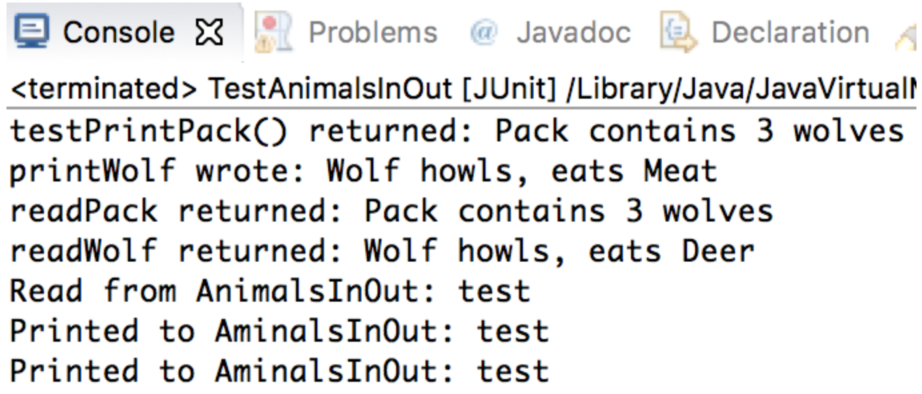


```
animals.TestAnimalsInOut [Runner: JUnit 4] (0.000 s)
    testPrintPack (0.000 s)
    testPrintWolf (0.000 s)
    testReadPack (0.000 s)
    testReadWolf (0.000 s)
    testConstructor (0.000 s)
```



```
@Test
public void testReadPack() {
    BufferedReader in = new BufferedReader(new StringReader("Deer\nDeer\nMeat"));
    outString = new ByteArrayOutputStream();

    aio = new AnimalsInOut(in, out, err);
    Pack p = null;

    p = aio.readPack();
    assertEquals("Pack contains 3 wolves", p.toString());
    System.out.println("readPack returned: " + p);
```

Print Output



```
<terminated> TestAnimalsInOut [JUnit] /Library/Java/JavaVirtual
testPrintPack() returned: Pack contains 3 wolves
printWolf wrote: Wolf howls, eats Meat
readPack returned: Pack contains 3 wolves
readWolf returned: Wolf howls, eats Deer
Read from AnimalsInOut: test
Printed to AminalsInOut: test
Printed to AminalsInOut: test
```