CS 201, Fall 2015 — Oct 23rd — Exam 2 Name: ______________________________

Question 1. [5 points] State a big-O upper bound on the worst case running time of the given method, where the problem size N is the number of elements in the array passed as its parameter. Explain your answer briefly.

```
public static int mystery(int[] arr) {
  int sum = 0;

  for (int i = 0; i < arr.length*arr.length; i++) {
    sum += arr[i % arr.length];
  }

  return sum;
}
```

Question 2. [5 points] State a big-O upper bound on the worst case running time of the given method, where the problem size N is the number of elements in the array passed as its parameter. Explain your answer briefly.

```
public static int mystery(int[] arr) {
  int sum = 0;

  for (int i = 0; i < arr.length*arr.length; i++) {
    sum += arr[i % arr.length];
  }

  for (int i = 0; i < arr.length; i++) {
    sum += arr[i];
  }

  return sum;
}
```

Question 3. [5 points] State a big-O upper bound on the worst case running time of the given method, where the problem size N is the number of elements in the array passed as its parameter. Explain your answer briefly.

```
public static int mystery(int[] arr) {
  int sum = 0;

  for (int i = 0; i < arr.length; i++) {
    for (int j = 0; j < arr.length; j++) {
      sum += arr[(i*j) % arr.length];
    }
  }

  return sum;
}
```

Question 4. [5 points] State a big-O upper bound on the worst case running time of the given method, where the problem size N is the number of elements in the array passed as its parameter. Explain your answer briefly.

```
public static int mystery(int[] arr) {
  int sum = 0;

  for (int i = 0; i < arr.length; i++) {
    for (int j = i; j >= i; j--) {
      sum += arr[(i*j) % arr.length];
    }
  }

  return sum;
}
```

Question 5. [5 points] State a big-O upper bound on the worst case running time of the given method, where the problem size N is the value of the method's parameter. Explain your answer briefly.

```
public static int mystery(int n) {
  int sum = 0;

  for (int i = n; i > 0; i = i / 2) {
    sum += i;
  }

  return sum;
}
```

Question 6. [5 points] Briefly explain the problem with the following method, and how to fix it.

```
public static<E> int countGreaterThan(ArrayList<E> list, E value) {
  int count = 0;

  for (int i = 0; i < list.size(); i++) {
    E elt = list.get(i);
    if (elt.compareTo(value) > 0) {
      count++;
    }
  }

  return count;
}
```

Question 7. [10 points] Complete the following method, called `makeAllPositive`. It takes a reference to an `ArrayList` of `Integer` elements as a parameter. It should change all of the negative elements in the list to positive values. Example JUnit test:

```
ArrayList<Integer> a = new ArrayList<Integer>();
a.addAll(Arrays.asList(-9, 0, -4, -2, 4));
makeAllPositive(a);
assertEquals((Integer)9, a.get(0));
assertEquals((Integer)0, a.get(1));
assertEquals((Integer)4, a.get(2));
assertEquals((Integer)2, a.get(3));
assertEquals((Integer)4, a.get(4));
```

Note that the Java compiler will automatically convert between `int` and `Integer` values.

Hints:

- Use the `size` method to get the number of elements in the list
- Use the `get` method to retrieve the value at a specific index
- Use the `set` method to change the value at a specific index

```
public static void makeAllPositive(ArrayList<Integer> list) {
```

Question 8. [10 points] Construct the class Shape that has abstract methods calcPerimeter and calcArea, which have no parameters, and return floating point numbers. Shape also has fields named type, origin, perimeter, and area. type is a String, origin is a Point, and perimeter and area are floating point numbers.

Make sure to define all of the appropriate accessor functions for the fields in Shape. Allow only the constructor to set type, and for extra credit (+2), only allow sub-classes of Shape to calculate or change perimeter and area.

Question 9. [10 points] Create the concrete class `RegularPolygon` from as a subclass of the `Shape` class you specified in Question 8. `RegularPolygon` should also implement the `Comparable` interface, comparing the areas of the two objects involved in the comparison. It has a constructor that accepts values for `type`, `origin`, `sides`, and `length`. The constructor calls `calcPerimeter` and `calcArea` to initialize the `perimeter` and `area` fields.

Implement the appropriate accessor methods, restricting access so that only the constructor can set the `side`, `length`, `perimeter`, and `area` fields. Remember to declare and implement everything necessary to make `RegularPolygon` a concrete class, except that you can insert "`<CODE>`" in the bodies of any required methods that are not accessor methods or part of the `Comparable` interface.

Programming Question

To get started, use a web browser to download the zipfile as specified by your instructor. Import it as an Eclipse project using File → Import... → General → Existing Projects into Workspace → Archive file.

Important: You may use the following resources:

- The textbook
- The lecture notes posted on the course web page
- Your previous labs and assignments

Do not open any other files, web pages, etc.

Question 10. [40 points] In the `Exam2` class, complete the `countBetween` static method. It takes an `ArrayList` (`list`) whose elements of generic type `E` are guaranteed to implement the `Comparable` interface, and also values `min` and `max` of the same type `E`. The method should return the count of how many elements of `list` have values that are greater than or equal to `min` *and* less than or equal to `max`.

JUnit tests are provided in the class `Exam2Test`. Make sure that the tests pass!

Hints:

- Compare elements using the `compareTo` method

When you are ready to submit your code, export the **CS201_Exam02** project as a zip file and upload it to the Marmoset server as **exam02**:

```
https://cs.ycp.edu/marmoset
```