Instructions

- **DO NOT** PUT YOUR SSN ON ANYTHING!

- **DO NOT WRITE YOUR NAME ON ANY PAGE EXCEPT THIS ONE!**

- Turn off any noise making device, especially **CELL PHONES**. You may lose up to one letter grade if your device disturbs the peace of the exam.

- You have lots of minutes. Relax, **pace yourself**, and pay attention to the point values.

- Do not add features that are not required by the problem. For example, if the instructions **don’t say anything** about user input or printing, then your program should **not** take user input or print.

- Do problems you are confident about first. If you finish the problems you know, write what you do know about other problems to gain partial credit; but erroneous information may detract from that credit or irritate the grader, so don’t make stuff up.

- Read **all** the directions **carefully** on each problem.

- Often writing a fast, rough version of a program in English or pseudocode will make your coding faster and more accurate. It also enables me to give partial credit in some circumstances.

- You may assume that input will not produce errors for the procedures described, unless the questions say otherwise.

- Do not do unnecessary testing. For example, testing for both \( x < 0 \) and \( x \geq 0 \) instead of using one test and then **else** would be considered unnecessary testing.

- As always, style and efficiency both matter.

- Have fun!
1. (18 pts) Read all of the following program. Then draw the stack and heap (as done in class) representing memory contents just before the end of main().

```java
class Tree {
    private Bird myBird;

    public static void main(String[] args) {
        Tree t1 = new Tree();
        Bird b1 = new Bird(2);
        Bird b2 = new Bird(2);
        t1.setMyBird(b1);
        Tree t2 = new Tree();
        t2.setMyBird(b1);
        b1.setNumLegs(1);
        b2.examScores[b1.getNumLegs()] = 85;
        //what do stack and heap look like now?
    }
    public Bird getMyBird() {
        return myBird;
    }
    public void setMyBird(Bird myBird) {
        this.myBird = myBird;
    }
}

class Bird {
    private int numLegs;
    public int[] examScores = {60, 80, 100, 99};
    Bird(int numLegs){ this.setNumLegs(numLegs); }
    public int getNumLegs() {return numLegs;}
    public void setNumLegs(int numLegs) {
        this.numLegs = numLegs;
    }
}
```

STACK

HEAP
2. (20 pts) Fill in the blanks with Java code. You may not need all lines. Do not alter existing code.

    public class Brick implements Comparable<Brick> {
        private int length;
        private double weight;
        public int getLength() { return length; }
        public double getWeight() { return weight; }

        // complete the class def so that Bricks are sorted by length,
        // then weight.

    }

Use your best Java style to code the following:

3. (5 pts) Make a typesafe ArrayList to hold bricks. You will use it in the next several problems.

4. (5 pts) Put two bricks in your list.

5. (5 pts) Sort the list.
6. (20 pts) Fill in the LETTERS columns to show which code fragments are required to represent the information provided. Do not represent extra information. Do not alter existing code. Not every LETTER blank will be filled. You may use letters again. Note the EXAMPLE of how to say “May the force be with you.”

1. May the
2. be with
3. you.

- All mammals can move.
- Zebras are mammals.
- Bats are mammals.
- All birds can fly.
- Crows are birds.
- Mammals and birds are animals.
- Bats, crows, and rockets can fly.
- Rockets and Zebras can move.

\[ \text{\textbackslash EXAMPLE: } \text{force } \text{letter letters}_1 \text{ letter letters}_2 \text{ letter letters}_3 \]\n
<table>
<thead>
<tr>
<th>LETTERS</th>
<th>\text{Mammal}</th>
<th>\text{Mover}</th>
<th>\text{Zebra}</th>
<th>\text{Bat}</th>
<th>\text{Flyer}</th>
<th>\text{Bird}</th>
<th>\text{Crow}</th>
<th>\text{Animal}</th>
<th>\text{Rocket}</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{letter letters}_1</td>
<td>\text{letter letters}_2</td>
<td>\text{letter letters}_3</td>
<td>\text{letter letters}_1</td>
<td>\text{letter letters}_2</td>
<td>\text{letter letters}_3</td>
<td>\text{letter letters}_1</td>
<td>\text{letter letters}_2</td>
<td>\text{letter letters}_3</td>
<td>\text{letter letters}_1</td>
</tr>
</tbody>
</table>
A. class
B. interface
C. extends Animal
D. implements Animal
E. implements Mover
F. extends Mover
G. implements CompareTo
H. implements Compare
I. implements Flyer
J. extends Flyer
K. public void move(){}
L. public void move();
M. public int move(){}
N. public int move();
O. extends Mammal
P. implements Mammal
Q. public int fly(){}
R. public int fly();
S. public void fly(){}
T. public void fly();
U. extends Bird
V. implements Flyer, Mover
W. extends Flyer, Mover
X. polymorphism
Y. System.out.println("error");
Z. System.out.println("interface");
7. (12 pts) Write a class Cow that keeps track of how many Cows have been made.

```java
class Fruit {
    private int seeds;
    Fruit(int s){ seeds = s; }
}

class FruitDriver {
    public static void main(String[] args) {
        Fruit f1 = new Apple();
    }
}
```

8. (10 pts) Read the Fruit code above carefully.
   In the lines below, write one more class to make the code above compile and run correctly. The code has no output. You may not need all lines.

```java
class Cow {
    private int count;
    Cow() { count = 0; }
    void addCow() { count++; }
    int getCowCount() { return count; }
}
```