Assignment: Figures
Abstract classes and inheritance

October 24, 2011

<table>
<thead>
<tr>
<th>Author</th>
<th>Kees Hemerik (original version), Gerard Zwaan (adapted version for 2IS45)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goals</td>
<td>Learn to implement and use inheritance from an abstract base class</td>
</tr>
</tbody>
</table>

Introduction

The GUI application to be completed in this assignment should produce a window that looks like the following screen shot:

(In this assignment the GUI part of the application is already provided. An assignment describing how to develop and construct the GUI part of an application similar in appearance and functionality to this one can be found on the web page of this course.)

The application maintains a list of geometrical figures, viz. rectangles, circles, and triangles. All figures in the list are displayed in the drawing area on the left with a black border and an interior in the figures own color, and labeled with its index near its center of gravity. Lower numbered figures are drawn over higher numbered figures. Clicking within the visible part of a figure selects that figure. The selected figure will drawn with a red border on top of all other figures. Some data of the selected figure appear in the panel on the right, such as x- and y-coordinates of
its center of gravity, its perimeter, and the area enclosed by the figure. The buttons on the right
serve to clear the list of figures, to create a new rectangle, circle, or triangle, and to create a new
list of figures.

On the inside, use has been made of a small class hierarchy with an abstract base class \texttt{Figure}
and subclasses \texttt{FigRectangle}, \texttt{FigCircle}, and \texttt{FigTriangle}. The base class declares (abstract) public
methods which are common to all figures and which may be used by clients. Each of the subclasses
provides particular implementations of these methods, as well as some public methods specific to
the subclass. The list of figures and the index of the selected figure within the list are represented
by the following private fields

\begin{verbatim}
private ArrayList<Figure> figureList;
private int selected;
\end{verbatim}

If there is no selected figure \texttt{selected} contains the value $-1$. The drawing area is implemented
using a private inner class \texttt{DrawingPanel} having access to the above private fields. Finally, the
application has its own random generator (field \texttt{random} of type \texttt{Random}) to be used in the methods
that generate a random figure or a list of random figures.

The NetBeans project accompanying this assignment is working but incomplete. Many code
sections have been left out and have been replaced by a \texttt{// TODO} marker. When these sections
are completed according to their specifications, the application will behave as described.

\section*{Description of the Project}

The project consists of a main class \texttt{FiguresApp.java}, which has been developed by means of the
NetBeans GUI Builder. The main class makes use of a package \texttt{Figures}.

The package \texttt{Figures} contains definitions of the following classes:

- \texttt{Figure} is an abstract base class. It defines a number of abstract public methods, which may
  be used by clients, such as class \texttt{FiguresApp.java}.

- \texttt{FigRectangle} is a subclass of class \texttt{Figure}. It is used to represent rectangles.

- \texttt{FigCircle}, likewise, for circles.

- \texttt{FigTriangle}, likewise, for triangles.

In files \texttt{FiguresApp.java}, \texttt{FigRectangle.java}, \texttt{FigCircle.java} and \texttt{FigTriangle.java} some code sec-
tions have been left out. Their places have been marked with a \texttt{//TODO} marker. The resulting
program is working, but just produces a window without any figures drawn (see screen shot on
next page).

Clicking buttons is possible, but internally will just generate \texttt{null} references to figures, which
will not be drawn.

\section*{Open and study project}

Unzip the project, open it in NetBeans, and study its structure. Look in particular at the specifi-
cation of the abstract class \texttt{Figure} and how the subclasses \texttt{FigRectangle}, \texttt{FigCircle} and \texttt{FigTriangle}
relate to it.

\section*{Implement paintComponent of drawing panel}

In the inner class \texttt{DrawingPanel} implement method \texttt{void paintComponent(g)} according to the speci-
fication and the hints given in the source code (in the application as it is provided a list of figures
is created containing nothing but null references; in the implementation of this method elements of
\texttt{figureList} that are null should be skipped).
Implement `findFirstFigure`

In the main class `FiguresApp` implement method `int `findFirstFigure`(x,y)` according to its specification.

**Implement Rectangles**

1. Choose an internal representation for class `FigRectangle` by means of some private or protected instance variables.

2. Implement the methods of class Rectangle according to their specifications.

3. In the main class `FiguresApp`, implement method `randomRectangle()` according to its specification (note: the width and height of the drawing panel can be obtained with its functions `getWidth()` and `getHeight()`).

4. Run the application. It should now be possible to generate and draw a rectangle by clicking button `New Rectangle`.

**Implement Circles**

Follow the same steps for implementation of circles, this time using class `FigCircle` and method `randomCircle()` in main class `FiguresApp`.

**Implementing Triangles**

The implementation of triangles is slightly more involved than that of rectangles or circles. Although essentially the same steps should be taken, some additional analysis of the problem is required and some additional helper methods have to be declared. We give a few hints:
Implementation of getPerimeter()

The perimeter of a triangle is the sum of the lengths of its sides. For computing the length of a side with end points \((x_a, y_a)\) and \((x_b, y_b)\) use a function `double dist(int xa, int ya, int xb, int yb)` based on the theorem of Pythagoras (582 - 500 B.C.).

Implementation of getArea()

The area \(O\) of a triangle with sides \(a\), \(b\) and \(c\) may be computed by the formula of Heron (10 - 70 A.D.):

\[
O = \sqrt{s(s-a)(s-b)(s-c)}
\]

where \(s = (a + b + c)/2\).

Implementation of containsPoint()

Let \(ABC\) be a triangle and let \(P\) be a point. Point \(P\) is inside the triangle iff the conjunction of the following 3 conditions holds:

- \(P\) and \(A\) are on the same side of \(BC\)
- \(P\) and \(B\) are on the same side of \(CA\)
- \(P\) and \(C\) are on the same side of \(AB\)

Let \((x_0, y_0)\) and \((x_1, y_1)\) be two points. the line equation of the straight line through \((x_0, y_0)\) and \((x_1, y_1)\) is

\[
(y - y_0)(x_1 - x_0) - (x - x_0)(y_1 - y_0) = 0
\]

In order to determine whether two points are on the same side of the line one can use the term \((y - y_0)(x_1 - x_0) - (x - x_0)(y_1 - y_0)\) as a discriminator. For all points \((x, y)\) on the line it yields 0; for all points \((x, y)\) on one side of the line it yields a positive value; for all points on the other side of the line it yields a negative value.