I/O Framework and Case Study

CS151

Chris Pollett

Nov. 2, 2005.

Outline

- Character Streams
- Random Access Files
- Design case study
  - Planning
  - Iterations

Character Streams

- Java internally represents strings as Unicode which uses two bytes/char.
- Externally, text files are stored in a variety of different formats depending on the machine's locale.
- Here a *locale* means a geographic region which generally shares the same language.
- The default locale here in California is ISO-8859-1. Also known as ASCII.
- Character based I/O streams perform conversions between Unicode and locale characters when reading and writing strings.
- This is in contrast to the byte streams we talked about last day which are not locale sensitive.

More on Character Streams

- Two abstract classes Reader and Writer are at the root of the inheritance hierarchy for character-based I/O.
- Reader -- supports read(), read(ca), read(ca,off,len), close()
- Writer -- supports write(c), write(ca), write(ca,off,len), close()
- InputStreamReader and OutputStreamReader are two filter that serve as a bridge to make a stream into a reader:

  Ex: InputStreamReader isr =new InputStreamReader(new FileInputStream("hi.txt"));

  /* another constructor takes a stream and an encoding name*/
- Other Reader's include FileReader and BufferedReader
- Other Writer's include FileWriter, BufferedWriter,and PrintWriter.

  The last supports print and println which are very useful.

Random Access Files

- Random access files support both reading and writing data at any position in a file.
- The Java class that encapsulates this ability is RandomAccessFile.
- The constructor is RandomAccessFile(filename, mode) where mode is a string “r” (for read-only), “rw” (read-write), “rws”, or “rwd” (rw with force-writing).
- This class implements DataInput and DataOutput as well as supports the methods:
  - seek(l) to move to the lth byte from the start of the file
  - skipBytes(i) to move forward or backward with the file i bytes (backward if negative).

Design case study

- Chapter 9 in the book consists of an extensive case study of a drawing pad programming.
- This study illustrates the iterative development process so we will go through it in some detail.

Planning

- It is decided in the planning phase that the program will use Swing rather than AWT.
- The following requirements were decided upon. The drawing tool should support:
  - Scribbling and drawing various shapes
  - Saving drawings to files and loading the drawings from files
  - Typing from the keyboard
  - Choosing fonts and colors

Iterations

- The plan to develop this project is divided into six iterations, each of which adds functionalities to the previous iteration.
- The iterations make heavy use of design patterns to keep the code flexible enough to make it easy to go from one iteration to the next.
- The iterations are:
  1. Create a simple scribble pad, consisting of only a canvas for scribbling
  2. Add support for saving, loading, a menu bar, and dialogs.
  3. Refactor to support various tools for different shapes.
  4. Add tools for lines, rectangles, and ovals.
  5. Refactor and tools for filled versions of (4).
  6. Add tools for drawing text.

Iteration 1

- The application class for this iteration is Scribble.
- This extends JFrame and has on it a ScribbleCanvas which extends JPanel.
- The ScribbleCanvas has a ScribbleCanvasListener on it that implements MouseListener and MouseMotionListerner.

ScribbleCanvas

- Has fields mouseButtonDown (boolean); x, y -- the location of the mouse in the canvas; and listener -- for the ScribbleCanvasListener.
- MouseListener needs: mousePressed, mouseReleased, mouseEntered, mouseExited, mouseClicked implemented.
- MouseMotionListener needs: mouseDragged and mouseMoved.
- ScribbleCanvasListener gives blank implementations to all but mousePressed, mouseReleased, and mouseDragged.
- The desired behavior for scribbling is: a stroke begins when a mouse button is pressed, it continues when the mouse is dragged, and it ends when the mouse is released.

Example ScribbleCanvasListener Methods

```
public void mousePressed(MouseEvent e)
{
        Point p = e.getPoint();
        canvas.mouseButtonDown = true;
        canvas.x =p.x;
        canvas.y =p.y;
}
public void mouseReleased(MouseEvent e)
{
      canvas.mouseButtonDown = false;
}
public void mouseDragged(MouseEvent e)
{
      Point p =e.getPoint();
      if(canvas.mouseButtonDown)
      {
              canvas.getGraphics().drawline(canvas.x, canvas.y, p.x, p.y); canvas.x =p.x; canvas.y=p.y;
      }
}
```

Iteration 2

- For this iteration we need:
  - to be able to store drawings so that they can be redrawn
  - to be able to save the drawing into files and load them
  - build a menu bar
  - use file dialogs
  - create a dialog box for selecting colors

Strokes

- The original draw program draws strokes to the screen but does not store them internally.
- In this iteration, ScribbleCanvas has a ArrayList of Stroke objects on it called strokes. It also keeps track of a curStroke and a curColor.
- A Stroke has an ArrayList of Point's called points and a color.
- A Stroke supports the methods setColor(), getColor(), addPoint(), getPoints()

New ScribbleCanvas

- To the original ScribbleCanvas the following methods are added:
  - setCurColor()
  - getCurColor()
  - startStroke() --invoked by the listener when a stroke is started (calls Stroke constructor)
  - addPointToStroke() -- invoked by the listener to append a point to the current stroke
  - endStroke() -- adds the current stroke to the list of Stroke's
  - paint() -- drawing is repainted onto canvas based on strokes stored internally
  - newFile() , saveFile(), openFile() -- are for saving/loading

Scribble Constructor/ Menubar

```
public Scribble()
{
        setTitle("Scribble Pad");
        canvas = new ScribbleCanvas();
        getContentPane().setLayout(new BorderLayout);
        getContentPane().add(createMenuBar(), BorderLayout.NORTH);
        getContentPane().add(canvas, BorderLayout.CENTER);
        //… rest of constructor
}
protected JMenuBar createMenuBar()
{
      JMenuBar menuBar = new JMenuBar();
      JMenu menu = new JMenu("File");
      JMenuItem mi = new JMenuItem("New");
      menuBar.add(menu);
      menu.add(mi);
      mi.addActionListener(new NewFileListener()); //…rest of method
}
```

NewFileListener

```
class NewFileListener implements ActionListener
{
    public void actionPerformed(ActionEvent e)
    {
        newFile();
    }
}
```