1. (10 points) 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) In Java, when is a type (a class, an interface, and an array) a subtype of another type?
2. (20 points) Draw a UML class diagram consisting of three classes: Curriculum, Course, and Lecture. Your class diagram should reflect all the following design decisions:

- A course is composed of zero or more lectures.
- A curriculum contains (or consists of) one or more courses. All the courses of a curriculum can be accessed from the curriculum, but not the other way around.
- A course may have other courses as prerequisites.
- The Course class has the following field and methods:
  
  ```java
  protected String name
  public String getName()
  public void setName(String name)
  ```

You should use the standard UML notation, especially for field and method declarations, and justify your decisions choosing particular kinds of relationships (association, aggregation, and composition).
3. (20 points) Write a JUnit test class named `CourseTest` for the class `Course` given below. Your test class should include test methods for all methods and constructors of the class `Course`. Your test cases should be reasonably complete, and if the method or constructor under test can terminate abruptly by throwing an exception, you should also include test data for such cases.

```java
class Course {
    /** The name of this course. */
    private String name;

    /** The credit of this course. */
    private int credit;

    /** Creates a new course with the given name and credit. If the given 
    * name is null or empty string, or the given credit is not 
    * positive, then an <code>IllegalArgumentException</code> is 
    * thrown.
    */
    public Course(String name, int credit) {
        if (name == null || name.length() == 0 || credit <= 0) {
            throw new IllegalArgumentException("Invalid name or credit");
        }
        this.name = name;
        this.credit = credit;
    }

    /** Returns true if and only if this course have the same credit as 
    * the given course, <code>another</code>. The argument is assumed 
    * to be non-null.
    */
    public boolean hasSameCredit(Course another) {
        return credit() == another.credit();
    }

    /** Returns the name of this course. */
    public String name() {
        return name;
    }

    /** Returns the credit of this course. */
    public int credit() {
        return credit;
    }
}
```
// WRITE YOUR TEST CLASS HERE ... (It’s okay to omit boiler-plate methods
// such as constructor, suite, and main.)
4. (total 50 points) This problem consists of three subproblems and is about writing an animation applet and enhancing it by introducing a subclass. You are to write an applet class and several helper classes whose design and partial implementation are given (see Figure 1). The applet paints a balloon, a filled circle, at the center of the viewing area. Initially the balloon is completely deflated (i.e., not visible), but it grows \( \delta \) pixels in radius at every \( \text{delay} \) milliseconds. In subproblems (a)–(c) below, you are to fill in the implementation of the classes, some of which skeleton code is provided.

```java
// File: BalloonApplet.java
import java.awt;
public class BalloonApplet extends AnimationApplet {
    private Balloon balloon;
    public void init() {
        super.init();
        balloon = createBalloon();
    }
    private Balloon createBalloon() {
        //return new GrowingBalloon(dim.width/2, dim.height/2, Color.GREEN);
        return new GrowingShrinkingBalloon(dim.width / 2, dim.height / 2, Color.GREEN);
    }
    protected void paintFrame(Graphics g) {
        balloon.draw(g, dim.width, dim.height);
    }
}
```

```java
// File: AnimationApplet.java
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public abstract class AnimationApplet extends java.applet.Applet {
    protected int delay = 10;
    protected Timer timer;
    protected Dimension dim;
    protected Image image;
    protected Graphics offscreen;
    public void init() {
```

Figure 1: Design of the balloon applet.
dim = getSize();
String att = getParameter("delay");
if (att != null) {
    delay = Integer.parseInt(att);
}
timer = new Timer(delay, new ActionListener() {
    public void actionPerformed(ActionEvent event) {
        repaint();
    }
});

public void start() {
timer.start();
}

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);
}

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);
(a) (10 points) Write the interface `Balloon` shown in Figure 1. (Hint: How is the interface used by the class `BalloonApplet`?)
(b) (20 points) Write the class `GrowingBalloon` that implements the interface `Balloon` by filling in the following skeleton code. (Hint: To draw a balloon, use the method `void fillOval(int x, int y, int width, int height)` of the class `Graphics`).

```java
import java.awt.*;
public class GrowingBalloon /* YOUR CODE HERE */
                          _________________________ {
    /** The x and y coordinates of this balloon. */
    protected int x, y;

    /** The current radius of this balloon. */
    protected int radius = 0;

    /** The growing rate of this balloon's radius. The radius grows <code>delta</code> pixels every time the <code>draw</code> method is called. @see #draw(Graphics, int, int) */
    protected int delta = 2;

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

    /** Creates a new balloon at the given position and with the given color. The initial size of the balloon is 0. */
    public GrowingBalloon(int x, int y, Color color) {
        this.x = x;
        this.y = y;
        this.color = color;
    }

    /** Increases this balloon's radius by <code>delta</code> pixels and draws it on given the graphics <code>g</code>. The balloon's position and color remain the same. */
    public void draw( // WRITE YOUR CODE HERE ... } // End of GrowingBall
(c) (20 points) Write the subclass `GrowingShrinkingBalloon` of the class `GrowingBalloon`. Here, the balloon, also positioned at the center, not only grows but also shrinks. As before, the balloon continues to grow `delta` pixels (in radius) each time the `draw` method is called. However, if the balloon reaches (i.e., touches) any of the four sides of the viewing area, it starts to shrink. The shrinking rate is the same as the growing rate. The shrinking balloon starts to grow again when it becomes completely deflated — i.e., when the radius becomes 0. Thus, the balloon repeats the growing and shrinking phases. Your program should be minimal in that it shouldn’t have unnecessary duplicate code; try to reuse as much code as possible by inheriting from the superclass `GrowingBalloon`! (Hint: Use the static method `Math.min(int, int)` to determine the minimum of two values. If your code is more than 20 lines long, you are doing it in a harder way!)
5. (extra credit: 20 points) Write an applet class, named `AlarmClock`, that behaves like the `DigitalClock` class (see the code below) except that an alarm rings at the specified time by playing an audio file `alarm.au`. The alarm time is specified by three applet parameters: `hour`, `minute`, and `second`. The audio file is assumed to be located in the same directory where the applet is located. Your applet should download the audio file only once throughout its lifetime.

Hints: Use the `AudioClip getAudioClip(URL url, String name)` method of the class `Applet` to retrieve the audio file, and the `void play()` method of the class `AudioClip` to play the audio.

```java
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.util.Calendar;
public class DigitalClock extends Applet {

    protected Timer timer;
    protected Font font = new Font("Monospaced", Font.BOLD, 48);
    protected Color color = Color.GREEN;

    public DigitalClock() {
        timer = new Timer(1000, createTimerTickHandler());
    }

    protected ActionListener createTimerTickHandler() {
        return new ActionListener() {
            public void actionPerformed(ActionEvent event) {
                repaint();
            }
        };
    }

    public void start() {
        timer.start();
    }

    public void stop() {
        timer.stop();
    }

    public void paint(/*@ non_null @*/ Graphics g) {
        Calendar calendar = Calendar.getInstance();
        int hour = calendar.get(Calendar.HOUR_OF_DAY);
        int minute = calendar.get(Calendar.MINUTE);
        int second = calendar.get(Calendar.SECOND);
        g.setFont(font);
        g.setColor(color);
        g.drawString(hour + "" + minute / 10 + minute % 10 +
            ":" + second / 10 + second % 10, 10, 60);
    }
}
```
DEFINE YOUR CLASS HERE ...