You have 110 minutes to finish this test. Your exam should contain 6 problems (numbered 0-5). This is an open-book test. You may consult any books, notes, or other paper-based inanimate objects available to you. Read the problems carefully. If you find it hard to understand a problem, please ask a question. Please write your answers in the spaces provided in the test; if you need to use the back of a page make sure to clearly tell us so on the front of the page.

Good Luck!

<table>
<thead>
<tr>
<th></th>
<th>out of</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1 point</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2 points</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>6 points</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3 points</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>3 points</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>9 points</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>24 points</td>
<td></td>
</tr>
</tbody>
</table>

Write your name on the last page
Question 1 (2 points)
Below is a class that creates a geometric shape. Draw a picture of what memory looks like at the spot indicated in the main method. Note: The Point class has instance variables x and y of type int.

```java
import java.awt.Point;

public class GeometricShape {
    private Point[] vertices;
    private int sides;

    public GeometricShape(int[] xValues, int[] yValues) {
        this.sides = xValues.length;
        this.vertices = new Point[this.sides];
        for (int i = 0; i < this.sides; i++) {
            this.vertices[i] = new Point(xValues[i], yValues[i]);
        }
    }

    public static void main(String[] args) {
        int[] myXs = {0, 2, 2, 0};
        int[] myYs = {0, 0, 5, 5};
        GeometricShape shape = new GeometricShape(myXs, myYs);

        // DRAW A PICTURE OF WHAT MEMORY LOOKS LIKE HERE!!!
    }
}
```

CORRECT ANSWER:
**RUBRIC:**

<table>
<thead>
<tr>
<th>CODE</th>
<th>Deduction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>-1</td>
<td>Not showing the integers within boxes (there is a maximum of 1 point taken off no matter how many times this mistake is made.)</td>
</tr>
<tr>
<td>B</td>
<td>-0.5</td>
<td>Leaving out the sides instance variable inside of the geometric shape object</td>
</tr>
<tr>
<td>C</td>
<td>-1</td>
<td>The last two elements actually point to objects.</td>
</tr>
<tr>
<td>D</td>
<td>-1</td>
<td>The shape array only has 4 elements.</td>
</tr>
<tr>
<td>E</td>
<td>-1</td>
<td>If the integer values are not in the correct order or are generally wrong.</td>
</tr>
<tr>
<td>F</td>
<td>-1</td>
<td>If shape points directly to the array of Points, instead of to an object with instance variables vertices and sides.</td>
</tr>
<tr>
<td>G</td>
<td>-1</td>
<td>If the GeometricShape object has instance variables xValues and yValues</td>
</tr>
<tr>
<td>H</td>
<td>-0.5</td>
<td>Not drawing the X’s in the last two spots in the array vertices.</td>
</tr>
<tr>
<td>I</td>
<td>-1</td>
<td>Sides instance variable is null</td>
</tr>
<tr>
<td>J</td>
<td>-0.5</td>
<td>No arrow from shape to geometricShape object</td>
</tr>
</tbody>
</table>

Things we didn’t take off points for:

- The Point objects do not need to be labeled “Point”.
- Drawing all arrays as an object that then points to the thing that we typically draw arrays as.
- Putting something inside the array like (0, 2) is fine for each of the points. The array should really have an arrow that points to a separate object but we weren’t picky.
- Geometric shape object does not need to be labeled “Geometric shape”
- Constructor stack frame drawn

**Question 2 (6 points)**

The next three problems deal with the class `Set` from lab and shown below.

Taking advantage of inheritance, define a class `ExpandableSet` that behaves just like the `Set` except that, when `insert` is called with a value to be inserted larger than the `Set` can currently hold, the `Set` doubles in size until the value can be added. `ExpandableSet` should have a no argument constructor that makes the initial size of the `Set` be 1.

```java
public class Set {
    // Represent a set of nonnegative ints from 0 to maxElement-1
    // for some initially specified maxElement.
    // contains[k] is true if k is in this set, false if it isn't
    protected boolean[] contains;

    // Initialize a set of ints from 0 to maxElement-1.
    public Set(int maxElement) {
        contains = new boolean[maxElement];
    }
    public void insert(int k) {
```
contains[k] = true;
}
public void remove(int k) {
    contains[k] = false;
}
public boolean member(int k) {
    return contains[k];
}

/*
 * JUnit has the following relevant methods (for Questions 3 and 4)
 */
static void assertTrue(boolean b)
static void assertFalse(boolean b)
static void assertEquals(String message, Object expected, Object actual)
static void fail()

Please write your solution to Question 2 here:

CORRECT ANSWER (multiple correct solutions possible):

```java
public class ExpandableSet extends Set{

    public ExpandableSet() {
        super(1);
    }
    public void insert(int k){
        if (k >= contains.length)
        {
            int origSize = contains.length;
            int newSize = origSize * 2;
            while (newSize <= k) {
                newSize = newSize * 2;
            }
            boolean[] biggerVersion = new boolean[newSize];
            for (int i = 0; i < origSize; i++) {
                biggerVersion[i] = contains[i];
            }
            contains = biggerVersion;
        }
        super.insert(k);
    }
}
```
if (k >= contains.length)
{
    boolean[] biggerVersion = new boolean[contains.length * 2];
    for (int i = 0; i < contains.length; i++) {
        biggerVersion[i] = contains[i];
    }
    contains = biggerVersion;
    insert(k);
}
super.insert(k);

RUBRIC:

<table>
<thead>
<tr>
<th>CODE</th>
<th>Deduction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>-1</td>
<td>Not including the line public class ExpandableSet extends Set{</td>
</tr>
<tr>
<td>B</td>
<td>-1</td>
<td>Not including a constructor that calls super(1); We take off this point for explicitly saying this.contains = new boolean[1]; b/c the compiler forces you to call super in this case b/c there is not a no argument constructor provided.</td>
</tr>
<tr>
<td>C</td>
<td>-1</td>
<td>Not including the insert method header (written exactly as: public void insert(int k))</td>
</tr>
<tr>
<td>D</td>
<td>-0.5</td>
<td>Not calling super.insert(k) when k &lt; contains.length (assuming that they do something like contains[k] = true;</td>
</tr>
<tr>
<td>E</td>
<td>-1</td>
<td>Forgetting to actually insert the value into the array</td>
</tr>
<tr>
<td>F</td>
<td>-1</td>
<td>Off by one error in checking k and contains.length. (If k == contains.length they need to expand).</td>
</tr>
<tr>
<td>G</td>
<td>-2</td>
<td>Writing to contains.length (which is not modifiable). For example: contains.length = contains.length * 2;</td>
</tr>
</tbody>
</table>
| H    | -3        | They don’t copy values from the old array into the new array. For example if they don’t include something like this: for (int i = 0; i < contains.length; i++) {
                biggerVersion[i] = contains[i];
            }
            OR if they write over contains before they have copied the values. For example.
            contains = new boolean[contains.length * 2]; |
| I    | -2        | Tries to write to this. |
| J    | -2        | Don’t point contains to the newly constructed array |
| K    | -3        | Doesn’t correctly increase the size by a factor of 2. |
| L    | -2        | They don’t handle the case where the array needs to expand to more than 2 times the size. In the first example solution this is done with the following loop:
            while (newSize <= k) {
                newSize = newSize * 2;
            }
            In the second example this is done by recursively calling insert. |
| M    | -3        | Tries to implicitly write over this by calling super within the insert method, this sometimes overlaps with H. Don’t double count., calls this() on insert |
| N    | -1        | Provided additional unnecessary methods |
| P    | -3        | Treating an expandable set as an array – type mismatch issues etc. |
Question 3 (3 points)

Now we are going to test ExpandableSet using JUnit. When we change the length of ExpandableSet, we want it to become exactly the length we expect. For example, if the ExpandableSet should have doubled in length we want it to be exactly double the length, no more, no less.

a) However in the JUnit file we can’t just check the length of the array. Explain why.

Example 1 point answer:
The code is not inside the Set class, nor a subclass of the Set class so we can’t access the protected variable `contains`.

Example 0.5 point answer:
The variable is protected so we can’t access it.

Example 0 point answer: Not mentioning protected — for example:
The length can change so we can’t access it.
JUnit can’t work with arrays.

- Note: Saying that there are no getter methods for the length does not increase or decrease a score on this question.

b) Complete the helper method below to appear in your JUnit file ExpandableSetTest. This method will return true if the length of the ExpandableSet `es` is equal to the int `expectedSize`. You may not write any additional methods. Hint: It will be helpful to use the concept of Exceptions.

```java
public class ExpandableSetTest extends TestCase {
    private boolean sizeCheckHelper(ExpandableSet es, int expectedSize){
        // CORRECT ANSWER:
    }
```
try {
    es.member(expectedSize - 1)
} catch (ArrayOutOfBoundsException e) {
    return false;
}

try {
    es.member(expectedSize)
    return false;
} catch (ArrayOutOfBoundsException e) {
    return true;
}

• 1 point for checking that it is big enough,
• 1 point for checking that it isn’t too big.

Errors:
A - you only check if it is big enough
B - you only check if it is too big.
1. No points off if the name of the exception isn’t exactly correct but the idea is correct
2. 0/2 if missing the idea that calling es.member(x) might throw an exception
3. 0/2 if doing something like es.length
4. No points off if es.contains(x) is called instead of es.member(x)
5. 0.5 off if the logic is sound but the wrong exception is caught (Null Pointer)
6. -1 for using insert instead of member

Question 4 (3 points)
Fill in the table below with a convincing set of JUnit test methods for the insert method in ExpandableSet. You should test 1 important thing about insert in each test method. You may use your method sizeCheckHelper.

<table>
<thead>
<tr>
<th>public class ExpandableSetTest extends TestCase {</th>
<th>Description of what this tests</th>
</tr>
</thead>
<tbody>
<tr>
<td>public void test1() {</td>
<td></td>
</tr>
<tr>
<td>}</td>
<td></td>
</tr>
<tr>
<td>public void test2() {</td>
<td></td>
</tr>
<tr>
<td>}</td>
<td></td>
</tr>
<tr>
<td>public void test3() {</td>
<td></td>
</tr>
<tr>
<td>}</td>
<td></td>
</tr>
</tbody>
</table>

Solution
// NONE
// Test that the size doesn’t increase unnecessarily
ExpandableSet e = new ExpandableSet();
e.insert(0);
assertTrue (sizeCheckHelper(e, 1));

// ONE
// Test that the size doubles when necessary
ExpandableSet e = new ExpandableSet();
e.insert(1);
assertTrue (sizeCheckHelper(e, 2));

// MANY
// Test that the size is 8 times as large when necessary.
ExpandableSet e = new ExpandableSet();
e.insert(4);
assertTrue (sizeCheckHelper(e, 8));

RUBRIC:

<table>
<thead>
<tr>
<th>CODE</th>
<th>Deduction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>+1</td>
<td>Provides no expansion case</td>
</tr>
<tr>
<td>B</td>
<td>+1</td>
<td>Provides single expansion case</td>
</tr>
<tr>
<td>C</td>
<td>+1</td>
<td>Provides multiple expansion case</td>
</tr>
<tr>
<td>D</td>
<td>+1</td>
<td>Provides a check that the value actually gets set by insert</td>
</tr>
<tr>
<td>E</td>
<td>-1</td>
<td>Doesn’t create the objects needed, for example – contains no calls to new ExpandableSet();</td>
</tr>
<tr>
<td>F</td>
<td>-1</td>
<td>If they used print statements instead of using JUnit asserts.</td>
</tr>
<tr>
<td>G</td>
<td>-0.5</td>
<td>SizeCheck helper with incorrect arguments</td>
</tr>
<tr>
<td>H</td>
<td>-0.5</td>
<td>Contains[k] check (where k is an incorrect position)</td>
</tr>
<tr>
<td>I</td>
<td>-1</td>
<td>Not using sizecheckHelper and using something that doesn’t work</td>
</tr>
<tr>
<td>J</td>
<td>-1</td>
<td>Treating expandable set as an array</td>
</tr>
</tbody>
</table>

- We didn’t take off points if people created a constructor that took in a size and used it in their tests.

Question 5 (9 points)

This question is based upon Project 1. Relevant details of the Picture class and Pixel class are provided on the next few pages. The goal is to iterate over all Pixels in a Picture that are within a Circle. The order in which Pixels are returned doesn’t matter, but all Pixels within the circle should be returned. For example, we want to be able to run the following method in the Picture class:

    // Colors the picture with a black circle of radius 10,
    // centered at (50, 50)
Based upon the comments below, implement the following methods for the `Picture` class. You may only add code within the methods defined below. Any other modifications will result in a score of 0 on this question.

```java
public class Picture extends SimplePicture {
    // instance variables - You may NOT define any new instance variables
    int midX;        // the X coordinate of the center of the circle
    int midY;        // the Y coordinate of the center of the circle
    int radius;      // the radius of the circle
    Pixel nextPixel; // Invariant: always references the next Pixel to
                     // be returned by next() or null if no
                     // pixels remain.

    // Returns true if there are more pixels to return in the circle
    // and false otherwise
    public boolean hasNext() {
        // if nextPixel is null, there are no more pixels to return
        return nextPixel != null;
    }

    // HELPER METHOD
    // Returns true if the instance variable nextPixel is within the
    // the circle and false otherwise.
    private boolean isNextPixelWithinRadius() {
        if (nextPixel == null) {
            return false;
        }
        int x = nextPixel.getX();
        int y = nextPixel.getY();
        return (Pixel.distance(x, y, midX, midY) < radius);
    }

    // This method should restore the invariant described for nextPixel.
    // You must call this method at least once in your code on the next page.
    // This method may call the optional helper method getUnexploredPixel
    private void restoreInvariant() {
    }
}
```

// OPTIONAL HELPER METHOD
// If you chose to implement this method you must implement exactly what
public void initIterator(int centerX, int centerY, int circleRadius) {
  // the comment specifies.
  //
  // This method should return the next unexplored Pixel based upon the
  // argument lastPixel. Unexplored Pixels should be returned regardless of
  // whether they are within the circle. This method does not modify
  // nextPixel. The method returns the next Pixel to be explored or null
  // if no unexplored Pixels remain.
  public Pixel getUnexploredPixel(Pixel lastPixel) {
  }

  CODE | Deduction | Description
  ----|-----------|------------------
  A   | -4        | Missing pixels in the traversal (major)
  B   | -2        | Bad math indexing picture, for example by accessing a pixel that may be outside of
       |            | the circle (getPixel(x-radius, y-radius) etc.
  C   | -1        | Off by 1 in accessing all of the pixels
  D   | -2        | Really pretty close to all of the pixels but more than an off by 1 error

  CODE | Deduction | Description
  ----|-----------|------------------
  E   | -4        | Invariant not maintained
  F   | -2        | Restore invariant does not restore the invariant
  G   | -3        | Wrong invariant

  CODE | Deduction | Description
  ----|-----------|------------------
  I   | -1        | Calls new Pixel(...) instead of getPixel(...) 
  J   | -1        | Next() doesn’t return a pixel/or calls restore invariant after the return.
  K   | -2        | Not constructing a pixel
  L   | -1        | Makes a new picture object

  CODE | Deduction | Description
  ----|-----------|------------------
  Y   | -1        | Doesn’t save instance variables midX, midY and radius
  Z   | -1        | Doesn’t make sure that the invariant is true at the end of the call to initIterator,
       |            | examples of this include not initializing the pixel, another is calling restore invariant
       |            | that may cause the initial pixel (0,0) to never be returned by next.

  // The circle is defined by a radius circleRadius and is centered at
  // coordinates (centerX, centerY).
  // This method will be called before the first call to hasNext() or next()
  SOLUTION
  public void initIterator(int centerX, int centerY, int circleRadius) {
    midX = centerX;
    midY = centerY;
    radius = circleRadius;
    nextPixel = this.getPixel(0, 0);
    if (!nextPixelWithinRadius()){
      restoreInvariant();
  }
RUBRIC:

<table>
<thead>
<tr>
<th>CODE</th>
<th>Deduction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td>Doesn’t save instance variables midX, midY and radius</td>
</tr>
<tr>
<td>B</td>
<td></td>
<td>Doesn’t make sure that the invariant is true at the end of the call to initIterator.</td>
</tr>
</tbody>
</table>

// return successive Pixels within the circle specified in initIterator

SOLUTION:

```java
public Pixel next() {
    Pixel pixelToReturn = nextPixel;
    restoreInvariant();
    return pixelToReturn;
}
```