Object Oriented Programming

Week 4 Part 2
ArrayList

Lecture

- Reason to use ArrayList
- Using ArrayList

Reason to use ArrayList

1-many; many-many

- We use an array to represent 1-many and many to many relationships
- However, we need to indicate how many items are in an array before we use it
- Therefore, we need to
  - Keep track of the end of the array to add element
  - Extend the array when we run out of space

ArrayList

- Fortunately, ArrayList provides a type that does everything necessary
- ArrayList is a class
  - It has methods to insert and retrieve elements
    - These methods are executed by the square bracket operator just like arrays
  - But the array expands when it needs to

Refactoring Pack to use ArrayList

Example: Pack

- Pack

Ordinary Array

Initialized in Constructor: without Wolf

To add a wolf, you need to copy the array

```
package animals;

public class Pack {

    private Wolf[] members;

    public Pack(Wolf wolves[]) {
        members = wolves;
    }

    public Pack() {
        members = new Wolf[0];
    }

    public Wolf[] getMembers() {
        return members;
    }

    public void addWolf(Wolf w) {
        Wolf[] temp = new Wolf[members.length+1];
        for (int i = 0; i < members.length; i++) {
            temp[i] = members[i];
        }
        temp[members.length] = w;
        members = temp;
    }

}
```

Refactor to use ListArray

- We do not need a new test to refactor
  - We want the behavior to remain the same
  - No new test because no new behavior
- We need to add java.util.ArrayList
  - Specifies package “java.util”
  - Specifies class to include “ArrayList”

Refactor to use ListArray: Pack

- Pack

Import ArrayList

Declare ArrayList<Wolf> field

Create ArrayList<Wolf>

Initialize ArrayList<Wolf> from array

Return array of wolves for getMember()

Add a new Wolf to ArrayList

```
package animals;

import java.util.ArrayList;
import java.util.Arrays;

public class Pack {

    private ArrayList<Wolf> members;
    private Deer hunting;

    public Pack() {
        members = new ArrayList<Wolf>();
    }

    public Pack(Wolf wolves[]) {
        members = new ArrayList<Wolf>(Arrays.asList(wolves));
    }

    public Wolf[] getMembers() {
        Wolf[] temp = new Wolf[0];
        return members.toArray(temp);
    }

    public void addWolf(Wolf w) {
        members.add(w);
    }

    public void hunts(Deer d) {
        hunting = d;
    }

    public Deer ishunting() {
        return hunting;
    }

}
```

Import Class

- “import java.util.ArrayList
  - Import causes the ArrayList class to be added to the classes the program can use
    - i.e., it adds this class to animals
- The class lives in the package java.util
  - The java.util package contains common extension to Java

Declare ArrayList variable

- The ArrayList class is a generic class
  - It takes a type in angle brackets
  - ArrayList<Wolf> can only contain wolves.
    - The compiler will complain if you try to put something else in it.
- This is an example of Parametric Polymorphism
  - We indicate what inheritance constrain as a parameter
    - Here <Wolf>
  - It is called a generic class in Java.

Initialize ArrayList<Wolf>

- As with any other variable, we need to initialize the variable before we can use it.
  - As with other variables, we initialize it in the constructor
- To initialize use ArrayList<Wolf> just as any other class
  - I.e, members = new ArrayList<Wolf>()
  - The name of the class is the name of the constructor

Supporting Existing Interface (1)

- Currently Pack takes an array of wolves as a parameter to the constructor and returns an array as the result of getMember.
- When refactoring, we do not want to change behavior at all
  - We are constrained to working in the class itself
- Changing the behavior of an existing class is a risky operation
  - It may have effect far from the class we are working on
- If we want to change the behavior of an existing class we need to know everywhere the class is used.

Supporting Existing Interface (2)

- If we want to change the behavior of an existing class we need to know everywhere the class is used.
- Here we need to translate from array to ArrayList in two places
  - We need to translate in the constructor that takes an array
    - i.e., Pack(Wolf wolves[])
  - We need to translate the getter for members
    - i.e. Wolf[] getMembers()

Initializing from an array (1)

- An ArrayList can be initialized from a Collection (an interface, which is like a class)
  - An array is not a Collection
  - We can create a List, which is a Collection, using the asList method of Arrays
  - The asList method is a static method
    - It exits as part of the Array class
    - It is called form the class name, not an object

Initializing from an array (2)

- We pass the array to the asList method to create a List, which is a Collection
  - We then use the List, that the asList method produces to create the ArrayList<Wolf>
  - We then assign the newly created ArrayList<Wolf> to members

Return array for getMembers

- ArrayList has a method to translate to an array
  - i.e., members.toArray(temp)
  - This method takes the array passed in as a parameter and fills it with the elements in the ArrayList
  - If the array is too small, it expands it. If it is too large, it puts and null at the end of the array.
    - Note that the length field of the array will not indicate the number of elements if the array is too big.
- We return the array created

Changing the interface to Pack

Changing the Interface

- Changing the interface to a class is a change in the behavior of a class
  - Before we change the behavior of a class, we must
    - Make sure we know ever place that class is being used
    - Create a test for the new behavior
- Since Pack is only used in the Test, now is the time to update the interface
  - We write a new Test that uses ArrayList

Changing the test: TestPack

- TestPack

Import ArrayList

Test second constructor

Constructor and member return ArrayList<Wolf>

AssertEquals needs to be changed to ArrayList method gets(0) from [0]

```
package animals;

import static org.junit.Assert.*;

import java.util.ArrayList;

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

public class TestPack {

    Pack p;

    @Before
    public void before() {
        p = new Pack();
    }

    @Test
    public void testConstructor() {
        assertEquals(p.getMembers().size(), 0);
    }

    public void testConstructor2() {
        Wolf temp1[] = new Wolf[0];
        Wolf temp2[] = new Wolf[0];
        ArrayList<Wolf> wolves = new ArrayList<Wolf>();
        for (int i = 0; i < 5; i++) {
            wolves.add(new Wolf("Meat"));
        }
        p = new Pack(wolves);
        assertArrayEquals(wolves.toArray(temp1), p.getMembers().toArray(temp2));
    }

    @Test
    public void testAddWolf() {
        Wolf w1 = new Wolf("Meat");
        p.addWolf(w1);
        assertEquals(p.getMembers().get(0), w1);
    }
```

Changing: Pack

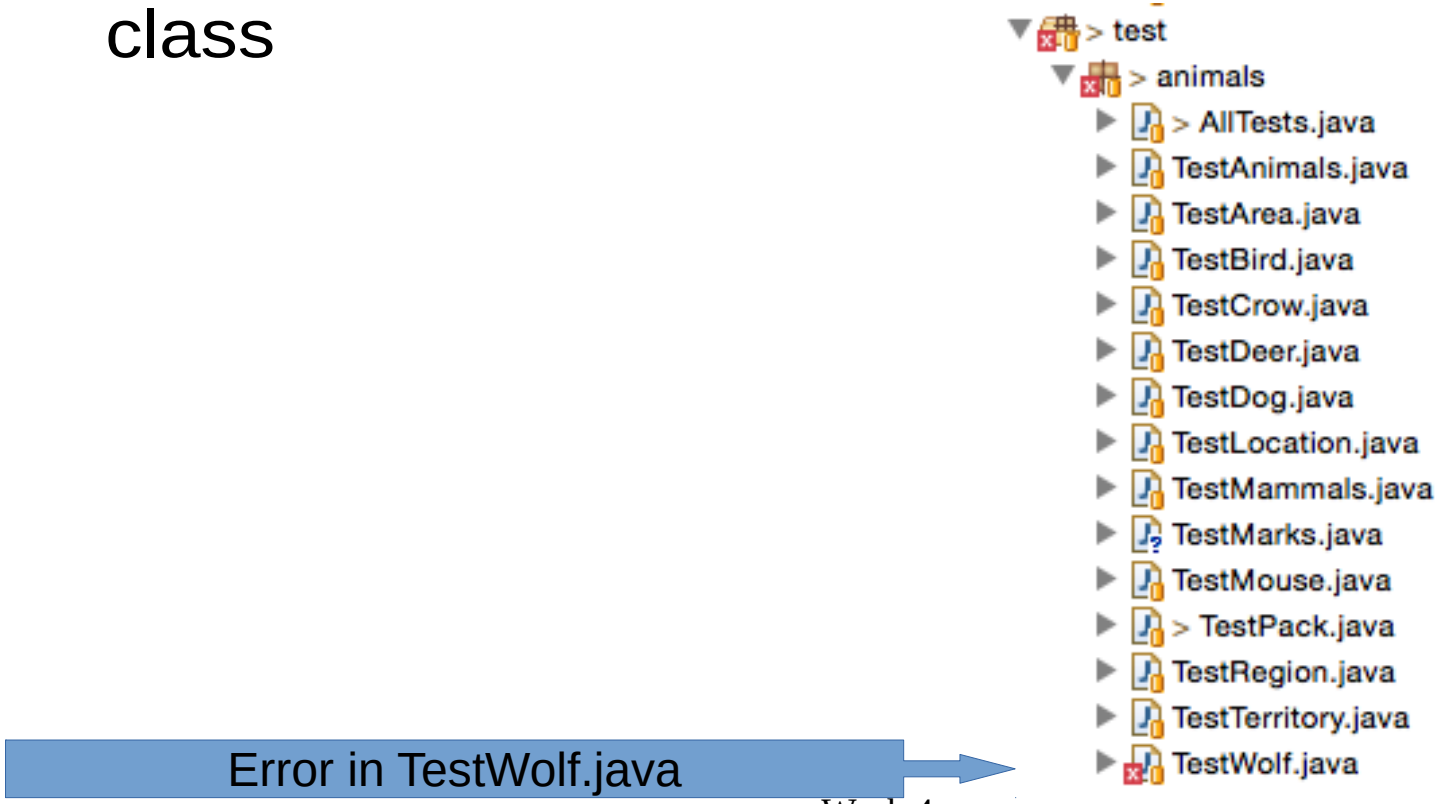
Change Constructor to accept ArrayList

Change getMembers to return ArrayList

```
package animals;

import java.util.ArrayList;
import java.util.Arrays;

public class Pack {

    private ArrayList<Wolf> members;
    private Deer hunting;

    public Pack() {
        members = new ArrayList<Wolf>();
    }

    public Pack(ArrayList<Wolf> wolves) {
        members = wolves;
    }

    public ArrayList<Wolf> getMembers() {
        return members;
    }

    public void addWolf(Wolf w) {
        members.add(w);
    }

    public void hunts(Deer d) {
        hunting = d;
    }

    public Deer ishunting() {
        return hunting;
    }

}
```

New Error Appears

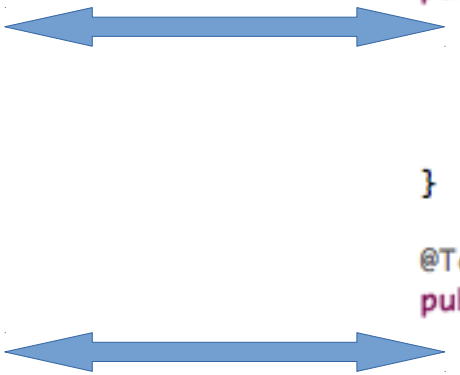
- When we change getMembers, we discover that another class TestWolf, used the class
  - This is the danger of changing the interface to a class



The Problem

- The problem is we are initializing the Pack from an array of wolves.
- We change use the zero parameter constructor

```
@Test
public void testMemberOf() {
    Pack p = new Pack(new Wolf[5]);
    w = new Wolf("Meat", p);

    assertEquals(w.getMemberOf(), p);
}

@Test
public void testSetMemberOf() {
    Pack p = new Pack(new Wolf[5]);
    w = new Wolf("Meat");
    w.setMemberOf(p);

    assertEquals(w.getMemberOf(), p);
}
```



```
@Test
public void testMemberOf() {
    Pack p = new Pack();
    w = new Wolf("Meat", p);

    assertEquals(w.getMemberOf(), p);
}

@Test
public void testSetMemberOf() {
    Pack p = new Pack();
    w = new Wolf("Meat");
    w.setMemberOf(p);

    assertEquals(w.getMemberOf(), p);
}
```

Update Area

- TestArea

```
package animals;

import static org.junit.Assert.*;

import java.util.ArrayList;

import org.junit.Test;

public class TestArea {

    @Test
    public void testConstructor() {
        Location temp1[] = new Location[0];
        Location temp2[] = new Location[0];
        ArrayList<Location> boundary = new ArrayList<Location>();
        for (int i = 0; i < 5; i++) {
            boundary.add(new Location(i + 0.7, i + 0.9));
        }
        Area a = new Area(boundary);
        assertArrayEquals(boundary.toArray(temp1),
                a.getBoundary().toArray(temp2));
    }

}
```

- Area

```
package animals;

import java.util.ArrayList;

public class Area {

    ArrayList<Location> boundary;

    public Area(ArrayList<Location> outline) {
        boundary = outline;
    }

    public ArrayList<Location> getBoundary() {
        return boundary;
    }

}
```

Update Region

- TestRegion

```
package animals;

import static org.junit.Assert.*;

import java.util.ArrayList;

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

public class TestRegion {

    Region r;
    ArrayList<Location> boundary;
    Territory territory;

    @Before
    public void before() {
        boundary = new ArrayList<Location>();
        for (int i = 0; i < 5; i++) {
            boundary.add(new Location(i * 0.7, i * 0.9));
        }
        r = new Region(boundary);
    }

    @Test
    public void testConstructor() {
        Location temp1[] = new Location[0];
        Location temp2[] = new Location[0];
        assertArrayEquals(boundary.toArray(temp1),
                r.getBoundary().toArray(temp2));
    }

    @Test
    public void testAddTerritory() {
        ArrayList<Location> tBound = new ArrayList<Location>();
        for (int i = 0; i < 5; i++) {
            tBound.add(new Location(i * 0.7, i * 0.9));
        }
        Territory t = new Territory(tBound);
        r.addTerritory(t);
        assertEquals(r.getContains()[0], t);
    }

}
```

- Region

```
package animals;

import java.util.ArrayList;

public class Region extends Area {

    private Territory[] contains;

    public Region(ArrayList<Location> outline) {
        super(outline);
        contains = new Territory[0];
    }

    public Territory[] getContains() {
        return contains;
    }

    public void addTerritory(Territory newTerritory) {
        Territory[] temp = new Territory[contains.length + 1];
        for (int i = 0; i < contains.length; i++) {
            temp[i] = contains[i];
        }
        temp[contains.length] = newTerritory;
        this.contains = temp;
    }

}
```

Update Territory

- TestTerritory

```
package animals;

import static org.junit.Assert.*;

import java.util.ArrayList;

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

public class TestTerritory {


    Territory t;
    ArrayList<Location> boundary;

    @Before
    public void before() {
        boundary = new ArrayList<Location>();
        for (int i = 0; i < 5; i++) {
            boundary.add(new Location(i * 0.7, i * 0.9));
        }
        t = new Territory(boundary);
    }

    @Test
    public void testConstructor() {
        Location temp1[] = new Location[0];
        Location temp2[] = new Location[0];
        assertArrayEquals(boundary.toArray(temp1),
                t.getBoundary().toArray(temp2));
    }

    @Test
    public void testAddRegion() {
        ArrayList<Location> rBound = new ArrayList<Location>();
        for (int i = 0; i < 5; i++) {
            rBound.add(new Location(i * 0.7, i * 0.9));
        }
        Region r = new Region(rBound);
        r.addTerritory(t);
        assertEquals(r.getContains()[0], t);
    }

}
```

- Territory

```
package animals;

import java.util.ArrayList;

public class Territory extends Area {

    private Region[] isIn;

    public Territory(ArrayList<Location> outline) {
        super(outline);
        isIn = new Region[0];
    }

    public Region[] getIsIn() {
        return isIn;
    }

    public void addRegion(Region newRegion) {
        Region[] temp = new Region[isIn.length + 1];
        for (int i = 0; i < isIn.length; i++) {
            temp[i] = isIn[i];
        }
        temp[isIn.length] = newRegion;
        this.isIn = temp;
    }

}
```

Update TestMarks

- TestMarks creates a territory from an array of locations
  - Need to change to an ArrayList

Change to TestMarks

- NewTestMarks

```
package animals;

import static org.junit.Assert.*;

import java.util.ArrayList;

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

public class TestMarks {


    ArrayList<Location> boundary;
    Territory territory;
    Wolf wolf;
    Marks m;


    @Before
    public void before() {
        boundary = new ArrayList<Location>();
        for (int i = 0; i < 5; i++) {
            boundary.add(new Location(i * 0.7, i * 0.9));
        }
        territory = new Territory(boundary);
        wolf = new Wolf("Meat");
        m = new Marks(wolf, territory);
    }

    @Test
    public void testConstructor() {
        assertEquals(m.getWolf(), wolf);
        assertEquals(m.getTerritory(), territory);
    }

    @Test
    public void testTime() {
        for (int i = 0; i < 5; i++) {
            m.setOneTime(i);
        }
        for (int i = 0; i < 5; i++) {
            assertEquals(m.getOneTime(i), i);
        }
    }

}
```

Valediction

- Ay seem like the change to ArrayList was a lot of effort, but
  - Most of the effort was just fixing syntactic errors, which are much easier to change
  - Early changes are much easier that later changes
    - If it looks like a change is going to improve the code clarity, it is usually worth doing
  - The longer changes are delayed, the longer it takes to implement them