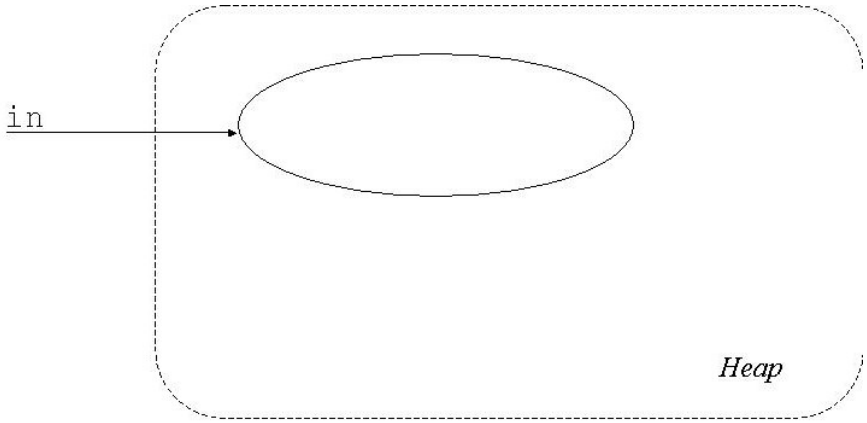
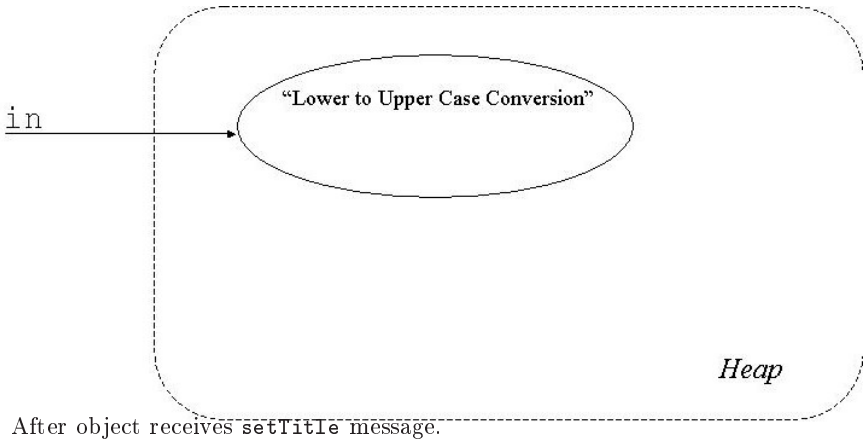
7 Classes and Methods III: Working with Objects

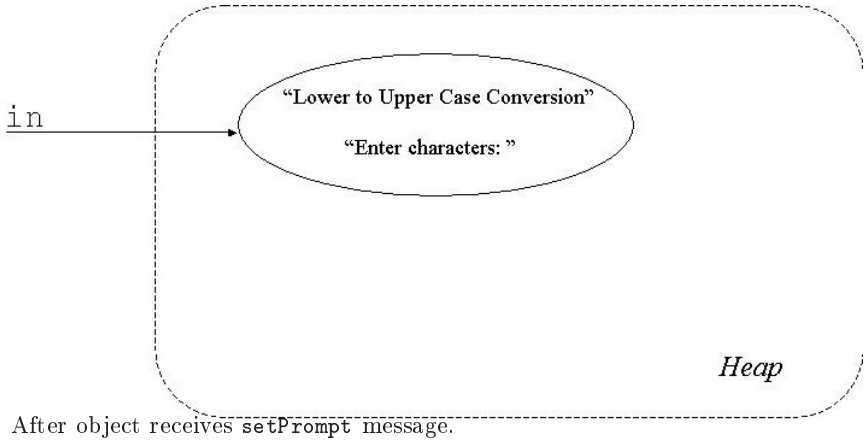
7.1



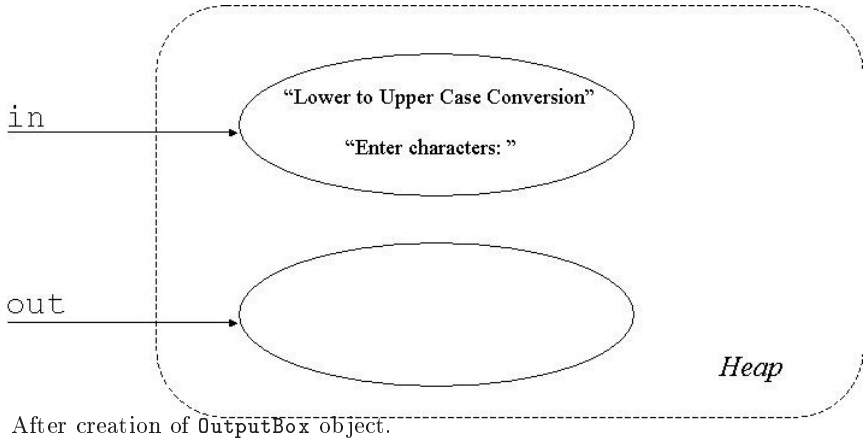
After creation of `InputBox` object.



After object receives `setTitle` message.



After object receives `setPrompt` message.



After creation of `OutputBox` object.

7.2

Here's a program to test when `InputBoxes` and `OutputBoxes` appear. By running it, you see that an `OutputBox` appears as soon as it is created, but an `InputBox` appears only when a `readXXX` is attempted. Once the `OK` button is clicked for the `InputBox`, it disappears.

```
1   import CSLib.*;
2
3   public class TestBox {
4     // Test InputBox and OutputBox to see when they appear.
5     // Author: X. Spearmint, June 2, 2001
6
7     public void test () {
8       // Create an output box, and two input boxes
9       OutputBox out = new OutputBox();
10      InputBox in1 = new InputBox("Number 1");
11      InputBox in2 = new InputBox("Number 2");
12
13      // Give 20 seconds to allow the user to move the boxes around
14      Timer.pause(10000);
15
16      int i = in2.readInt();
17      out.println("i="+i);
18      i = in1.readInt();
19    }
20  }
```

7.3

Here is the modified `Clock` class, with seconds added:

```
1   import CSLib.*;
2   import java.awt.*;
3
4   public class Clock {
5     // Maintain a clock.
6     // Author: Courtney Mickunas, November 21, 2000
7
8     int hour,
9         minute,
10        second;
11
12    public Clock () {
13      hour = 12;
14      minute = 0;
15      second = 0;
16    }
17
18    // Obtain values for hour and minute from user input.
19    public void getData () {
```

```
20        InputBox in = new InputBox();
21        in.setPrompt("What is the hour?");
22        hour = in.readInt();
23        in.setPrompt("What is the minute?");
24        minute = in.readInt();
25        in.setPrompt("What is the second?");
26        second = in.readInt();
27      }
28
29      // Return the current time as a string.
30      public String toString() { return (hour + ":" + minute + ":" + second); }
31
32      public void setHour (int h) { hour = h; }
33      public void setMinute (int m) { minute = m; }
34      public void setSecond (int s) { second = s; }
35
36      public int getHour () { return hour; }
37      public int getMinute () { return minute; }
38      public int getSecond () { return second; }
39
40      // Compare the time of this clock to another.
41      // Return true if this time is less than
42      // the time of the other clock.
43      public boolean priorTo (Clock c) {
44        return (hour < c.getHour() ||
45                hour == c.getHour() && minute < c.getMinute()
46               );
47      }
48
49      // Draw a clock with radius r at (x,y) in the DrawingBox d
50      public void display (DrawingBox d, int x, int y, int r) {
51        double theta;    // angle from 12:00
52        int x1, y1;      // end point of a clock hand
53
54        // Draw the clock in black.
55        d.setColor(Color.black);
56        d.drawOval(x-r, y-r, 2*r, 2*r);
57
58        // Set the color to blue for the hands
59        d.setColor(Color.blue);
60
61        // Draw the second hand.
62        theta = 2*Math.PI*second/60.0;
63        x1 = x + (int)(r*Math.sin(theta));
64        y1 = y - (int)(r*Math.cos(theta));
65        d.drawLine(x, y, x1, y1);
66
67        // Draw the minute hand (a little shorter).
68        theta = 2*Math.PI*(minute+second/60.0)/60.0;
69        x1 = x + (int)(r*.9*Math.sin(theta));
```

```
      y1 = y - (int)(r*.9*Math.cos(theta));
      d.drawLine(x, y, x1, y1);

      // Draw the hour hand.
      theta = 2*Math.PI*(hour+minute/60.0+second/3600.0)/12.0;
      x1 = x + (int)(r*.8*Math.sin(theta));
      y1 = y - (int)(r*.8*Math.cos(theta));
      d.drawLine(x, y, x1, y1);
    }
  }
```

7.4

Here is the modified `Clock` class, with the two new constructors:

```
  import CSLib.*;
  import java.awt.*;

  public class Clock {
    // Maintain a clock.
    // Author: Courtney Mickunas, November 21, 2000

    int hour,
        minute;

    public Clock () {
      hour = 12;
      minute = 0;
    }

    public Clock (int h, int m) {
      hour = h;
      minute = m;
    }

    public Clock (int h) {
      hour = h;
      minute = 0;
    }

    // Obtain values for hour and minute from user input.
    public void getData () {
      InputBox in = new InputBox();
      in.setPrompt("What is the hour?");
      hour = in.readInt();
      in.setPrompt("What is the minute?");
      minute = in.readInt();
    }

    // Return the current time as a string.
```

```
36    public String toString() { return (hour + ":" + minute); }
37
38    public void setHour (int h) { hour = h; }
39    public void setMinute (int m) { minute = m; }
40
41    public int getHour () { return hour; }
42    public int getMinute () { return minute; }
43
44    // Compare the time of this clock to another.
45    // Return true if this time is less than
46    // the time of the other clock.
47    public boolean priorTo (Clock c) {
48      return (hour < c.getHour() ||
49              hour == c.getHour() && minute < c.getMinute()
50             );
51    }
52
53    // Draw a clock with radius r at (x,y) in the DrawingBox d
54    public void display (DrawingBox d, int x, int y, int r) {
55      double theta;    // angle from 12:00
56      int x1, y1;      // end point of a clock hand
57
58      // Draw the clock in black.
59      d.setColor(Color.black);
60      d.drawOval(x-r, y-r, 2*r, 2*r);
61
62      // Set the color to blue for the hands
63      d.setColor(Color.blue);
64      // Draw the minute hand.
65      theta = 2*Math.PI*minute/60.0;
66      x1 = x + (int)(r*Math.sin(theta));
67      y1 = y - (int)(r*Math.cos(theta));
68      d.drawLine(x, y, x1, y1);
69
70      // Draw the hour hand.
71      theta = 2*Math.PI*(hour+minute/60.0)/12.0;
72      x1 = x + (int)(r*.8*Math.sin(theta));
73      y1 = y - (int)(r*.8*Math.cos(theta));
74      d.drawLine(x, y, x1, y1);
75    }
76  }
```

Here is the `TwoClocks` client from Chapter 5, modified to use these new constructors:

```
1   import CSLib.*;
2
3   public class TwoClocks {
4     // Place two clocks in a DrawingBox, and compare them
5     // Author: Mary Angela McDermott, December 30, 2000
6
```

```
    Clock c1,
          c2;
    DrawingBox d;
    OutputBox out;

    public void drawClocks () {
      // Draw two clocks.

      d = new DrawingBox();
      d.setDrawableSize(300,300);

      // Create and set the first clock.
      c1 = new Clock(3,15);
      c1.display(d,50,50,50);

      // Create and set the second clock
      c2 = new Clock(4);
      c2.display(d,250,50,50);
    }

    public void compareClocks () {
      // Compare the times of the two clocks.

      out = new OutputBox("Comparison");
      out.print(c1.toString() + " is ");
      if (!c1.priorTo(c2)) out.print("not ");
      out.print("prior to " + c2.toString());
    }
  }
```

```
  public class TwoClocksClient {

    public static void main (String[] args) {
      TwoClocks twins = new TwoClocks();
      twins.drawClocks();
      twins.compareClocks();
    }
  }
```

7.5

Attempting to add the new constructor

```
    public Clock (int m) {
      hour = 12;
      minute = m;
    }
```

Results in a compilation error:

```
Clock.java:26: Clock(int) is already defined in Clock
```

```
  public Clock (int m) {
         ^
1 error
```

The existing version of `Clock(int)` is:

```
public Clock (int h) {
  hour = h;
  minute = 0;
}
```

7.6

The class `OverloadTest` itself does not cause a problem. However, if you try to write a client:

```
1  public class OverloadTestClient {
2    public static void main (String[] args) {
3      OverloadTest o = new OverloadTest();
4      o.f(3,1);
5    }
6  }
```

the compiler is unable to decide which version of `f` to call. That is, the compiler could either convert the first argument to a `double` and call the first version of `f`, or it could convert the second argument to a `double` and call the second version of `f`.

1. Here is the program:

   ```
   1  public class OverloadTest {
   2    double f(double x) {return x;}
   3    int f(int i, double x) {return i;}
   4  }
   ```

   ```
   1  public class OverloadTestClient {
   2    public static void main (String[] args) {
   3      OverloadTest o = new OverloadTest();
   4      o.f(3);
   5    }
   6  }
   ```

   There is no difficulty since the call `o.f(3)` cannot possibly match the two-argument version. The compiler will convert `3` to a `double` and call the `f(double)` version.

2. Here is the program:

   ```
   1  public class OverloadTest {
   2    int g(int x, int i) {return i;}
   3    double g(double i, double x) {return i;}
   4  }
   ```

```
1  public class OverloadTestClient {
2    public static void main (String[] args) {
3      OverloadTest o = new OverloadTest();
4      o.g(3,1.3);
5    }
6  }
```

There is no ambiguity here. The compiler will never attempt to convert a `double` to an `int`. The only alternative is to convert `o.g(3,1.3)` to `o.g(3.0,1.3)` and call the `g(double,double)` version.

7.7

The following program prints all of the times between set(-50,-100) and set(50,100).

```
1  import CSLib.*;
2
3  public class TestSet {
4    public void test () {
5      OutputBox out = new OutputBox();
6      Clock c = new Clock();
7      for (int h = -50; h <= 50; h++) {
8        for (int m = -100; m <= 100; m++) {
9          c.set(h,m);
10         out.print(" "+c.toString());
11       }
12       out.println();
13     }
14   }
15 }
```

7.8

Since none of our clients have directly used any of the instance variables of `Clock`, they will be unaffected if we declare those instance variables to be `private`.

7.9

Following is the code to add to the `Clock` class a display with a Point as the center of the clock.

```
// Draw a clock with radius r at point p in the DrawingBox d
public void display (DrawingBox d, Point p, int r) {
  display(d, p.x, p.y, r);
}
```

7.10

Following is the code to add to the `Clock` class a display of the largest possible clock.

```
// Draw a clock as large as possible in the local DrawingBoox
public void display () {
  display (Math.min(dbox.getDrawableHeight()/2,
                    dbox.getDrawableWidth()/2));
}
```

7.11

The drawing methods draw the 12^{th} hour exactly the same as the 0^{th} hour. This is because the angle, θ, that is computed for hour=0 is $\theta = 2\pi(hour + minute/60)/12 = 2\pi(minute/720)$, while for hour=12, it is $\theta = 2\pi(hour + minute/60)/12 = 2\pi(1 + minute/720) = 2\pi + 2\pi(minute/720)$. However, $sin(x) = sin(2\pi + x)$ and $cos(x) = cos(2\pi + x)$, so the computations for the end-points of the hands are identical in the two cases.

7.12

The version of `Clock` that uses only a `minute` instance variable is:

```
1  import CSLib.*;
2  import java.awt.*;
3
4  public class Clock {
5    // Maintain a clock.
6    // Author: Courtney Mickunas, November 21, 2000
7
8    int minute;
9
10   public Clock () {
11     this(12, 0);
12   }
13
14   public Clock (int hour, int minute) {
15     set(hour, minute);
16   }
17
18   // Obtain values for hour and minute from user input.
19   public void getData () {
20     InputBox in = new InputBox();
21     in.setPrompt("What is the hour?");
22     setHour(in.readInt());
23     in.setPrompt("What is the minute?");
24     setMinute(in.readInt());
25   }
26
27   // Return the current time as a string.
28   public String toString() {
29     return (getHour() + ":" + getMinute());
30   }
31
```

```
32    public void setHour (int h) { set(h, getMinute()); }
33    public void setMinute (int m) { set(getHour(), m); }
34
35    public void set (int h, int m) {
36      minute = (h * 60 + m) % (12 * 60);
37      // minute is between -(12 * 60) and (12 * 60)
38      if (minute < 0) minute = minute + (12 * 60);
39    }
40
41    public int getHour () {
42      int hour = minute / 60;
43      if (hour == 0)
44        return 12;
45      else
46        return hour;
47    }
48
49    public int getMinute () { return (minute % 60); }
50
51    // Compare the time of this clock to another.
52    // Return true if this time is less than
53    // the time of the other clock.
54    public boolean priorTo (Clock c) {
55      int oldhour = getHour(),
56          oldchour = c.getHour();
57      return (oldhour < oldchour ||
58              oldhour == oldchour && getMinute() < c.getMinute()
59             );
60    }
61
62    // Draw a clock with radius r in the center of the local DrawingBox
63    public void display (DrawingBox d, int r) {
64      display(d, d.getDrawableWidth()/2, d.getDrawableHeight()/2, r);
65    }
66
67    // Draw a clock with radius r at (x,y) in the DrawingBox d
68    public void display (DrawingBox d, int x, int y, int r) {
69      double theta;    // angle from 12:00
70      int x1, y1;      // end point of a clock hand
71
72      // Draw the clock in black.
73      d.setColor(Color.black);
74      d.drawOval(x-r, y-r, 2*r, 2*r);
75
76      // Set the color to blue for the hands
77      d.setColor(Color.blue);
78      // Draw the minute hand.
79      drawHand (d, x, y, r, getMinute()/60.0, 1.0);
80
81      // Draw the hour hand.
```

```
82        drawHand (d, x, y, r, (getHour()+getMinute()/60.0)/12.0, 0.8);
83      }
84
85      public void drawHand (DrawingBox d, int x, int y, int r,
86                            double fraction, double scale) {
87        int x1, y1;
88        double theta = 2*Math.PI*fraction;
89        x1 = x + (int)(r*scale*Math.sin(theta));
90        y1 = y - (int)(r*scale*Math.cos(theta));
91        d.drawLine(x, y, x1, y1);
92      }
93    }
```