Isaac Ray Shoebottom
CS 1083
Assignment 3
3429069

Section A

Answer to 1A:

```
cA.equals(cB): true

cA.equals(cC): false

cA.equals(cD): false

cA.equals(sA): false

tA.equals(sA): false
```

This is the output because we are comparing the object directly, not the contents on the objects. We are essentially asking if this is the same object in memory, not asking for the contents of the object.

Section B

Source code for Shape:

```
/**
 * Shape class
 * @author Isaac Shoebottom (3429069)
 */


public abstract class Shape implements Comparable<Shape>{
	public abstract double getArea();


	public abstract double getPerimeter();


	public boolean equalsArea(Circle obj){
		return obj.getArea() == getArea();
	}
	public boolean equalsArea(Square obj){
		return obj.getArea() == getArea();
	}
	public boolean equalsArea(EquilTriangle obj){
		return obj.getArea() == getArea();
```

```
	}


	/**
	 * Compares all shapes. Shapes must have a getPerimeter and a getArea
method
	 * @param o Input object
	 * @return Integer -1, 0, 1
	 */
	@Override
	public int compareTo(Shape o) {
		double selfPerimeterArea = getPerimeter()/getArea();
		double objPerimeterArea = o.getPerimeter()/o.getArea();
		int returnValue = 0;
		if (selfPerimeterArea > objPerimeterArea)
			returnValue = 1;
		else if (selfPerimeterArea < objPerimeterArea)
			returnValue = -1;
		return returnValue;
	}



}//End class Shape
```

Section C

Test2 Driver Program:

```
/**
 * Test2
 * @author Isaac Shoebottom (3429069)
 */
```

```
public class Test2
{
	public static void main(String[] args){
		// Create several shapes with two that
		// have the same perimeter and two references
		// that share the same object.

		Circle cA = new Circle(2);
		Circle cB = new Circle(2);
		Circle cC = new Circle(3);
		Square sA = new Square(Math.PI);
		EquilTriangle tA = new EquilTriangle(2);



		System.out.println("cA.equals(cB): "+cA.equalsArea(cB) );
		System.out.println("cA.equals(cC): "+cA.equalsArea(cC) );
		System.out.println("cA.equals(sA): "+cA.equalsArea(sA) );
		System.out.println("tA.equals(sA): "+tA.equalsArea(sA) );

	}
}//End Test2
```

Section D

Output of Test2:

```
cA.equals(cB): true
cA.equals(cC): false
cA.equals(sA): false
tA.equals(sA): false
```

Section E

Source Code for Find:

```
/**
 * Find class
 * @author Isaac Shoebottom (3429069)
 */

public class Find<T extends Comparable<T>> {

    /**
     * Gets the largest object in an array
     * @param a The input array of objects
     * @return The largest object
     */
    public T getLargest(T[] a) {
        int index = 0;
        for (int i = 1; i < a.length; i++) {
            if (a[i].compareTo(a[i - 1]) > 0) {
                index = i;
            }
        }
        return a[index];
    }

    /**
     * Checks if an object is present in an array
     * @param a The array to be checked against
     * @param key The object that will be checked to be in the array
     * @return Boolean true or false if in the array
     */
```

```
    public boolean isPresent(T[] a, T key) {
        for(T obj: a) {
            if (obj.compareTo(key) == 0)
                return true;
        }
        return false;
    }

    /**
     * Checks how many times an object of the same perimeter/area is in the
array
     * @param a The array to be checked against
     * @param key The object that will be counted
     * @return The number of times it has been counted
     */
    public int presentNTimes(T[] a, T key) {
        int counter = 0;
        for(T obj: a) {
            if (obj.compareTo(key) == 0)
                counter++;
        }
        return counter;
    }

    /**
     * Checks if the array is sorted in accending order
     * @param a The array to be checked
     * @return Boolean true or false if it is sorted
     */
    public boolean isSorted(T[] a) {
```

```
        for (int i = 1; i < a.length; i++) {
            if (a[i].compareTo(a[i - 1]) < 0) {
                return false;
            }
        }
        return true;
    }
}
```

Section F

Source of Test 3:

```
/**
 * Test3
 * @author Isaac Shoebottom (3429069)
 */

public class Test3
{
      public static void main(String[] args){
            // Create several shapes with two that
            // have the same perimeter and two references
            // that share the same object.

            Circle cA = new Circle(2);
            Circle cB = new Circle(2);
            Circle cC = new Circle(3);
            Square sA = new Square(Math.PI);
            Square sB = new Square(4);
            EquilTriangle tA = new EquilTriangle(2);
            Shape[] shapes = {cA, cB, cC, sA, sB, tA};
```

```
            Find<Shape> find = new Find<>();



            System.out.println(find.getLargest(shapes));
            System.out.println(find.isPresent(shapes, sB));
            System.out.println(find.isSorted(shapes));
            System.out.println(find.presentNTimes(shapes, sA));

            System.out.println("cA.equals(cB): "+cA.equalsArea(cB) );
            System.out.println("cA.equals(cC): "+cA.equalsArea(cC) );
            System.out.println("cA.equals(sA): "+cA.equalsArea(sA) );
            System.out.println("tA.equals(sA): "+tA.equalsArea(sA) );


      }
}//End Test3
```

Section G

Output of Test 3:

```
1.0
1.0
0.6666666666666666
1.2732395447351628
1.0
3.464101615137755
[Equilateral Triangle: Length: 2.0]
true
false
1
cA.equals(cB): true
```

```
cA.equals(cC): false
cA.equals(sA): false
tA.equals(sA): false
```

Section H

Source Code for Cat:

```
/**
 * Cat class
 * @author Isaac Shoebottom (3429069)
 */

public class Cat implements Comparable<Cat>{
    String name;
    double weight;

    /**
     * Cat object
     * @param name The name of the cat
     * @param weight The weight of the cat
     */
    Cat(String name, double weight) {
        this.name = name;
        this.weight = weight;

    }

    /**
     * For output to console
     * @return The name of the object
```

```
     */
    public String toString(){
        return "[Cat: name: "+name+"]";
    }

    /**
     * Compares cat objects
     * @param obj Input cat object
     * @return The cat that is greater, -1, 0, 1
     */
    @Override
    public int compareTo(Cat obj) {
        if(name.compareTo(obj.name) > 0)
            return 1;
        else if(name.compareTo(obj.name) < 0)
            return -1;
        else return Double.compare(obj.weight, weight);
    }
}
```

Section I

The source code for Test 4:

```
/**
 * Test4
 * @author Isaac Shoebottom (3429069)
 */

public class Test4
{
```

```
	public static void main(String[] args){

		Cat cat1 = new Cat("Nelly", 18);
		Cat cat2 = new Cat("Yenny", 9);
		Cat cat3 = new Cat("Zelly", 7);
		Cat cat4 = new Cat("Kitty", 6);
		Cat cat5 = new Cat("Penny", 10);
		Cat cat6 = new Cat("Kitty", 6);

		Cat[] cats1 = {cat1, cat2, cat3};
		Cat[] cats2 = {cat4, cat5, cat6};

		Find<Cat> find = new Find<>();

		System.out.println(find.presentNTimes(cats2, cat6));
		System.out.println(find.isPresent(cats2, cat5));
		System.out.println(find.isSorted(cats2));
		System.out.println(find.getLargest(cats2));

		System.out.println(find.presentNTimes(cats1, cat2));
		System.out.println(find.isPresent(cats1, cat4));
		System.out.println(find.isSorted(cats1));
		System.out.println(find.getLargest(cats1));
	}
}//End Test4
```

Section J

The output of Test 4:

2

true

false

[Cat: name: Penny]

1

false

true

[Cat: name: Zelly]