This exam is closed-book, closed-notes, no electronic devices allowed. The exception is the “sage page” on which you may have notes to consult during the exam. Answer questions on the pages of the exam. Do not unstaple the pages of this exam, nor should you attach any other pages to the exam. You are welcome to use the blank space of the exam for any scratch work.

Your work must be legible. Work that is difficult to read will receive no credit. Do not dwell over punctuation or exact syntax in code; however, be sure to indent your code to show its structure.

You must sign the pledge below for your exam to count. The penalty for cheating will be decided during academic integrity review, but the instructors will recommend an F in this course as the minimum penalty. If you need to leave the room for any reason prior to turning in your exam, you must give your exam and any electronic devices with a proctor.

You must fill in your identifying information correctly. Failure to do so is grounds for a zero on this exam. When you reach this point in the instructions, please give the instructor or one of the proctors a meaningful glance.

Print clearly the following information:

<table>
<thead>
<tr>
<th>Name (print clearly):</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Student 6-digit ID (one digit per box): [   ] [   ] [   ] [   ]</th>
</tr>
</thead>
</table>

Your answers below tell us where to return your graded exam.

<table>
<thead>
<tr>
<th>What time do you arrive in studio/lab? (circle one) 11:30 1:00 2:30 4:00</th>
</tr>
</thead>
<tbody>
<tr>
<td>11:30</td>
</tr>
<tr>
<td>1:00</td>
</tr>
<tr>
<td>2:30</td>
</tr>
<tr>
<td>4:00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Which Urbauer lab? (your best guess, circle one) 210 214 216 218</th>
</tr>
</thead>
<tbody>
<tr>
<td>210</td>
</tr>
<tr>
<td>214</td>
</tr>
<tr>
<td>216</td>
</tr>
<tr>
<td>218</td>
</tr>
</tbody>
</table>

Pledge: On my honor, I have neither given nor received any unauthorized aid on this exam.

Signed: ____________________________

(Be sure you filled in your information in the box above!)
<table>
<thead>
<tr>
<th>Problem Number</th>
<th>Possible Points</th>
<th>Received Points</th>
<th>Grader Initials</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>100</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
1. (15 points)
Throughout this and other problems pertaining to our project, assume unless otherwise stated that we are playing the CSE131 game of WUggle:

- 16 squares arranged as a 4x4 board
- Each square contains a single letter. For the purposes of this exam, you can ignore QU.
- There are 16 different dice in the game.
- Each die is assigned randomly to one square of the board.
- The side of the die that shows is also randomly computed as one of the six possible sides.
- In our WUggle simplification of the Boggle game, words are formed by concatenating letters either horizontally or vertically.
- Words must be comprised of 3 or more letters to count in the game.

Consider the design of a WUggleDie object based on the following story. A WUggleDie offers the following methods:

public WUggleDie(String[] strs) is the constructor. The supplied array has exactly 6 entries, one for each side of this WUggleDie.

public String getOriginal(int side) returns the String provided to the constructor for the specified side, \(0 \leq \text{side} < 6\), of this WUggleDie.

public String getRandom() is called to return the value of a random side. Each of the sides should be equally likely to be returned by this call.

For example, consider the following code sequence:

```java
String[] sides = new String[] { "A", "B", "C", "D", "E", "F"};
WUggleDie wd = new WUggleDie(sides);

System.out.println(wd.getOriginal(0)); // prints "A"
System.out.println(wd.getRandom()); // prints "A" (maybe)
System.out.println(wd.getRandom()); // prints "D" (maybe)
System.out.println(wd.getOriginal(0)); // prints "A"
```

Continued on next page...
public class WUggleDie {

}
2. (20 points) Consider the following interface:

```java
public interface ProvidesPerimeter {
    public double getPerimeter();
}
```

In this problem you are given the beginnings of a `Rectangle` and `Circle` class. Your job is to finish them by:

- Completing the constructor (5 points)
- Completing the `getPerimeter()` method (5 points)

Here are the “stories” related to the constructors you are provided for these objects:

- When a `Rectangle` is constructed it is provided the lower-left and upper-right points of the specified rectangle. These are supplied as two `int`s each: `llx` and `lly` denote the lower-left corner, and `urx` and `ury` denote the upper-right corner. Assume that these coordinates are supplied without error: the lower-left point is no higher or further to the right than the upper-left point. In other words, `llx ≤ urx` and `lly ≤ ury`.

- When a `Circle` is constructed, it is provided a center point and a radius. The center point is specified as two `int`s, `x` and `y`, and the radius is supplied as the `int r, r ≥ 0`.

Some notes:

- In the work that follows for this problem, you are not allowed to introduce any other instance variables.
- Recall that the perimeter of a rectangle is the sum of the length of all of its sides.
- Recall that the perimeter of a circle of radius `r` is $2 \times \text{Math.PI} \times r$.

Continued on next page...
(a) (10 points)
    public class Rectangle implements ProvidesPerimeter {
        public final int llx, lly, urx, ury;

        public Rectangle(int llx, int lly, int urx, int ury) {

        }

        public double getPerimeter() {

        }
    }

(b) (10 points)
    public class Circle implements ProvidesPerimeter {
        public final int x, y, r;

        public Circle(int x, int y, int r) {

        }

        public double getPerimeter() {

        }
    }
3. (15 points) This problem pertains to how objects are organized in memory. We will use only ints, as we did in the lecture material (videos) for module 7. As was the case there, we make the simplifying assumption that every int takes up a single location in memory, and that memory begins at address 1000. Memory and its contents are shown in the table below:

<table>
<thead>
<tr>
<th>Address</th>
<th>Contents</th>
<th>Address</th>
<th>Contents</th>
<th>Address</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>12</td>
<td>1006</td>
<td>10</td>
<td>1012</td>
<td>4</td>
</tr>
<tr>
<td>1001</td>
<td>-14</td>
<td>1007</td>
<td>-12</td>
<td>1013</td>
<td>5</td>
</tr>
<tr>
<td>1002</td>
<td>-3</td>
<td>1008</td>
<td>2</td>
<td>1014</td>
<td>10</td>
</tr>
<tr>
<td>1003</td>
<td>0</td>
<td>1009</td>
<td>2</td>
<td>1015</td>
<td>12</td>
</tr>
<tr>
<td>1004</td>
<td>0</td>
<td>1010</td>
<td>8</td>
<td>1016</td>
<td>40</td>
</tr>
<tr>
<td>1005</td>
<td>4</td>
<td>1011</td>
<td>8</td>
<td>1017</td>
<td>4</td>
</tr>
</tbody>
</table>

The above table contains some Rectangle and Circle objects, whose definitions appeared in Problem 2. Based on object layout for those objects, answer the following questions.

(a) What is the perimeter of a Rectangle if it is found at address 1012?
(b) What is the (approximate) perimeter of a Circle if it is found at address 1012?
(c) What is the radius of the Circle whose center is at the origin (0,0)?
(d) At which address do you find a Rectangle that is actually a square?
(e) Why would a Circle found at address 1000 be incorrectly specified?

4. (10 points) For each of the scenarios below, place an X in the column whose associated abstract data type (ADT) is better suited for the scenario. The game mentioned below is our WUggle project game.

<table>
<thead>
<tr>
<th>Scenario</th>
<th>List</th>
<th>Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>The sequence of GridPos objects that correspond to a word found on the game grid</td>
<td></td>
<td></td>
</tr>
<tr>
<td>This ADT allows duplicates</td>
<td></td>
<td></td>
</tr>
<tr>
<td>An iterator for this ADT could visit its elements in any order</td>
<td></td>
<td></td>
</tr>
<tr>
<td>The Strings that are legal words for a game</td>
<td></td>
<td></td>
</tr>
<tr>
<td>The Terms of a SparsePolynomial</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
5. (20 points) An episode of a show is initially constructed based on the name of the show (a String) and the name of the episode (also a String), as follows:

```java
Episode casa = new Episode("South Park", "Casa Bonita");
```

where "South Park" is the name of the show and "Casa Bonita" is the name of the episode. An episode has a collection of characters that are added after construction to the episode:

```java
casa.addCharacter("Cartman");
casa.addCharacter("Kenny");
```

And, sadly, sometimes a character dies after joining the episode:

```java
casa.kill("Kenny");
```

At any point in an Episode's life, you should be able to ask whether a character is currently in the Episode:

```java
casa.contains("Kenny")
```

which will return

- true if the character has been added to the episode and has not died, or
- false if the character was never added, or was added and has since died.

At any time, you can tell how many characters are currently in the episode:

```java
casa.numCurrentlyAlive()
```

would return 2 before Kenny was killed, and 1 afterwards.

All other behaviors of an Episode object are completely up to you, such as what happens if a character is killed twice, or if a character is killed but added again later.

Below (continue on the back of the exam if necessary), write the code for the Episode class. Be sure to include the constructor, instance variables, and methods that are necessary based on the story. Do not worry about hashCode or equals.

Continued on next page...
public class Episode {
6. (20 points) In this problem you consider writing another method for the Rectangle class introduced in Problem 2. We are interested in computing a Rectangle that is the smallest such rectangle that contains a provided Circle. A picture of this is shown below.

The method you are to write for class Rectangle from Problem 2 has the following signature:

    public Rectangle contains(Circle c)

(a) (5 points) If the circle has center coordinates x and y and radius r, what are the coordinates (llx, lly) of the rectangle’s lower left corner?

(b) (5 points) If the circle has center coordinates x and y and radius r, what are the coordinates (urx, ury) of the rectangle’s upper right corner?

Continued on next page...
(c) (10 points) Below, write a method that returns a new Rectangle that is the smallest rectangle that contains Circle c.

```java
public Rectangle contains(Circle c) {
}
```

(d) (4 points) We would like two Circles to equal each other if they overlap each other exactly when drawn.

i. (2 points) Why can we not use `getPerimeter()` to determine equality?

ii. (2 points) On what instance variables should equality and hashCode be determined for a Circle?

(e) (1 points) Can the Rectangle's contains(Circle c) method be static? Why or why not?