Object Oriented Programming

Week 3 Part 1
Inheritance

Lecture

- What is Inheritance
- Declaring Inheritance in Java
- Declaring Inheritance using Eclipse

What is Inheritance in OOP?

Object, Classes and Inheritance

- Objects let us reason about programs as if they were constructed with things.
- Classes let us define types of objects.
- Inheritance lets us define a particular type of relationship between types of objects.
  - Inheritance defines and *is-a* relationship.
  - E.g.
    - A dog is a mammal
    - A mammal is an animal

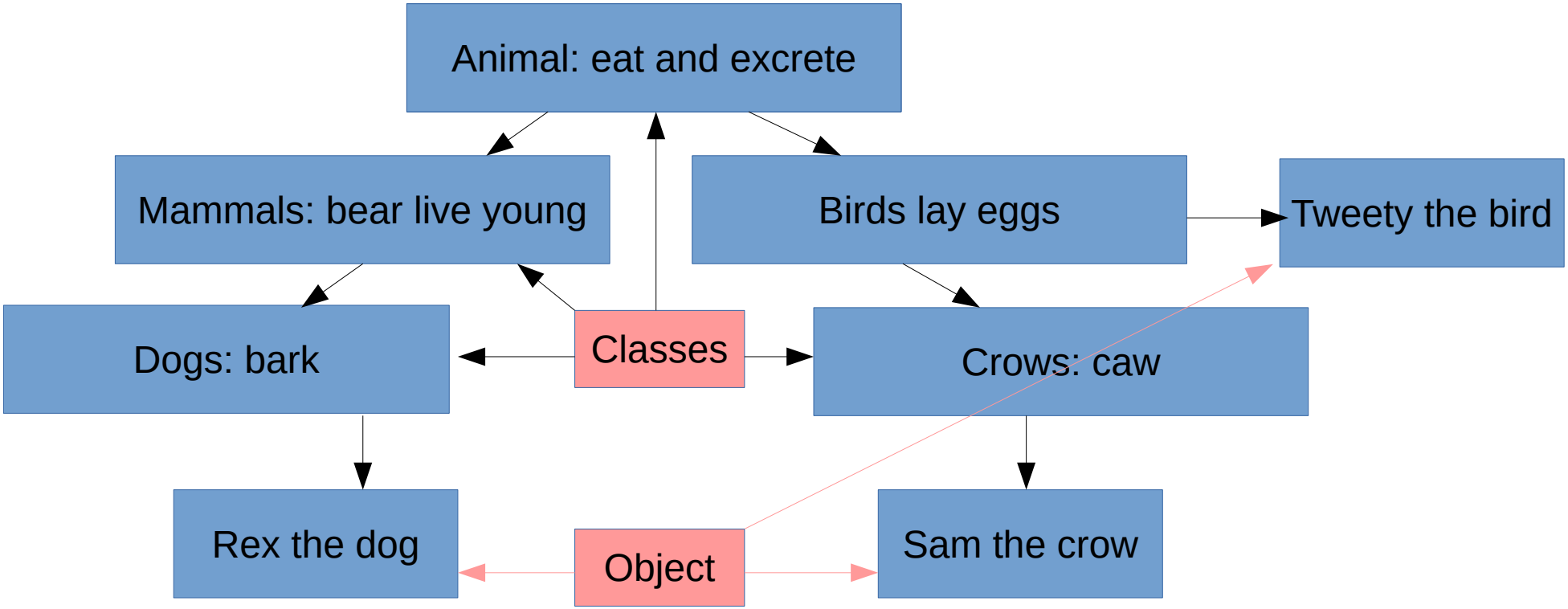
Object inherit properties

- An animal eats, respires and excretes
- All mammals, because they are animals, eat and excrete
- In addition mammals bear live young.
- Dogs,
  - because they are mammals, bear live young
  - Because they are animals, eat and excrete

Birds are animals but nor mamals

- Birds
  - Because they are animals, eat and excrete
  - But they do not bear live young, instead they lay eggs

Animal Hierarchy



Defining the Hierarchy

- A super class is the parent of sub classes in the hierarchy
  - E.g., Dogs are sub classes of the Mammal super class
- Each sub class *inherits* all of the member variables and methods of the super class
- The sub class may *extend* the super class by defining new member variables or methods
- A sub class may *override* a method by redefining a method that is defined in the super class.

Expressing Inheritance in Java

To specify a class

- Specify the class name
  - Public means that others can use it
- Specify the fields or member variables
  - Specified private so others cannot see them
  - Others access fields through setter and getter functions so the class can control access
- Specify the constructor method
  - Declared public if the class is public
  - Usually sets up the fields
- Specify the methods
  - Public for those others can use
  - Private for those used only in the class

Animal Class

Package: animals

Class: Animal

Field: food

Constructor: specify kind of food

Getter: return kind of food eaten

```
package animals;

public class Animal {

    private String food;

    public Animal(String eats) {
        food = eats;
    }

    public String getFood() {
        return food;
    }

}
```

To specify a sub-class

- Use *extends* to specify the super-class
  - The sub-class inherits all fields and methods of the super class
  - The sub-class can only use the fields and methods that are declared private or protected
- Use *super()* to call the super-class's constructor
  - Must be the first function call in the class's constructor
- Specify fields and methods as for any class

Mammal Class

Package: animals

Class: Mammal

Field: offspring

Constructor:
super(): superclass constructor
specify kind of food
Sets offspring field

Getter: return offspring field

```
package animals;

public class Mammal extends Animal {

    private String offspring;

    public Mammal(String food, String young) {
        super(food);
        offspring = young;
    }

    public String getOffspring() {
        return offspring;
    }

}
```

To override a method

- Next we will add a method says() that will indicate what sound the animal makes
- We want to make sure that all Animals have the says() method, so we add it to Animals and Mammals.
- We can use the @Override annotation to indicate the it overrides previous definitions
  - The annotation is not needed, but it clarifies what is going on.

Animal and Mammal changes

```java
package animals;

public class Animal {

    private String food;

    public Animal(String eats) {
        food = eats;
    }

    public String getFood() {
        return food;
    }

    public String says() {
        return "Animals say many different things";
    }

}
```

```java
package animals;

public class Mammal extends Animal {

    private String offspring;

    public Mammal(String food, String young) {
        super(food);
        offspring = young;
    }

    public String getOffspring() {
        return offspring;
    }

    @Override
    public String says() {
        return "Mammals say many different things";
    }

}
```

Add says() to Dog

```
package animals;

public class Dog extends Mammal {

    public Dog(String food, String young) {
        super(food, young);
    }

    @Override
    public String says() {
        return "Dog goes woof";
    }

}
```

Overrides Mammal says()

New method says()

Keyword Static

- The keyword static specifies that a variable or method belongs to the class, not the object created from the class.
- For example, live birth, is a characteristic of Mammals as a class rather than species of mammals.
  - We can specify that the method of birth for Mammals is static and simplify the constructor
  - We can also specify it as final, since we will not change it
    - Final declares a constant

Changes to Mammal

Field offspring: static and final

Only one parameter to constructor
Only need call super()
All Mammals will have live offspring

```
package animals;

public class Mammal extends Animal {

    private static final String offspring = "Live";

    public Mammal(String food) {
        super(food);
    }

    public String getOffspring() {
        return offspring;
    }

    @Override
    public String says() {
        return "Mammals say many different things";
    }


}
```

Changes to Dog

Only one parameter to constructor
Only need to specify food
Live offspring inherited from Mammal

```
package animals;

public class Dog extends Mammal {

    public Dog(String food) {
        super(food);
    }

    @Override
    public String says() {
        return "Dog goes woof";
    }

}
```

Test Class

- We have Unit tests
  - TestAnimal
  - TestMammal
  - TestDog
- We need a system or interaction test
  - Tests the system as a whole from the point of view of the user
  - Tests the interactions between classes

Add Test

Package: animals

Contains only main()

Creates objects

Calls methods

```java
package animals;

public class Test {

    public static void main(String[] args) {
        Animal a = new Animal("Food");
        Mammal m = new Mammal("Milk");
        Dog d = new Dog("Meat");

        System.out.println("Animals eat " + a.getFood());
        System.out.println(a.says());
        System.out.println("Mammals eat " + m.getFood());
        System.out.println("Mammal young are " + m.getOffspring());
        System.out.println(m.says());
        System.out.println("Dogs eat " + d.getFood());
        System.out.println("Dog young are " + d.getOffspring());
        System.out.println(d.says());
    }
}
```

```
Animals eat Food
Animals say many different things
Mammals eat Milk
Mammal young are Live
Mammals say many different things
Dogs eat Meat
Dog young are Live
Dog goes woof
```

Output

Add Bird

Package: animals

Extends Animal

Offspring are Eggs

Says tweet

```
package animals;

public class Bird extends Animal {

    private static final String offspring = "Eggs";

    public Bird(String eats) {
        super(eats);
    }

    public String getOffspring() {
        return offspring;
    }

    @Override
    public String says() {
        return "Bird says tweet";
    }

}
```

Add Crow

Package: animals

Extends Animal

Says caw

```
package animals;

public class Crow extends Bird {

    public Crow(String eats) {
        super(eats);
        // TODO Auto-generated constructor stub
    }

    @Override
    public String says() {
        return "Crow goes caw";
    }

}
```

Update Test

Package: animals

Contains only main()

Creates objects

Calls methods

```java
package animals;

public class Test {

    public static void main(String[] args) {
        Animal a = new Animal("Food");
        Mammal m = new Mammal("Milk");
        Dog d = new Dog("Meat");
        Bird b = new Bird("Food");
        Crow c = new Crow("Seeds");

        System.out.println("Animals eat " + a.getFood());
        System.out.println(a.says());
        System.out.println("Mammals eat " + m.getFood());
        System.out.println("Mammal young are " + m.getOffspring());
        System.out.println(m.says());
        System.out.println("Dogs eat " + d.getFood());
        System.out.println("Dog young are " + d.getOffspring());
        System.out.println(d.says());
        System.out.println("Birds eat " + b.getFood());
        System.out.println("Bird young are " + b.getOffspring());
        System.out.println(b.says());
        System.out.println("Crows eat " + c.getFood());
        System.out.println("Crow young are " + c.getOffspring());
        System.out.println(c.says());
    }

}
```

Output

```
Animals eat Food
Animals say many different things
Mammals eat Milk
Mammal young are Live
Mammals say many different things
Dogs eat Meat
Dog young are Live
Dog goes woof
Birds eat Food
Bird young are Eggs
Bird says tweet
Crows eat Seeds
Crow young are Eggs
Crow goes caw
```

Expressing Inheritance using Eclipse

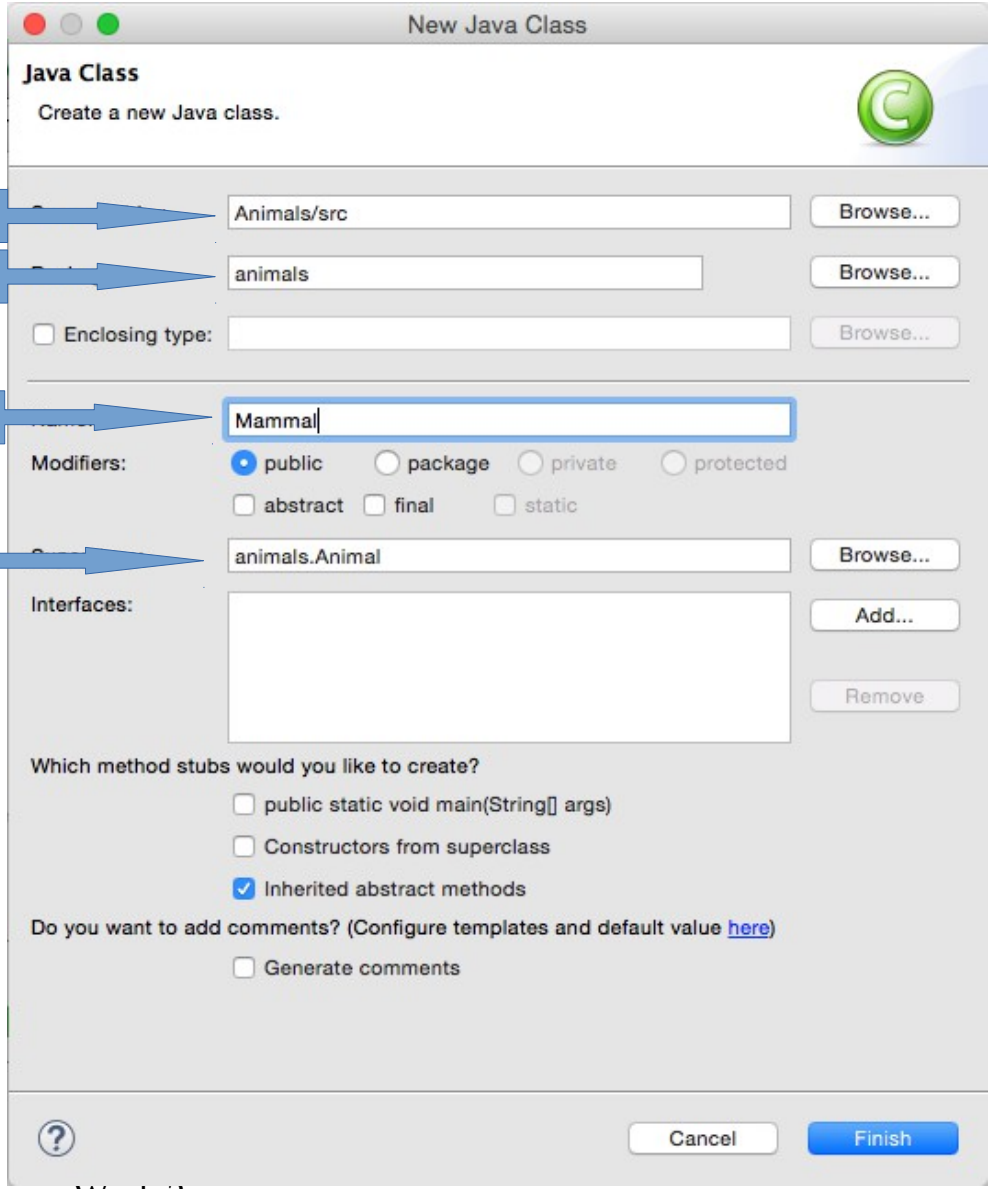
Create Animal Class

Source folder: Animals project; src

Package: animals

Name: Mammal

Parent Class: animals.Animal



Template Package

```
package animals;

public class Animal {

}
```

Create TestAnimals JUnit Test

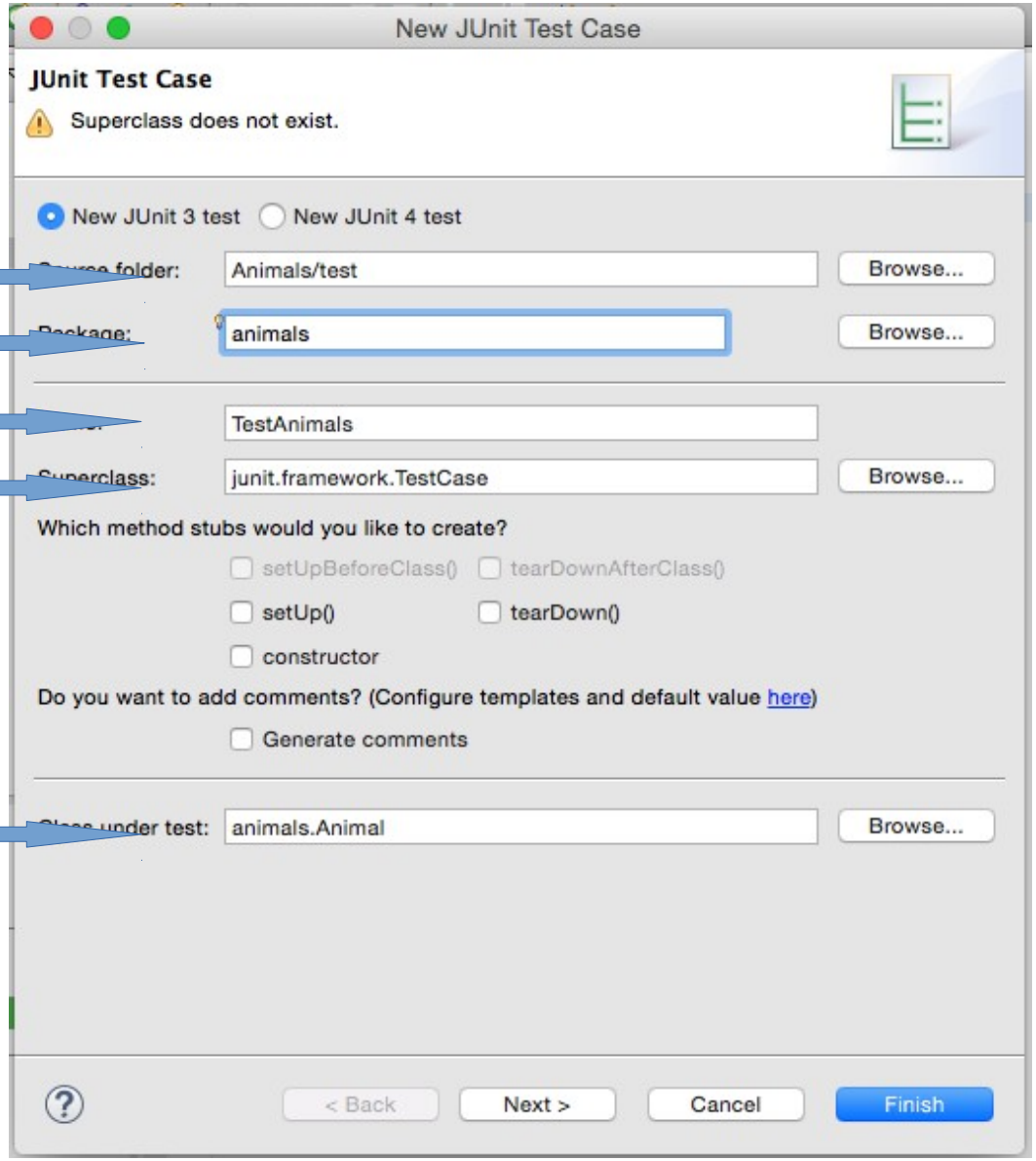
Source folder: Animals project; test

Package: animals

Name: TestAnimal

Parent Class: TestCase

Class under test: animals.Animal



TestAnimals

```java
package animals;

import static org.junit.Assert.*;

public class TestAnimals {

    @Test
    public void testConstructor() {
        Animal a = new Animal("Unknown");
        assertEquals(a.getFood(), "Unknown");
    }

}
```

Constructor and Getter added

```java
package animals;

public class Animal {

    private String food;

    public Animal(String eats) {
        food = eats;
    }

    public String getFood() {
        return food;
    }

}
```

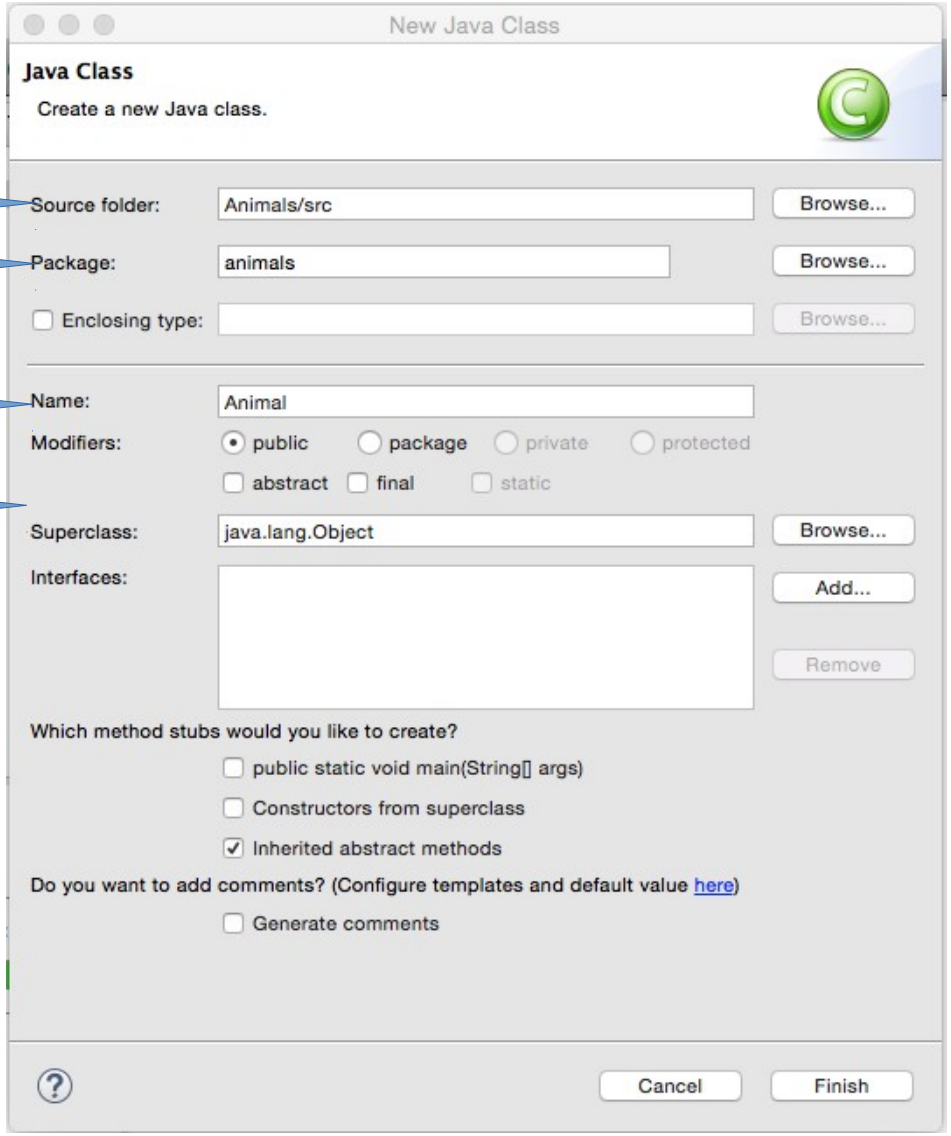
Create Mammal Class

Source folder: Animals project; src

Package: animals

Name: Animal

Parent Class: Object



Template Mammal Class

Error:
Animal(String)

```
package animals;

public class Mammal extends Animal {

}
```

Error Message:

Implicit super constructor Animal() is undefined for default constructor. Must define an explicit constructor

Create TestMammal JUnit Test

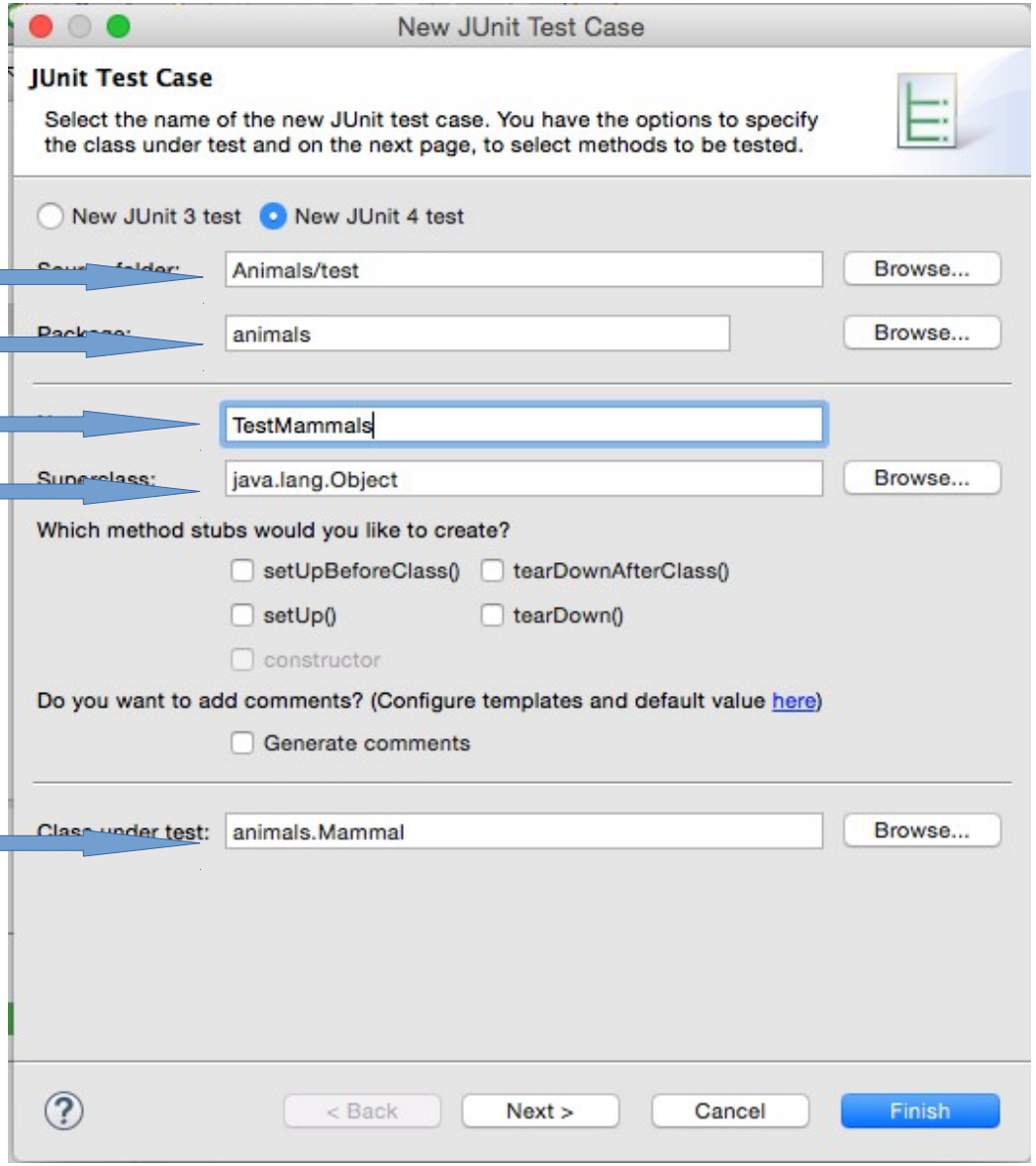
Source folder: Animals project; test

Package: animals

Name: TestMammal

Parent Class: java.lang.Object

Class under test: animals.Mammal



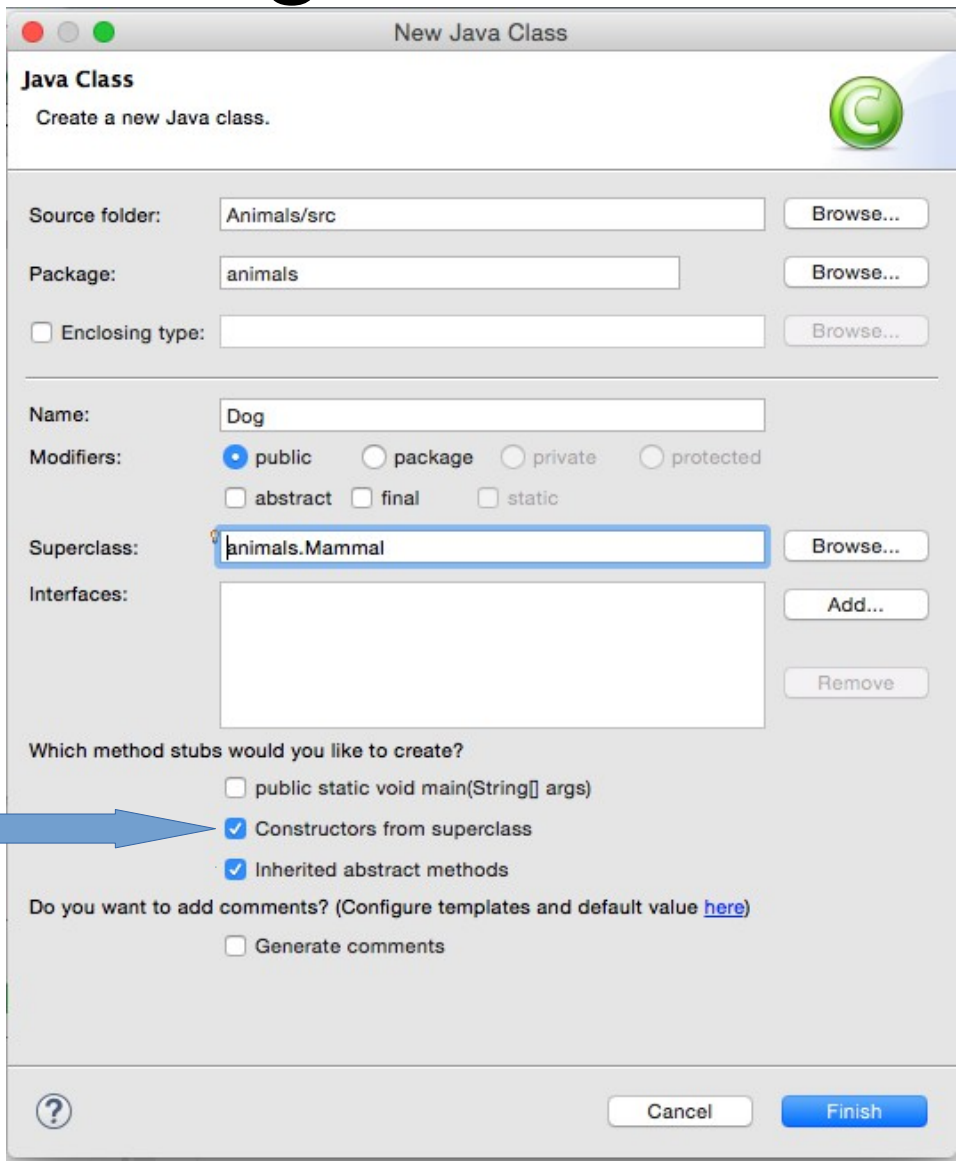
Mammal Test

```java
package animals;

import static org.junit.Assert.*;

public class TestMammals {

    @Test
    public void testConstructor() {
        Mammal m = new Mammal("Milk", "Live");
        assertEquals(m.getFood(), "Milk");
        assertEquals(m.getOffspring(), "Live");
    }

}
```

Mammal Constructor and Getter

```java
package animals;

public class Mammal extends Animal {

    private String offspring;

    public Mammal(String food, String young) {
        super(food);
        offspring = young;
    }

    public String getOffspring() {
        return offspring;
    }

}
```

Create Dog



Check to automatically create constructor

Dog Template

Constructor with super() added

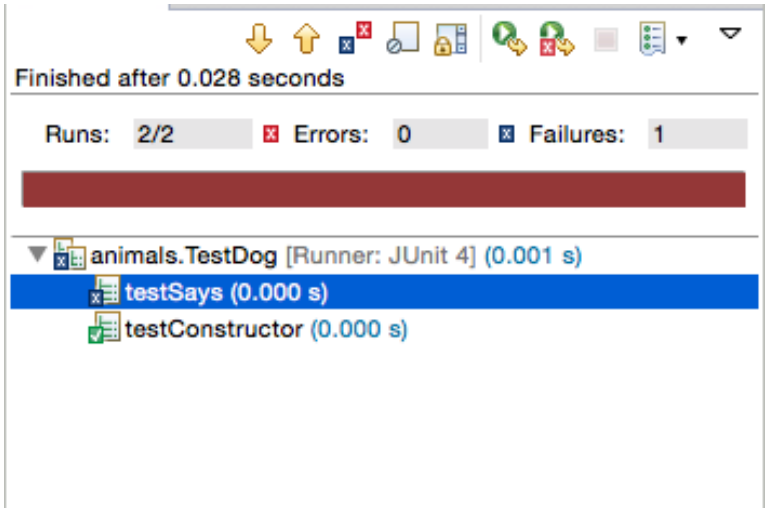
```
package animals;

public class Dog extends Mammal {

    public Dog(String food, String young) {
        super(food, young);
        // TODO Auto-generated constructor stub
    }

}
```

Dog Test

```java
package animals;

import static org.junit.Assert.*;

public class TestDog {

    @Test
    public void testConstructor() {
        Dog d = new Dog("Meat", "Live");
        assertEquals(d.getFood(), "Meat");
        assertEquals(d.getOffspring(), "Live");
    }

}
```

Add says() to Dog—Test First

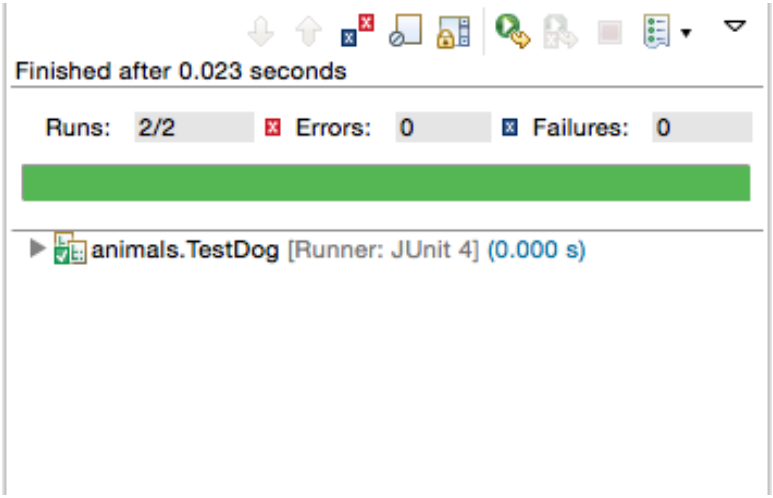
```java
package animals;

import static org.junit.Assert.*;

public class TestDog {

    @Test
    public void testConstructor() {
        Dog d = new Dog("Meat", "Live");
        assertEquals(d.getFood(), "Meat");
        assertEquals(d.getOffspring(), "Live");
    }

    @Test
    public void testSays() {
        Dog d = new Dog("Milk", "Live");
        assertEquals(d.says(), "Dog goes woof");
    }

}
```



Test fails: no says

Add says to Dog class

```java
package animals;

public class Dog extends Mammal {

    public Dog(String food, String young) {
        super(food, young);
    }

    public String says() {
        return "Dog goes woof";
    }

}
```

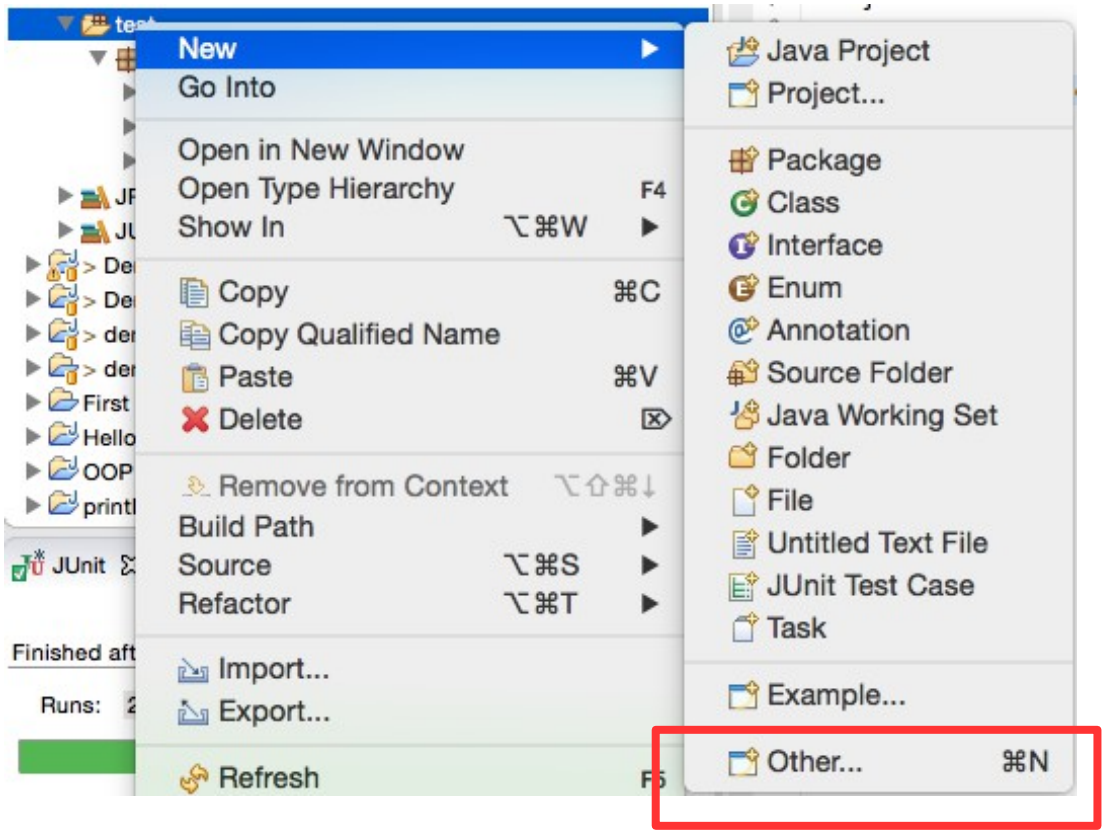


Test passes

Refactor Mammal

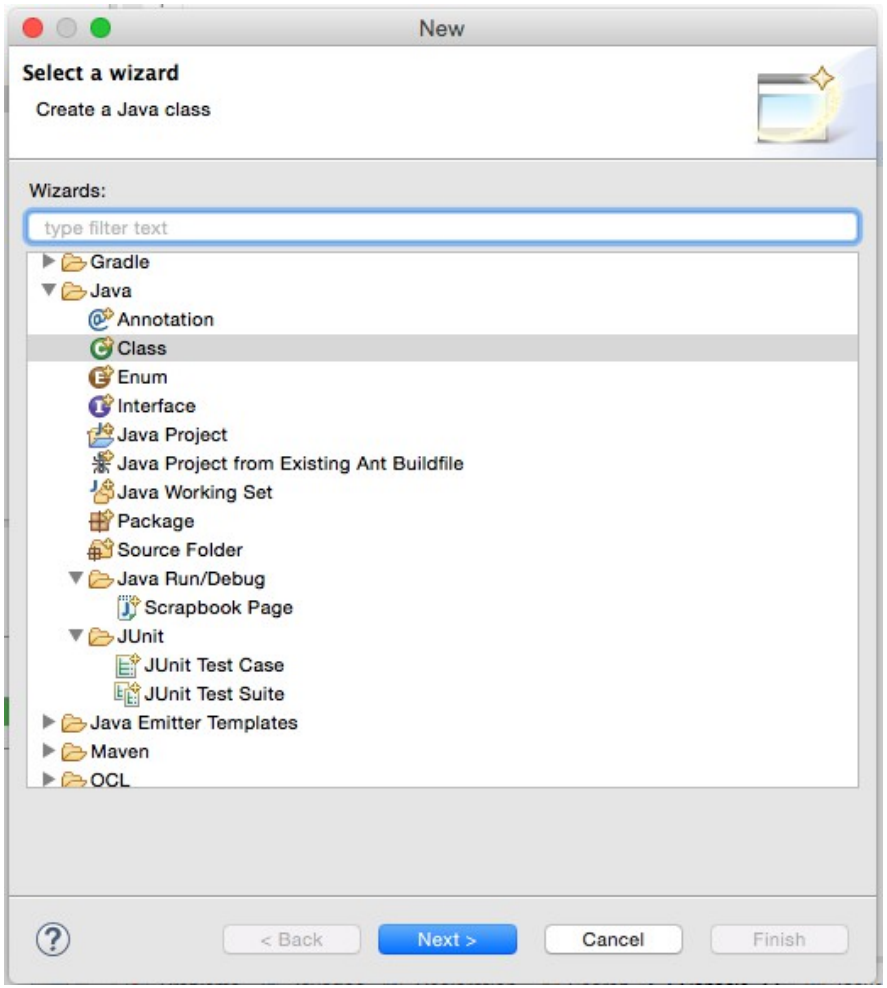
- Changing offspring to be static and final is refactoring.
  - It makes the program correspond more closely to our expectations of Mammals and Dogs
- To refactor we need to run all of the tests, because we are changing interactions between classes
  - We want to make sure we are not breaking another class when we change a super class
- We create a JUnit test suite to run all of the tests

JUnit Test Suite (New > Other)



1. Double click test source folder
2. Select New
3. Select Other

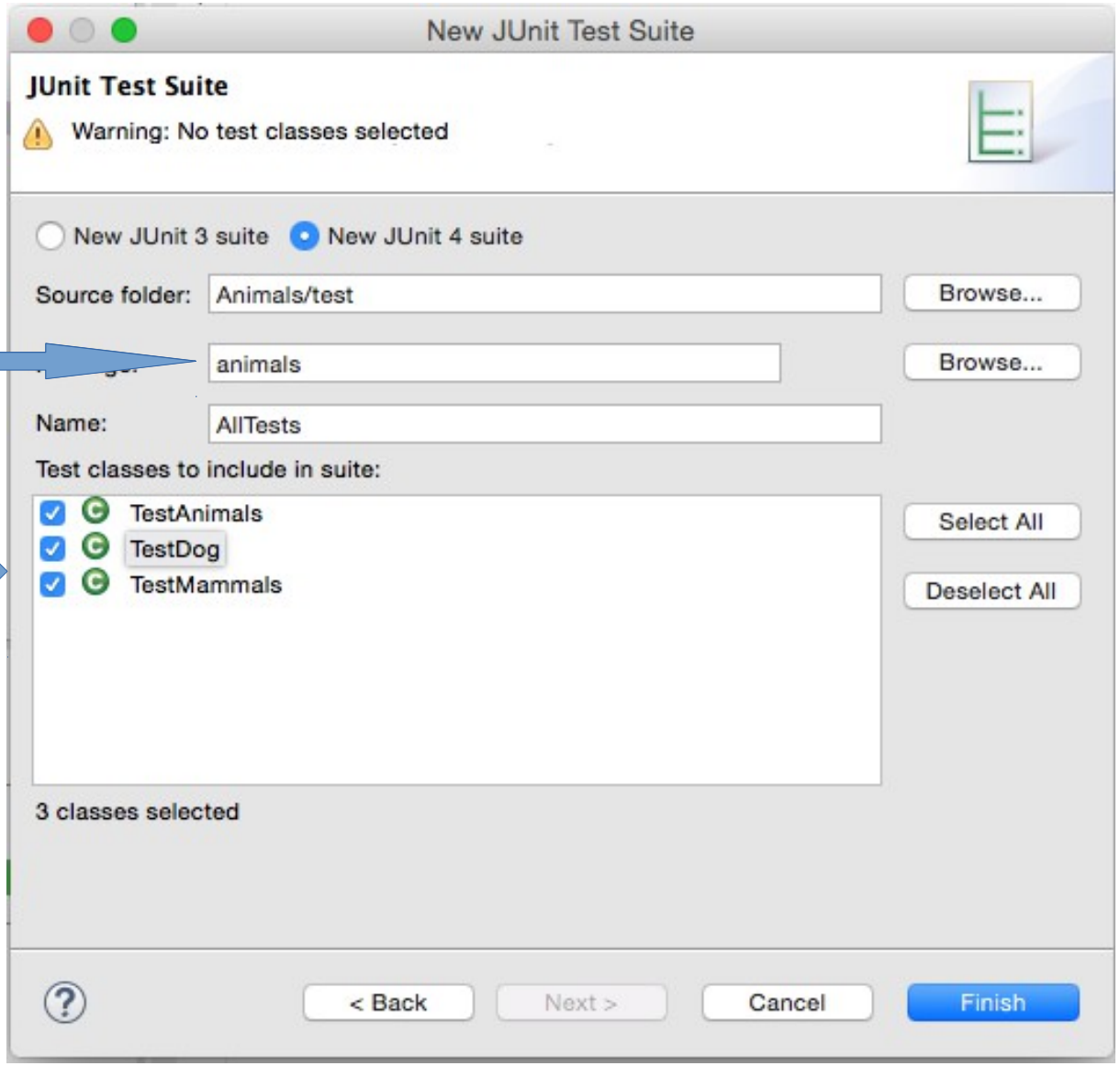
Select Wizard
Java > JUnit > JUnit Test Suite



Set Package

Package: animals

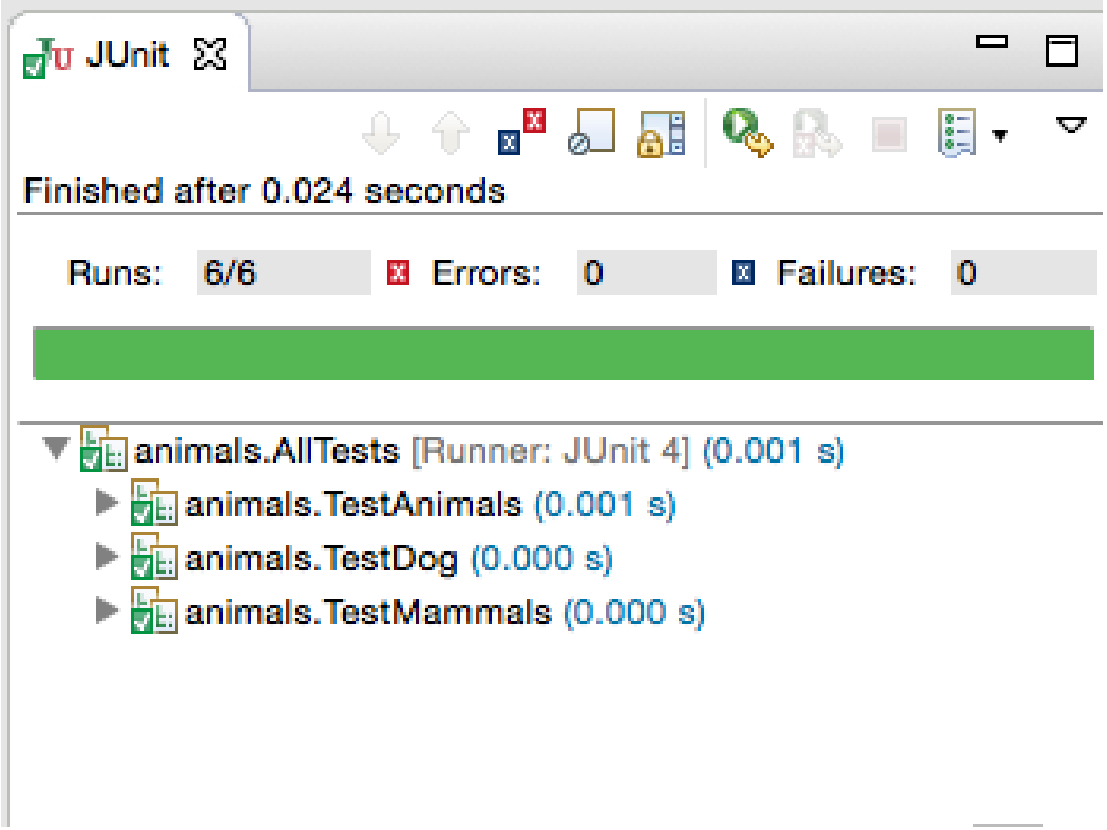
Tests in package are automatically filled in



Sets up to run all tests in package

```
package animals;

import org.junit.runner.RunWith;

@RunWith(Suite.class)
@SuiteClasses({ TestAnimals.class, TestDog.class, TestMammals.class })
public class AllTests {

}
```

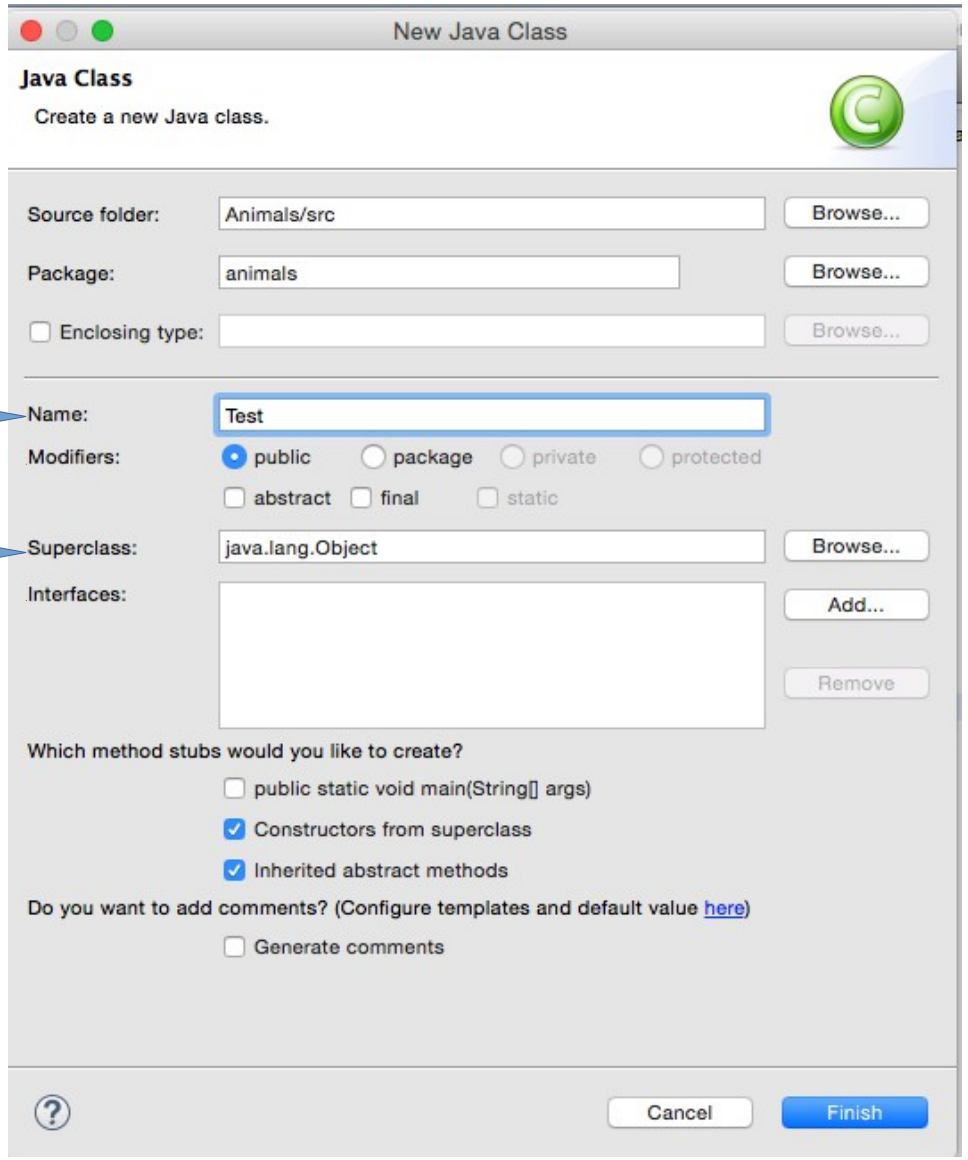
Run As JUnit Test runs all tests



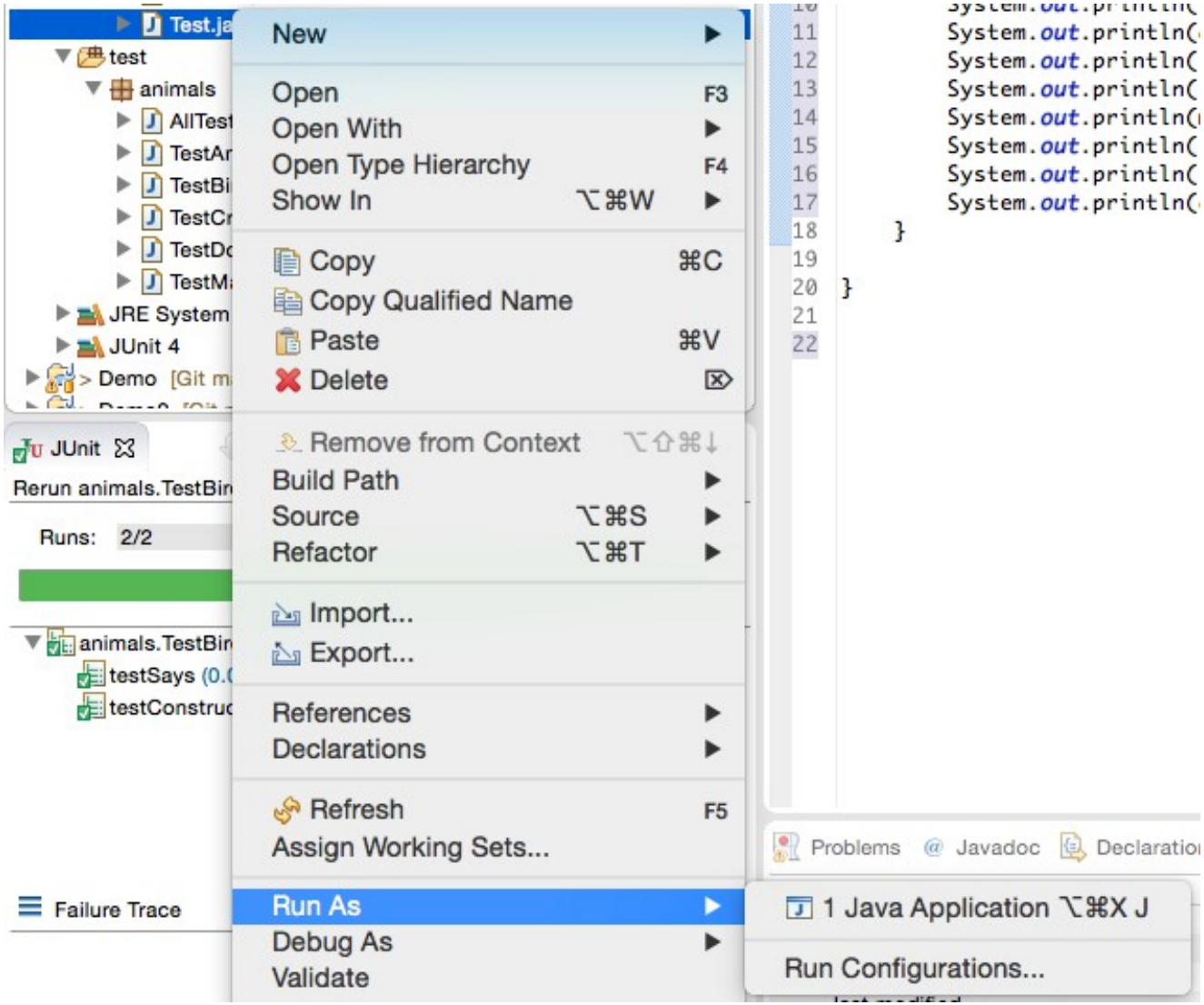
Add System Test

Name: Test

Superclass: Object



Run as Java Application



Output appears in Console

Dog.java | OldTest.java | Bird.java | TestBird.java | Test.java

```java
package animals;

public class Test {

    public static void main(String[] args) {
        Animal a = new Animal("Food");
        Mammal m = new Mammal("Milk");
        Dog d = new Dog("Meat");

        System.out.println("Animals eat " + a.getFood());
        System.out.println(a.says());
        System.out.println("Mammals eat " + m.getFood());
        System.out.println("Mammal young are " + m.getOffspring());
        System.out.println(m.says());
        System.out.println("Dogs eat " + d.getFood());
        System.out.println("Dog young are " + d.getOffspring());
        System.out.println(d.says());
    }

}
```

Console Tab

Consol | Tasks | Error L | Proper

```
<terminated> Test (1) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_05.jdk/Contents/Home/bin/java
Animals eat Food
Animals say many different things
Mammals eat Milk
Mammal young are Live
Mammals say many different things
Dogs eat Meat
Dog young are Live
Dog goes woof
```

Make TestBird before Bird

Import annotation @Before

Bird Changed to Field

@Before method runs before all tests

@Before gives each test a new Bird

@Before gives each test a new Bird

```
package animals;

import static org.junit.Assert.*;

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

public class TestBird {

    Bird b;

    @Before
    public void before() {
        b = new Bird("Food");
    }

    @Test
    public void testConstructor() {
        assertEquals(b.getFood(), "Food");
        assertEquals(b.getOffspring(), "Eggs");
    }

    @Test
    public void testSays() {
        assertEquals(b.says(), "Bird says tweet");
    }

}
```

Create Bird

Change offspring to “Eggs”

```
package animals;

public class Bird extends Animal {

    private static final String offspring = "Eggs";

    public Bird(String eats) {
        super(eats);
    }

    public String getOffspring() {
        return offspring;
    }

    @Override
    public String says() {
        return "Bird says tweet";
    }

}
```

Make TestCrow before Crow

```java
package animals;

import static org.junit.Assert.*;

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

public class TestCrow {

    Crow b;

    @Before
    public void before() {
        b = new Crow("Food");
    }

    @Test
    public void testConstructor() {
        assertEquals(b.getFood(), "Food");
        assertEquals(b.getOffspring(), "Eggs");
    }

    @Test
    public void testSays() {
        assertEquals(b.says(), "Crow goes caw");
    }

}
```

Create Crow

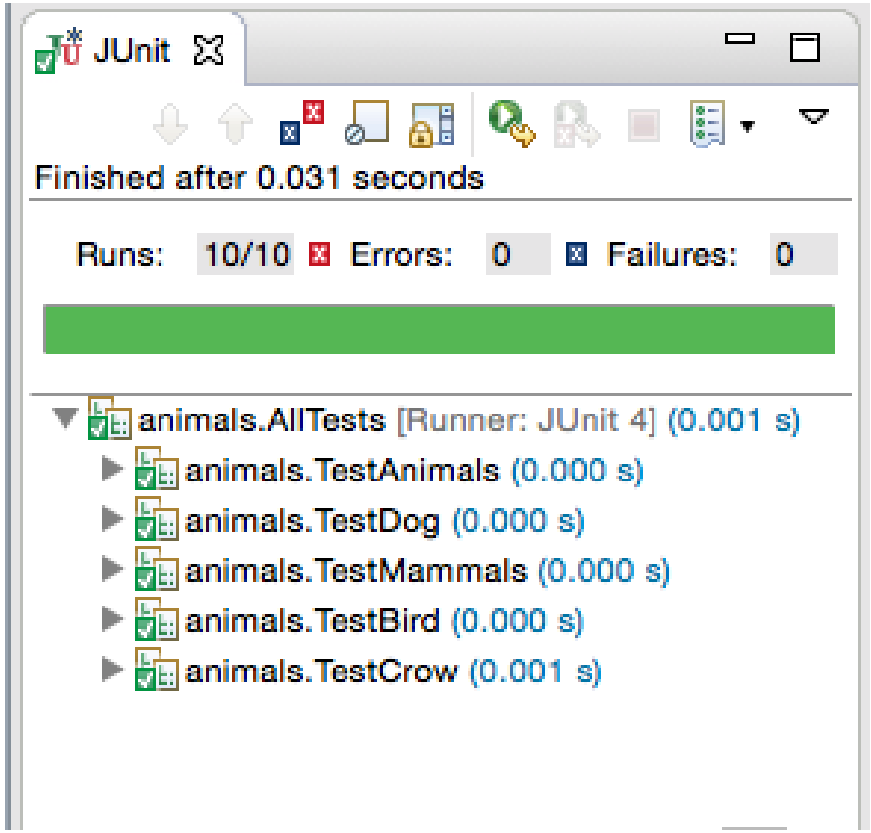
```
package animals;

public class Crow extends Bird {

    public Crow(String eats) {
        super(eats);
        // TODO Auto-generated constructor stub
    }

    @Override
    public String says() {
        return "Crow goes caw";
    }

}
```

Update AllTests

Added

```
package animals;

import org.junit.runner.RunWith;

@RunWith(Suite.class)
@SuiteClasses({ TestAnimals.class, TestDog.class, TestMammals.class,
    TestBird.class, TestCrow.class})
public class AllTests {

}
```

Run AllTests as JUnit Test



Update System Test

No offspring for Animal

Offspring defined for Mammal

Offspring inherited for Dog

Offspring defined for Bird

Offspring inherited for Crowl

```
package animals;

public class Test {

    public static void main(String[] args) {
        Animal a = new Animal("Food");
        Mammal m = new Mammal("Milk");
        Dog d = new Dog("Meat");
        Bird b = new Bird("Food");
        Crow c = new Crow("Seeds");

        System.out.println("Animals eat " + a.getFood());
        System.out.println(a.says());
        System.out.println("Mammals eat " + m.getFood());
        System.out.println("Mammal young are " + m.getOffspring());
        System.out.println(m.says());
        System.out.println("Dogs eat " + d.getFood());
        System.out.println("Dog young are " + d.getOffspring());
        System.out.println(d.says());
        System.out.println("Birds eat " + b.getFood());
        System.out.println("Bird young are " + b.getOffspring());
        System.out.println(b.says());
        System.out.println("Crows eat " + c.getFood());
        System.out.println("Crow young are " + c.getOffspring());
        System.out.println(d.says());
    }

}
```

Updated System Test Output

No offspring for Animal

Offspring defined for Mammal

Offspring inherited for Dog

Offspring defined for Bird

Offspring inherited for Crowl

```
Animals eat Food
Animals say many different things
Mammals eat Milk
Mammal young are Live
Mammals say many different things
Dogs eat Meat
Dog young are Live
Dog goes woof
Birds eat Food
Bird young are Eggs
Bird says tweet
Crows eat Seeds
Crow young are Eggs
Dog goes woof
```