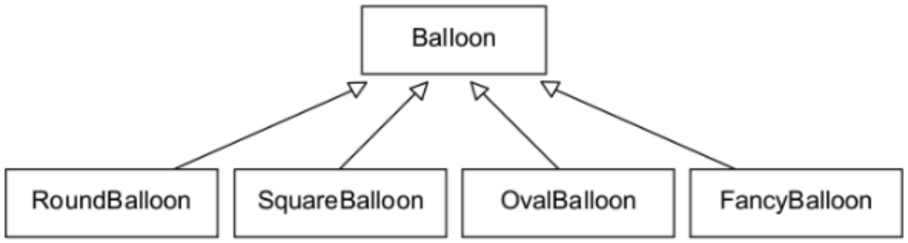
4.6 *Case Study and Lab:* Balloons of All Kinds

The purpose of this lab is to add support for balloons of different shapes to the *BalloonDraw* program. Different balloons will be implemented as subclasses of the Balloon class:



We have updated the `ControlPanel` class, replacing the "Add Balloon" button with a pull-down list, which allows the user to pick a balloon of a particular shape. We have also added a parameter to the `addBalloon` method in the `DrawingPanel` class. It indicates what kind of balloon to add to the list of balloons. Your task is to write classes for different kinds of balloons.

1. Make a copy of `JM\Ch04\BalloonDraw\Balloon.java` and name it `RoundBalloon.java`. Replace the class header in `RoundBalloon` to read

   ```
   public class RoundBalloon extends Balloon
   ```

   Remove all the fields. Rename the constructors appropriately and replace the code in them with calls to `super` as follows:

   ```
     public RoundBalloon()
     {
       super();  // this is optional: default
     }

     public RoundBalloon(int x, int y, int r, Color c)
     {
       super(x, y, r, c);
     }
   ```

`super` calls the corresponding constructor of the superclass.

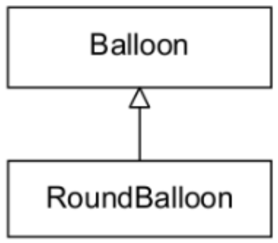
Remove all the methods from `RoundBalloon.java` except `draw`.

Go back to the `Balloon` class and remove the `draw` method, leaving only empty braces:

```
public void draw(Graphics g, boolean makeItFilled)
{
}
```

There is a better way to "disable" a method: we can declare it "abstract" and provide no code for it at all, not even empty braces. We will discuss abstract classes and methods in Chapter 12.

We now have a class `Balloon` and its subclass `RoundBalloon`:



Notice how `Balloon` and its subclass share responsibilities: `Balloon` supplies more general methods, while `RoundBalloon` supplies a `draw` method specific to the round shape.

At this point it would be nice to test what we have got, but there is a problem: the program also expects `OvalBalloon`, `SquareBalloon`, and `FancyBalloon` classes and we don't have them. The solution is to use temporary "stub" classes. For example:

```
import java.awt.Color;

public class OvalBalloon extends Balloon
{
  public OvalBalloon()
  {  }

  public OvalBalloon(int x, int y, int r, Color c)
  { super(x, y, r, c); }
}
```

Type up the above stub class for `OvalBalloon` and create similar stub classes for `SquareBalloon` and `FancyBalloon`.

Set up a project with the following eight classes in it: `Balloon.java`, `RoundBalloon.java`, your three stub classes, and the three classes provided for you in the `JM\Ch04\Balloons-All-Kinds` folder — `BalloonDraw.java`, `ControlPanel.java`, and `DrawingPanel.java`.

Test your program; make sure it runs the same way as the original *BalloonDraw* when you choose "Round" from the pull-down list. What happens when you choose "Oval" or another balloon that has not been fully implemented yet? Explain why.

2. Replace the `OvalBalloon` stub class with a real class. The easiest way of doing this is to adapt the code from the `RoundBalloon` class. Copy `RoundBalloon.java` into `OvalBalloon.java`, rename the class and the constructors, and change the `draw` method. An oval balloon is twice as tall as it is wide; its height is `2*radius`, same as before, but its width is `radius`. When you draw it, make sure its center remains at `xCenter`, `yCenter`.

   You also need to redefine the `distance` method. Copy it from the `Balloon` class into `OvalBalloon` and make a simple change, so that the "distance" from any point on the border is equal to `radius`. Hint: if you stretch the oval horizontally by a factor of two, it becomes a circle, so when you define the "distance," you need to multiply `dx` by 2.

   Retest your program thoroughly. Make sure you can "grab" an oval balloon at any point inside it to move it, and "grab" the balloon at any point on the border to stretch it.

3.■ Repeat Step 2 for the `SquareBalloon` class. Explore the documentation for the Java library class `Graphics` and find the methods that draw and fill a rectangle. The parameters for drawing a square will be the same as the parameters for drawing a circle (because the library methods that draw an oval use the oval's bounding rectangle).

   You need to redefine the `distance` method again. It turns out there is a simple formula that will make the "distance" from any point on the border of the square equal to radius. Hint: `Math.abs(dx)` returns the absolute value of `dx`; `Math.max(dx, dy)` returns the larger of `dx` and `dy`. Retest your program thoroughly again.

Continued ⇨

4.◆ Create a `FancyBalloon` class that has a shape of your own design. One example may be a vertically stretched rectangle with rounded corners (see the `Graphics` class). Another example is a diamond shape. For that you will need to use the `drawPolygon` method that takes a list of vertices. (There is a simple "distance" formula for a diamond.) Another way to create an interesting shape is to take two overlapping shapes with the same center, for example, two ovals, one stretched vertically, the other horizontally, or a square and a diamond. If you know the "distance" formula for each shape, you can combine them to make the "distance" for their combination. Hint: `Math.min(d1, d2)` returns the smaller of `d1` and `d2`.

You can also play with colors, mixing different colors and adding decorations to your balloon. Refer to the Java documentation for the `Color` class. If `c` is a `Color`, `c.darker()` returns a darker color of the same tint; `c.brighter()` returns a lighter color.