

Fall 2006 (Tuesday, October 10)

Name: _____

CS 3331 — Advanced Object-Oriented Programming

Exam 1 Practice

This test has questions and pages numbered 1 through 10.

Reminders

This test is open book and notes; but no laptop computers, PDAs, calculators, or similar devices are allowed. However, it is to be done individually, and you are not to exchange or share materials with other students during the test.

If you need more space, use the back of a page, noting this on the front.

This test is timed. Your test will not be graded if you try to take more than the time allowed. Therefore, before you begin, please take a moment to look over the entire test so that you can budget your time.

For programs and diagrams, clarity is important; if your programs or diagrams are sloppy or hard to read, you will lose points. Correct syntax also makes some difference.

There are 100 points all.

1. (10 points/6 mins) Briefly answer the following questions:

(a) Explain the life-cycle of applets by describing when applet methods such as `init`, `start`, `stop`, and `destroy` are invoked by a web browser or an applet viewer.

(b) What does the substitution property mean in object-oriented programming languages such as Java? Give an example of it.

2. (25 points/10 mins) Write a JUnit test class named `ModuloCounterTest` for the class `ModuloCounter` by filling out the skeleton code given below. Your test class should include test methods for all methods and constructors of the class `ModuloCounter`. If the method or constructor under test can terminate abruptly by throwing an exception, you should also include test data for such cases.

```
public class ModuloCounter {

    private final int mod;

    private int val = 0;

    public ModuloCounter(int mod) {
        if (mod > 0) {
            this.mod = mod;
        } else {
            throw new IllegalArgumentException("Invalid argument " + mod);
        }
    }

    public int mod() {
        return mod;
    }

    public int val() {
        return val;
    }

    public void incr() {
        val = (val + 1) % mod;
    }
}

/** A JUnit test class to test the class {@link ModuloCounter}. */
public class ModuloCounterTest extends TestCase {

    /** Creates a new instance. */
    public ModuloCounterTest(String name) {
        super(name);
    }

    /** Returns the test suite for this test class. */
    public static Test suite() {
        return new TestSuite(ModuloCounterTest.class);
    }

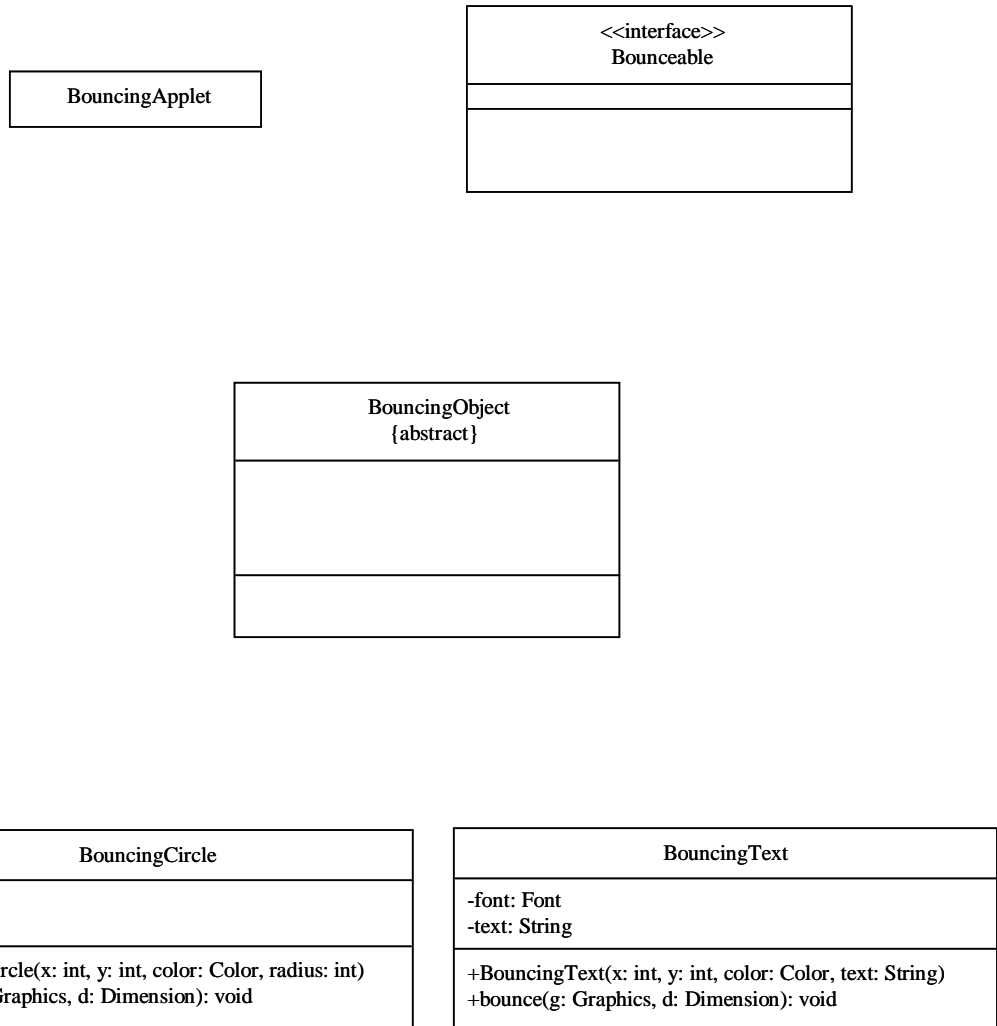
    /** Runs the tests. */
    public static void main(String[] args) {
        junit.textui.TestRunner.run(suite());
    }

    // WRITE YOUR TEST METHODS HERE (MORE SPACE ON THE NEXT PAGE)...
}
```

```
} // End of ModuloCounterTest
```

3. (65 points total/50 mins) This problem consists of four subproblems and is about writing an animation applet that bounces different kinds of balls on the screen. You are to write the main applet class and several helper classes whose design and partial implementation are given below. The applet bounces two balls in the screen. A ball changes (i.e., reverses) its direction if it touches any of the four sides of the screen. One ball is a (filled) circle and the other is a text. The applet uses the animation idiom and the double buffering technique. In subproblems (a)–(d) below, you are to fill in the missing design and implementation of the classes, some of whose skeleton code is provided. The whole program consists of one interface (**Bounceable**) and four classes (**BouncingApplet**, **BouncingObject**, **BouncingCircle**, and **BouncingText**). Before you work on individual subproblems, it might be a good idea to look at other subproblems, for the subproblems are related to each others.

(a) (10 points/10 mins) Complete the design of the bouncing applet program by adding to the following UML class diagram various relationships among the classes and the interface. For associations, aggregates, and compositions, include direction and multiplicity information. For the interface **Bounceable** and the class **BouncingObject**, specify field and method declarations; for this, use the standard UML notation. (Hint: refer to subproblems (b)–(d) below for the definitions of **BouncingApplet**, **Bounceable**, and **BouncingObject**.)



- (b) (10 points/5 mins) Given the interface `Bounceable` below, complete the class `BouncingApplet` by filling in the definition of the `paintFrame` method on page 6. The import statements are suppressed to save space.

```
/** An interface denoting bounceable objects. An object can be
 * bounced if it has the <code>bounce</code> method. */
public interface Bounceable {

    /** Bounces this object inside the screen of dimension
     * <code>d</code> by using the given graphics <code>g</code>. This
     * method will be called continuously and is supposed to adjust
     * the current position of the object and paint it. */
    void bounce(Graphics g, Dimension d);
}

/** An animation applet to show several kinds of moving objects
 * bouncing around the screen. */
public class BouncingApplet extends AnimationApplet {

    /** The set of balls to bounce. */
    private Bounceable[] balls;

    /** Initializes the applet by creating several bounceable objects. */
    public void init() {
        super.init();
        balls = new Bounceable[] {
            // circle with the initial position, color and radius.
            new BouncingCircle(50, 50, Color.RED, 10),
            // text with the initial position, color and string
            new BouncingText(10, 30, Color.YELLOW, "I Love Exam!"),
        };
    }

    /** Bounces each ball of the array <code>balls</code> by
     * calling its <code>bounce</code> method. */
    protected void paintFrame(Graphics g) {
        // WRITE YOUR CODE HERE!
    }
}
```

```

    }
}

public abstract class AnimationApplet extends java.applet.Applet {

    /**
     * The speed of animation. At every <code>delay</code>
     * milliseconds, a new frame is painted.
     */
    protected int delay = 10;

    /**
     * A timer that generates a tick event every <code>delay</code>
     * milliseconds. The timer initiates painting of a new frame.
     */
    protected Timer timer;

    /** The dimension of this applet's display screen. */
    protected Dimension dim;

    /** Off-screen image for double-buffered animation. */
    protected Image image;

    /** Off-screen graphics. I.e., <code>image.getGraphics()</code>. */
    protected Graphics offscreen;

    /** Initializes the data structure including the animation timer. */
    public void init() {
        // set the dimension
        dim = getSize();

        // set the delay
        String att = getParameter("delay");
        if (att != null) {
            delay = Integer.parseInt(att);
        }

        // create the animation timer
        timer = new Timer(delay, new ActionListener() {
            public void actionPerformed(ActionEvent event) {
                repaint();
            }
        });
    }

    /**
     * Starts the timer. This method is overridden from the class
     * Applet.
     *
     * @see #timer
     */
    public void start() {
        timer.start();
    }
}

```

```
/**
 * Stops the timer to prevent waste of CPU time. This method is
 * overridden from the class Applet.
 *
 * @see #timer
 */
public void stop() {
    timer.stop();
}

/**
 * Overridden to implement double buffering. This method calls
 * the hook method, {@link #paintFrame(Graphics)}.
 *
 * @see #paintFrame(Graphics)
 */
public void update(Graphics g) {
    if (image == null) {
        image = createImage(dim.width, dim.height);
        offscreen = image.getGraphics();
    }
    paintFrame(offscreen);
    g.drawImage(image, 0, 0, this);
}

/** Overridden to call the update method that implements double
    buffering. */
public void paint(Graphics g) {
    update(g);
}

/** Hook method to be called by the update method to implement
 * double buffering. This method should be overridden by a
 * concrete subclass to paint the current frame.
 */
protected abstract void paintFrame(Graphics g);
}
```

- (c) (20 points/15 mins) Given the abstract class `BouncingObject` below, write the concrete class `BouncingCircle`. (Hint: refer to the design given in subproblem (a) and a sample use by the class `BouncingApplet` in subproblem (b). The detailed design of this class is found on page 4; i.e., you only need to add one field, one constructor and one method.)

```
/**
 * An abstract class representing various bouncing objects. A
 * bouncing object has the current position, color, and moving speed.
 */
public abstract class BouncingObject implements Bounceable {

    /** The current position of this object. */
    protected int x, y;

    /** The color of this object. */
    protected Color color;

    /** The x and y speeds of this object; i.e., the object moves dx and
     * dy pixels each time the <code>bounce</code> method is called. */
    protected int dx, dy;

    /** Creates a new instance with the given initial position and
     * and color. The dx and dy are set to default values. */
    protected BouncingObject(int x, int y, Color color) {
        this.x = x;
        this.y = y;
        this.color = color;
        dx = 2;
        dy = 2;
    }
}

// WRITE YOUR ANSWER ON THE NEXT PAGE BY FILLING IN THE GIVEN TEMPLATE!
```

```
/** A concrete class representing bouncing circles. */  
public class BouncingCircle /* YOUR CODE HERE -> */ ----- {
```

```
} // End of BouncingCircle
```

- (d) (25 points/20 mins) Write the concrete class `BouncingText`. (Hints: refer to subproblems (a) – (c) and a sample use by `BouncingApplet` in subproblem (b); use the `FontMetrics` class (on page 193 of textbook) to measure the size of a text. The detailed design of this class is found on page 4; i.e., you only need to add one field, one constructor and one method.)

```
/** A concrete class representing bouncing texts. */  
public class BouncingText /* YOUR CODE HERE -> */ ----- {
```

```
    /** The font to paint this bouncing text. */  
    private Font font = new java.awt.Font("Sans serif", Font.BOLD, 24);
```

```
} // End of BouncingText
```