Midterm 2
COMP 401
Spring 2012

I have not received nor given any unauthorized assistance in completing this exam.

Signature: ______________________

Name: _________________________

PID: ___________________________
This page intentionally left blank.
Part I: True or False

30 questions, 1 points each, 30 points total.

F___ The for (Object o : iterable) construct only works for built-in arrays and ArrayLists.

T___ Subinterfaces are an example of inheritance.

F___ A subinterface can override methods specified in its parent interface.

T___ A class that implements a subinterface has an "is-a" relationship with the parent interface.

F___ A class must explicitly specify that it is implementing both a subinterface and its parent interface.

F___ All methods in the parent interface must also be specified in the subinterface.

F___ A class that implements a subinterface must be extended from a class that implements the parent interface.

F___ Only subclasses can implement subinterfaces.

T___ A subclass inherits both public and protected instance variables from the parent class.

F___ A subclass can directly access private instance variables declared in the parent class.

T___ A method defined by a subclass is unable to invoke a private method defined in the parent class.

F___ A subclass can not specify methods not already specified in the parent class.

T___ Subclassing is useful for extending existing classes with new behavior.

T___ The implementation of an overriding method in the subclass must have the same method signature as a method defined in the parent class.

T___ A subclass constructor can use super to explicitly invoke a constructor of its parent class.
F___ A call to super can be invoked anywhere within a subclass constructor.

F___ A subclass constructor can invoke any method of the parent class by using super.

T___ In an overriding method, super can be used to invoke the same method as specified in the parent class.

T___ All overriding methods in Java are virtual.

F___ A finally block will not be called unless an exception occurs.

T___ Within a try-catch-finally construct, the finally block will execute even if a return statement is encountered.

F___ An exception should generally be caught as soon as possible.

T___ A RuntimeException is not subject to the “catch or specify” policy.

T___ A class with one or more abstract methods must always be specified as an abstract class.

T___ An abstract class can not be instantiated directly.

T___ A non-abstract subclass of an abstract class must provide implementations for all abstract methods.

T___ The Decorator pattern can be an alternative to subclassing.

F___ The Decorator pattern relies on class inheritance.

T___ The Decorator pattern involves delegation.

F___ In the Observer / Observable design pattern, the Observer object periodically polls the Observable object to check if it has changed.
Part II: Subclassing

25 points total. The questions in this part rely on the code below defining the classes Shape, Square, and Circle.

```java
public class Shape {
    private int x;
    private int y;

    public Shape() {
        this(0, 0);
    }

    public Shape(int xposition, int yposition) {
        x = xposition;
        y = yposition;
    }

    public int numberOfSides() throws Exception {
        throw new Exception("Shape does not have sides.");
    }

    public int getX() {
        return x;
    }

    public int getY() {
        return y;
    }

    public void move(int dx, int dy) {
        x += dx;
        y += dy;
    }

    public String toString() {
        return "Shape at (" + x + ", " + y + ")";
    }
}
```
public class Square extends Shape {

    private int side;

    public Square(int x, int y, int sidelength) {
        this.x = x;
        this.y = y;
        side = sidelength;
    }

    public int getSide() {
        return side;
    }

    public int numberOfSides() {
        return 4;
    }

    public String toString() {
        return super.toString() + " is a square.";
    }
}

public class Circle extends Shape {

    private int radius;

    public Circle(int x, int y, int circle_radius) {
        move(x, y);
        radius = circle_radius;
    }

    public int getRadius() {
        return radius;
    }

    public String toString() {
        return super.toString() + " is a circle.";
    }
}
(2 pts) Identify any overriding (non-constructor) methods in the code.
In Square: numberOfSides() and toString()
In Circle: toString()
Also, technically, in Shape, toString() since this overrides the method defined at top of hierarchy in Object (the implied parent class of all top level classes), but specifying this is not required for full credit.

(2 pts) What is the inheritance relationship between Shape and Square?
Any statement equivalent to at least one of the following:
Shape is the parent class or superclass of Square.
Square is a subclass of Shape.
Square inherits from Shape.

(2 pts) What is the inheritance relationship between Square and Circle?

There is no inheritance relationship between Square and Circle. It is fine, but not necessary, to out that Square and Circle have the same parent class.

(2 pts) Is the constructor for Circle legal (i.e., it will compile and produce the expected result)? If not, why not?

Yes. Because super is not invoked, the default, no parameter constructor for Shape is implicitly used. The rest of the constructor moves the position of the shape to the appropriate x and y values and sets the radius instance variable specific to circles.

(2 pts) Is the constructor for Square legal? If not, why not?

No. It directly accesses the instance variables x and y which are declared private to the parent class and thus are not directly visible in the subclass.
(5 pts) Rewrite the constructors for Circle and Square to properly take advantage of the available constructors in Shape.

```java
public Circle(int x, int y, int circle_radius) {
    super(x, y);
    radius = circle_radius;
}

public Square(int x, int y, int sidelength) {
    super(x, y);
    side = sidelength;
}
```

(10 pts total, 2 per part) Given the following code to set up variables:

```java
Circle c = new Circle(2, 3, 5);
Square s = new Square(10, -2, 10);
```

What is value of each of the following expressions (write “Error” if the expression is illegal and “Exception” if the expression raises an exception):

a) `c.toString()`  
   Shape at (2, 3) is a circle.

b) `((Shape) s).numberOfSides()`  
   4

c) `s.getSide()`  
   10

d) `((Shape) c).getRadius()`  
   Error

e) `c.numberOfSides()`  
   Exception
Part III: Exception Handling

20 points total. The questions in this part rely on the code below defining the classes IllegalLengthException, SideLengthZeroException, TriangleInequalityException, and Triangle.

```java
public class IllegalLengthException extends Exception {
    public IllegalLengthException() {
        this("Illegal side length.");
    }
    public IllegalLengthException(String message) {
        super(message);
    }
} // End of IllegalLengthException

public class SideLengthZeroException extends IllegalLengthException {
    public SideLengthZeroException() {
        super("Zero length side.");
    }
} // End of SideLengthZeroException

public class TriangleInequalityException extends Exception {
    public TriangleInequalityException() {
        super("Triangle inequality violation");
    }
} // End of TriangleInequalityException
```
public class Triangle {
    private double a;
    private double b;
    private double c;

    public Triangle (double side_a, double side_b, double side_c) throws IllegalLengthException, SideLengthZeroException, TriangleInequalityException {
        if ((side_a < 0.0) || (side_b < 0.0) || (side_c < 0.0)) {
            throw new IllegalLengthException();
        } else if ((side_a == 0.0) || (side_b == 0.0) || (side_c == 0.0)) {
            throw new SideLengthZeroException();
        } else if ((side_a + side_b <= side_c) || (side_a + side_c <= side_b) || (side_b + side_c <= side_a)) {
            throw new TriangleInequalityException();
        }

        a = side_a;
        b = side_b;
        c = side_c;
    }

    public double getArea() {
        double s = (a+b+c) / 2.0;
        return Math.sqrt(s*(s-a)*(s-b)*(s-c));
    }
} // End of Triangle.
This page left intentionally blank.
Using the classes defined, complete the implementation of test_triangles started below which accepts as parameters three arrays of side lengths. You may assume that all three arrays have at least one element and that they are all the same size. Use the corresponding entries as side lengths for a Triangle object. Your code should calculate and set the following variables by taking advantage of exception handling:

(3 pts) num_legal_triangles: Number of legal triangles.

(3 pts) num_side_zero: Number of triangles that are illegal because at least one of the side lengths is zero.

(3 pts) num_negative: Number of triangles that are illegal because at least one of the side lengths is negative.

(3 pts) num_inequality: Number of triangles that are illegal due to a triangle inequality violation.

(3 pts) average_area: The average area of all legal triangles.

(5 pts) sum_of_a_sides: The sum of all of the values in a_sides regardless of whether or not it was part of a valid triangle.

void test_triangles(double[] a_sides, double[] b_sides, double[] c_sides) {
    int num_legal_triangles = 0;
    int num_side_zero = 0;
    int num_negative = 0;
    int num_inequality = 0;
    double average_area = 0.0;
    double sum_of_a_sides = 0.0;

    // Your code should continue below and on the reverse.
double sum_of_areas = 0.0;
for(int i=0; i<a_sides.length; i++) {
    try {
        Triangle t = new Triangle(a_sides[i],
                                 b_sides[i], c_sides[i]);
        num_legal_triangles++;
        sum_of_areas += t.getArea();
    } catch (SideLengthZeroException e) {
        num_side_zero++;
    } catch (IllegalLengthException e) {
        num_negative++;
    } catch (TriangleInequalityException e) {
        num_inequality++;
    } finally {
        sum_of_a_sides += a_sides[i];
    }
}
average_area = sum_of_areas / (double) num_legal_triangles;
}