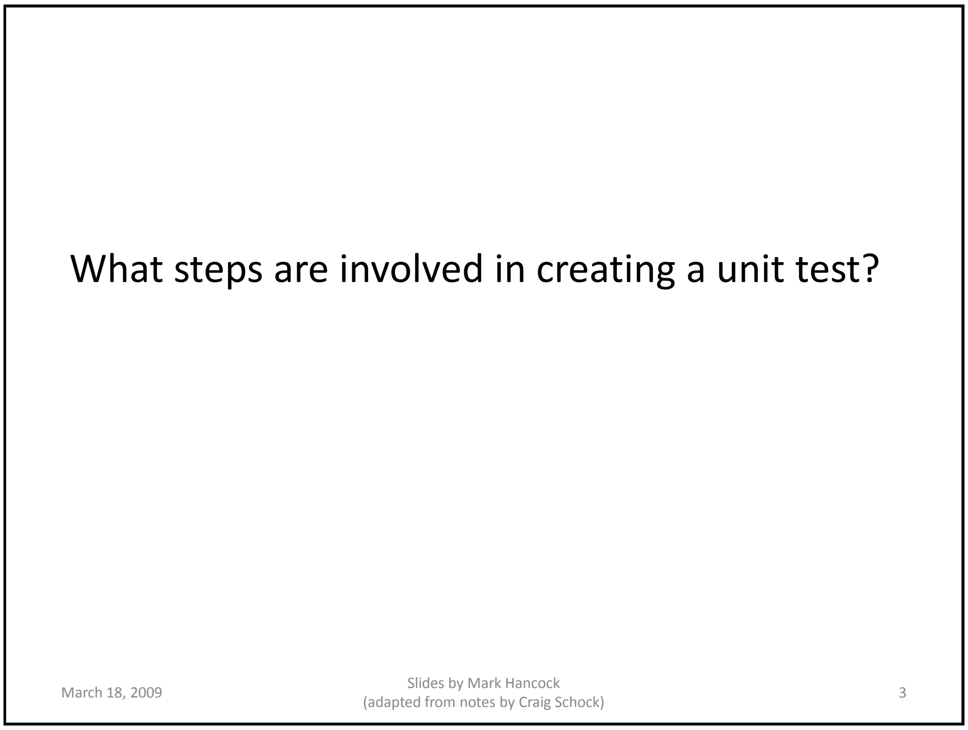
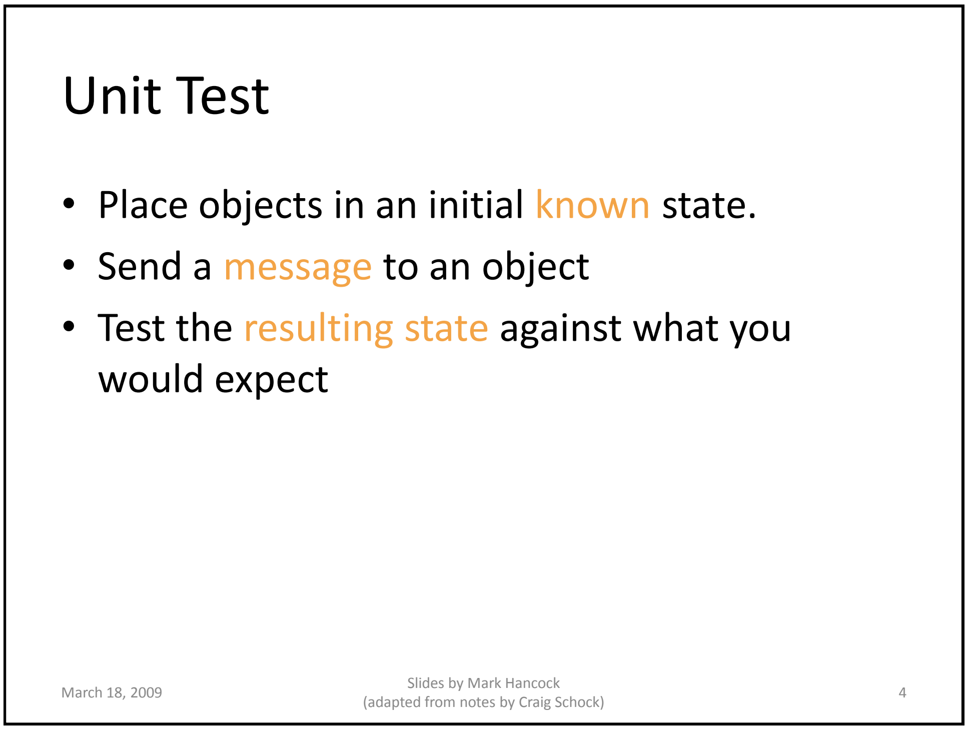
JUnit Summary

- Test methods (@Test)
- Testing exceptions
- Common known states (@Before)

By the end of this lecture you will be able to implement unit tests using JUnit in Java.



What steps are involved in creating a unit test?



Unit Test

- Place objects in an initial known state.
- Send a message to an object
- Test the resulting state against what you would expect

Example: Atom

```
public class Atom {
    private String symbol;

    private int number;

    private float weight;

    public Atom(String symbol, int number, float weight) {
        this.symbol = symbol;
        this.number = number;
        this.weight = weight;
    }

    public String getSymbol() {
        return symbol;
    }

    public float getWeight() {
        return weight;
    }

    public int getNumber() {
        return number;
    }
}
```

Exercise: Write a method that would setup the initial know state using the Atom constructor.

@Test

- To create a unit test in JUnit, just add @Test before your method

Example

```
import org.junit.*;

public class UnitTester
{
  @Test
  public void testAtomConstructor()
  {
    Atom atom = new Atom("C", 6, 12.01f);
  }
}
```

How do we know whether the test was successful?

assertTrue & assertFalse

- There are two methods in the Assert class (assertTrue and assertFalse) that test whether a boolean expression is true or false respectively.

Example

```
import org.junit.*;


public class UnitTester
{
  @Test
  public void testAtomConstructor()
  {
    Atom atom = new Atom("C", 6, 12.01f);
    Assert.assertTrue(atom.getSymbol().equals("C"));
    Assert.assertTrue(atom.getNumber() == 6);
    Assert.assertTrue(atom.getWeight() == 12.01f);
  }
}
```

Example: common shortcut

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

public class UnitTester
{
  @Test
  public void testAtomConstructor()
  {
    Atom atom = new Atom("C", 6, 12.01f);
    assertTrue(atom.getSymbol().equals("C"));
    assertTrue(atom.getNumber() == 6);
    assertTrue(atom.getWeight() == 12.01f);
  }
}
```

Running JUnit Tests

```
> javac <tester>.java
> java org.junit.runner.JUnitCore <tester>
```

Example: Molecule

```
import java.util.HashMap;
import java.util.Scanner;
import java.util.Map;

public class Molecule
{
    private HashMap<Atom,Integer> atomMap;

    public Molecule()
    {
        atomMap = new HashMap<Atom,Integer>();
    }

    public void parse(PeriodicTable table, String molecularFormula)
    {
        // ...
    }

    public float getMass()
    {
        // ...
    }
}
```

Example: PeriodicTable

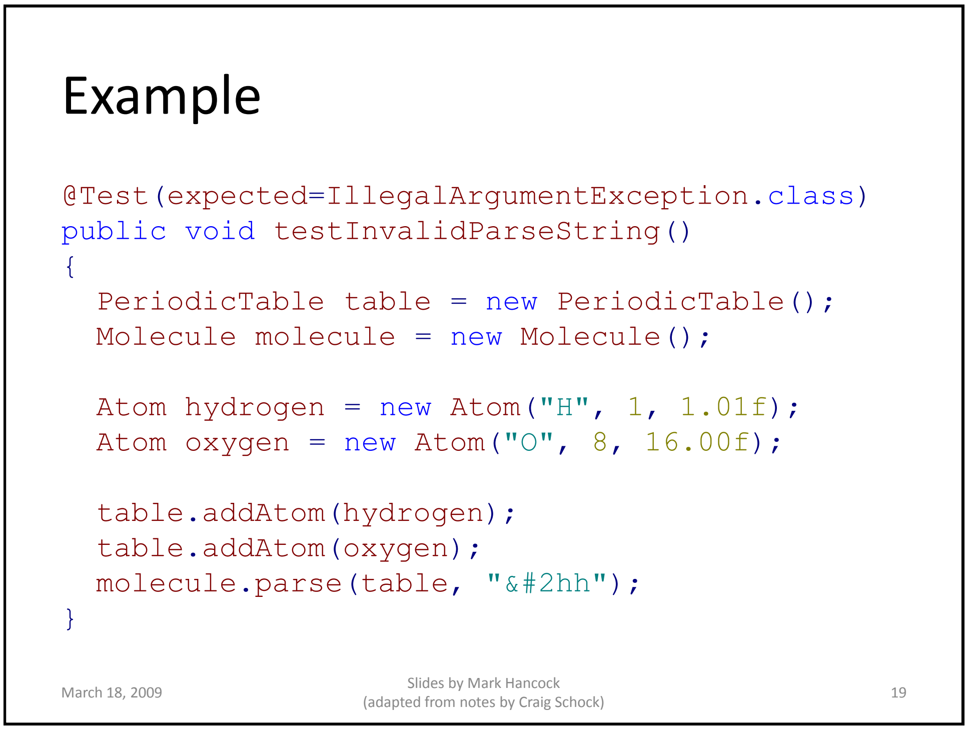
```
import java.util.ArrayList;

public class PeriodicTable
{
    private ArrayList<Atom> atoms;

    public PeriodicTable()
    {
        atoms = new ArrayList<Atom>();
    }

    public void addAtom(Atom atom)
    {
        // ...
    }

    public Atom getAtom(String symbol)
    {
        // ...
    }
}
```

Exercise: Write a method that uses JUnit to test the parse method in the Molecule class.

How would you test for incorrect input to the parse method? E.g., “&#2hh”

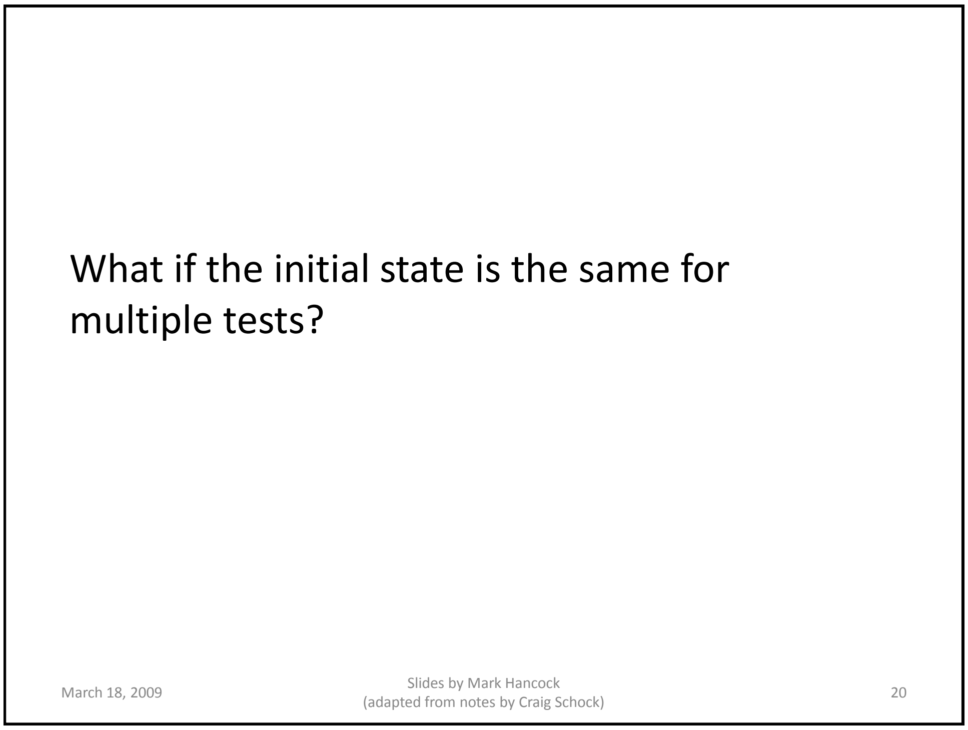
Exception Tests

- If a sequence of steps *should* result in a particular kind of exception, you can check for this.



Example

```
@Test(expected=IllegalArgumentException.class)
public void testInvalidParseString()
{
  PeriodicTable table = new PeriodicTable();
  Molecule molecule = new Molecule();

  Atom hydrogen = new Atom("H", 1, 1.01f);
  Atom oxygen = new Atom("O", 8, 16.00f);

  table.addAtom(hydrogen);
  table.addAtom(oxygen);
  molecule.parse(table, "&#2hh");
}
```



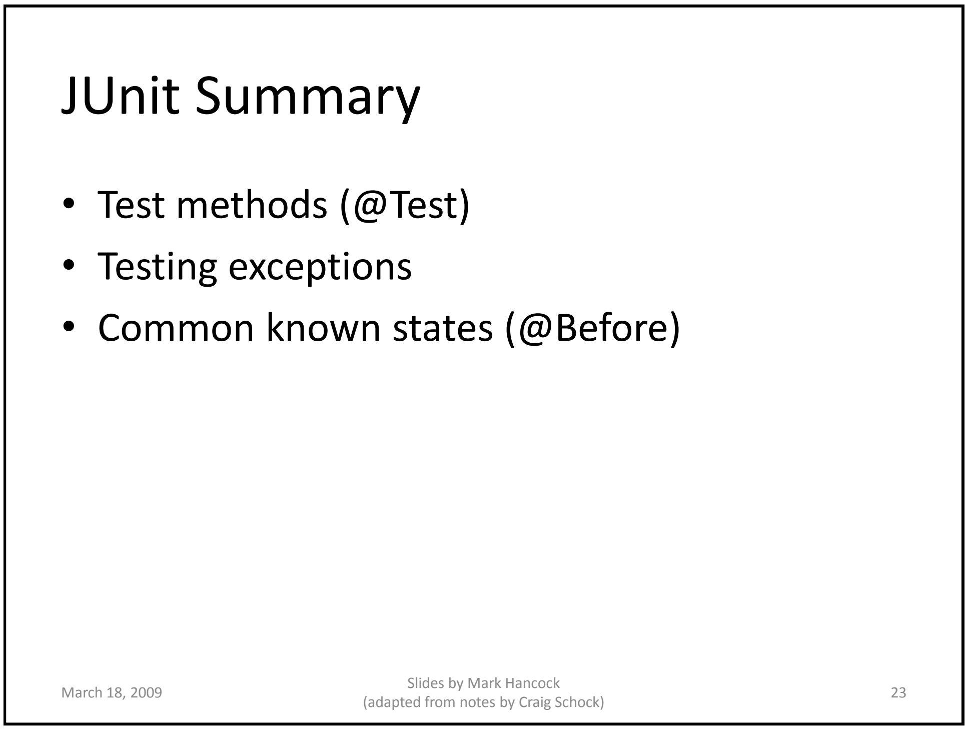
What if the initial state is the same for multiple tests?

@Before

- Any method with @Before before it will be run before the @Test methods.

Example

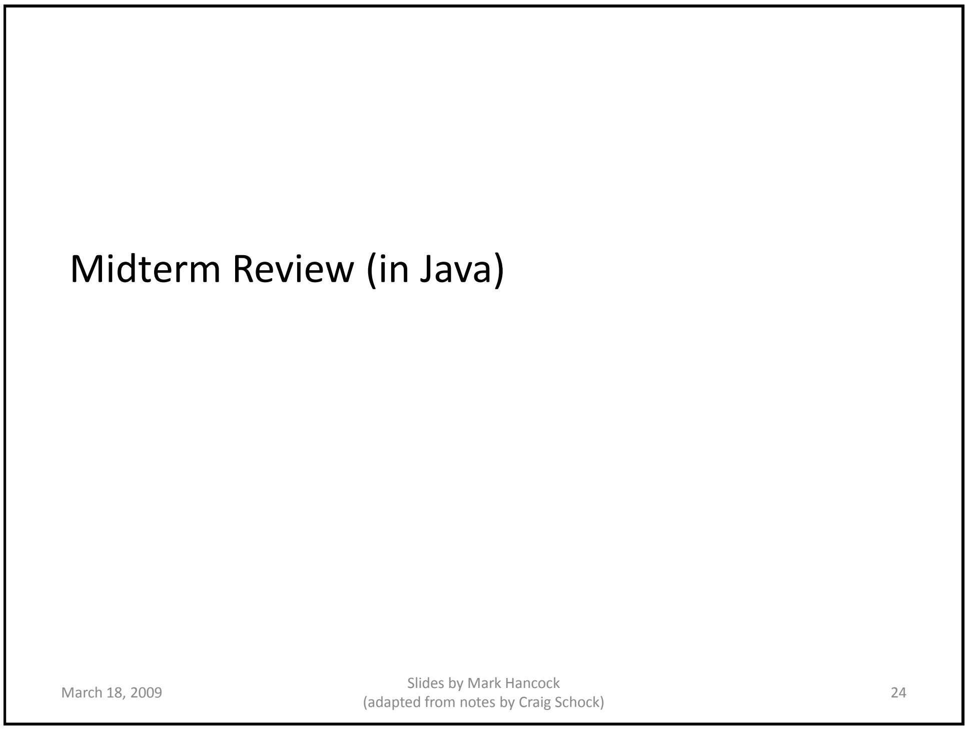
```
public class UnitTester
{
  Atom atom;
  Atom hydrogen;
  Atom oxygen;

  PeriodicTable table;
  Molecule molecule;

  @Before
  public void initialize()
  {
    atom = new Atom("C", 6, 12.01f);
    hydrogen = new Atom("H", 1, 1.01f);
    oxygen = new Atom("O", 8, 16.00f);

    table = new PeriodicTable();
    molecule = new Molecule();

    table.addAtom(hydrogen);
    table.addAtom(oxygen);
  }

  // ...
```



JUnit Summary

- Test methods (@Test)
- Testing exceptions
- Common known states (@Before)

March 18, 2009 Slides by Mark Hancock (adapted from notes by Craig Schock) 23



Midterm Review (in Java)

March 18, 2009 Slides by Mark Hancock (adapted from notes by Craig Schock) 24

Question #1

- (15 %) Identify 5 differences in syntax between python and Java. Describe each with at least one sentence and provide an example which shows both the Java and Python versions.

Question #2

- (5 %) Why is it not necessary to indent your code in Java?

Question #3

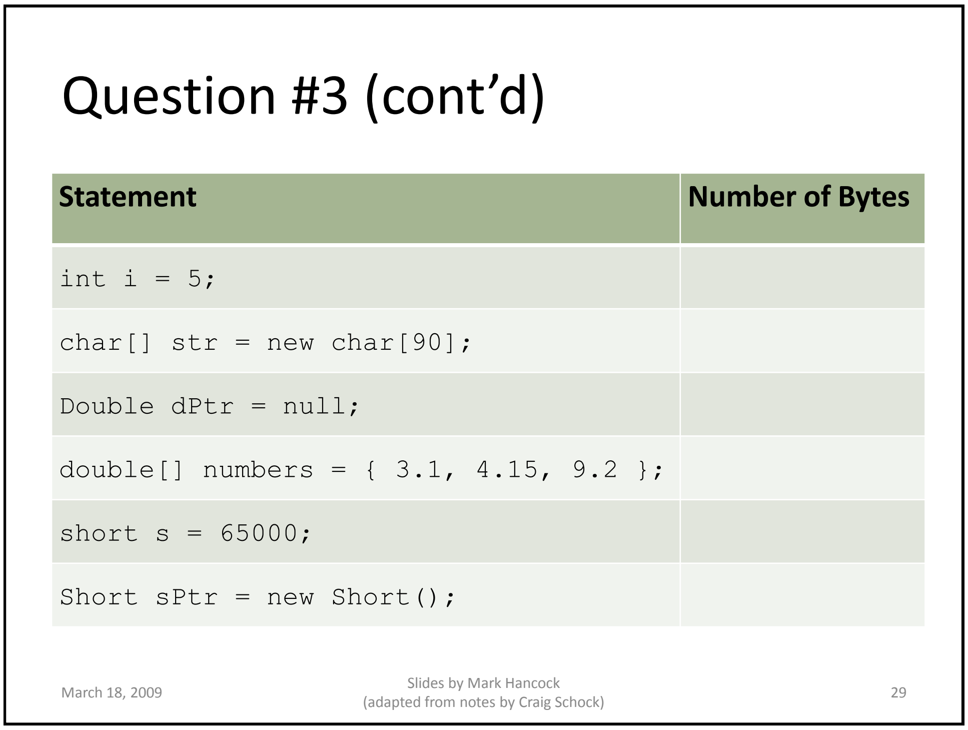
(10 %) Consider the following program and its output:

```
public class SizeProgram
{
  public static void main()
  {
    System.out.println("char:\t" + Character.SIZE / 8);
    System.out.println("int:\t" + Integer.SIZE / 8);
    System.out.println("double:\t" + Double.SIZE / 8);
    System.out.println("short:\t" + Short.SIZE / 8);
    System.out.println("Object:\t" + 4);
  }
}
```

Question #3 (cont'd)

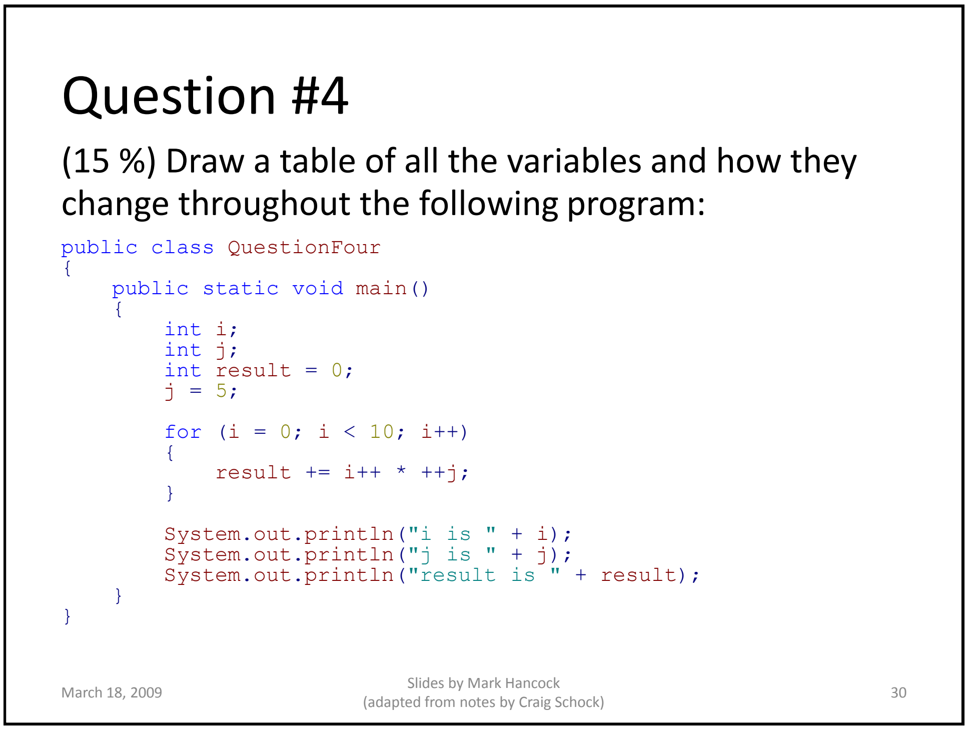
Output:

```
char:   1
int:    4
double: 8
short:  2
Object: 4
```



Question #3 (cont'd)

Statement	Number of Bytes
`int i = 5;`	
`char[] str = new char[90];`	
`Double dPtr = null;`	
`double[] numbers = { 3.1, 4.15, 9.2 };`	
`short s = 65000;`	
`Short sPtr = new Short();`	



Question #4

(15 %) Draw a table of all the variables and how they change throughout the following program:

```
public class QuestionFour
{
    public static void main()
    {
        int i;
        int j;
        int result = 0;
        j = 5;

        for (i = 0; i < 10; i++)
        {
            result += i++ * ++j;
        }

        System.out.println("i is " + i);
        System.out.println("j is " + j);
        System.out.println("result is " + result);
    }
}
```

Question #5

(15 %) Complete the following method.

```
public class QuestionFive
{
/*
 * Method: insert
 * Purpose: This function inserts an element into an
 *          array of integers.
 *
 * Parameters:
 *   array - the array of integers
 *   size - the number of elements currently in the array
 *   elem - the element to insert into the array
 *   i - the index of the element to insert the new element
 *       before in the array.
 *
 * Precondition:
 *   'size' is less than the maximum size of the array.
 *
 * Postcondition:
 *   The 'array' should be unchanged from 0 to i - 1 and all
 *   of the elements from i upward should be shifted to the
 *   right. Element i should now be 'elem'.
 */
```

Question #6

(10 %) In object-oriented programming, we introduced the concept of *mutability*. Is the following structure in Java mutable or immutable? Why?

```
public class Location
{
  public int longitude;
  public int latitude;
};
```

Question #7

- (10 %) What are the two main components that make up an abstract data type (ADT)?

Question #8

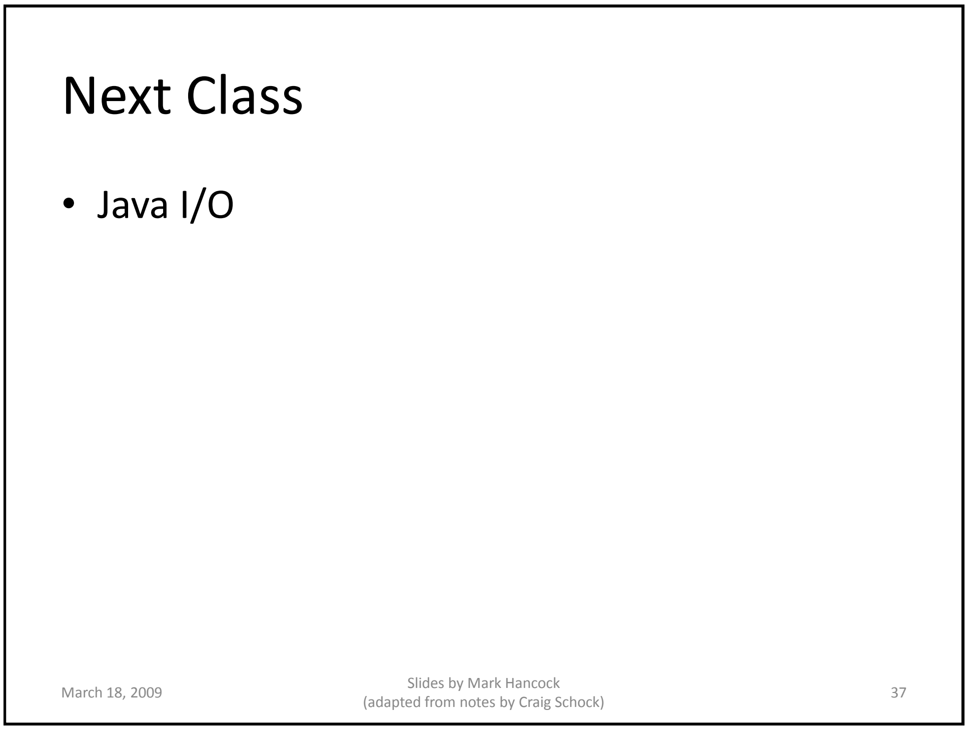
(20 %) Consider the following program. Draw a diagram of memory for when the program reaches the marked point in the code (including the stack, heap, and all global variables). Make sure each variable on the stack and in the global variable space is labeled and that the value of each variable is specified (when known). You may use curly braces ({) to name a group of variables, arrows (→) to represent references, and question marks (?) to represent uninitialized data. Also label the part of memory associated with each function.

Question #8 (Point.java)

```
public class Point
{
    public int x;
    public int y;

    public Point() {}

    public Point(int x, int y)
    {
        this.x = x;
        this.y = y;
    }
}
```

Question #8 (QuestionEight.java)

```
public class QuestionEight
{
    public static final Point g = new Point( 5, 3 );

    public static Point addPoints(Point p1, Point p2)
    {
        Point result = new Point();
        result.x = p1.x + p2.x;
        result.y = p1.y + p2.y;
Here
        return result;
    }

    public static void main(String[] args)
    {
        Point a;
        Point b;

        a = new Point();
        a.x = 4;
        a.y = 5;

        b = add_points( g, a );
    }
}
```



Next Class

- Java I/O

March 18, 2009

Slides by Mark Hancock
(adapted from notes by Craig Schock)

37