CS102

Introduction to
data structures, algorithms,
and object-oriented
programming

April 10th, 2017

File Input & Output (File I/O)

- File I/O in Java can be accomplished by using one of many built-in Java classes imported from java.io.*
  - Reading input from a file:
    - **BufferedReader inFile = new BufferedReader(new FileReader("input.dat"));**

      Function: Reads text as a stream of characters from input.dat.
  - Writing output to a file:
    - **PrintWriter outFile = new PrintWriter(new FileWriter("output.dat"));**

      Function: Prints formatted representations of objects to text output stream, output.dat.

File I/O Exceptions

- Doing any file operations requires the programmer to either
  1. use a try/catch block around every opening of a file and every read from the file
  2. write a throws clause on every method in the call stack up to and including the main method.
  3. Exception is very high level, so it can be used for IOExceptions and FileNotFoundExceptions.

String.split method

- The StringTokenizer class allows an application to break a string into tokens and, at one time, was the favored way to parse a string into whitespace-delimited words.
- But, StringTokenizer has been *deprecated* in favor of using the split method of the String class.
- StringTokenizer still works for the sake of "backwards compatibity", but you should learn to use the String.split method to stay current.

String.split("")

- The split method of the String class allows you to get the same result as the StringTokenizer (with a little more work).
- The "\\s" says to split the string by whitespace. Split returns an array of Strings.
- The array of Strings called result (shown below) will contain an array of Strings after this code is run, with result[0]="this", result[1]="is", result[2]="a", and result[3]="test".

```
String[] result = "this is a test".split("\\s");
for (int x=0; x < result.length; x++) {
    System.out.println(result[x]);
}
```

Using "hard-coded" strings for file names

```
1. import java.io.*;
2. public class TestReadWriteSplit {
3.    public static void main (String[] args) throws Exception{
4.     BufferedReader fileIn = new BufferedReader
                              (new FileReader("in.txt"));
5.     PrintWriter outFile = new PrintWriter
                              (new FileWriter ("out.txt"));
6.     String line;
7.     String[] token;
8.     while ((line = fileIn.readLine()) != null) {
9.         token = line.replaceAll("[^a-zA-Z ]","")
                   .toLowerCase().split("\\s+");
10.        String lcString = "";
11.        for(int i = 0; i < token.length; i++) {
12.          lcString += token[i];
13.        }
14.        outFile.println(lcString);
15.    }
16.    fileIn.close();
17.    outFile.close();
18.   }
19. }
```

Hard coding is considered very poor style.

Reading Command-Line Arguments

- Command-line arguments are read through the main method's array of Strings parameter, usually called args (or whatever you call this parameter to main).

  IMPORTANT: *You must have a non-empty file in the same directory as the TestReadWriteSplit.class file called "in.txt" when you run this program!!* Also, any file called "out.txt" in the current directory will be overwritten.

  All I/O files should be closed when you are done with them.

Reading Command-Line Arguments

- In the version of the TestReadWriteSplit program on the next slide, suppose args[0] = "input.txt" and args[1] = "output.txt" during execution of the program.
- You can run this file in the Interactions window of DrJava (after it compiles with no syntax errors) by typing:

```
java TestReadWriteSplit input.txt output.txt
```

- Before this will work, you need to create a non-empty file in the same directory as your Java program called "input.txt". If you have any file in the directory that is already called "output.txt", its contents will be overwritten.

Reading Command-Line Arguments

```
1. import java.io.*;
2. public class TestReadWriteSplit {
3.    public static void main (String[] args) throws Exception{
4.     BufferedReader fileIn = new BufferedReader
                          (new FileReader(args[0]));
5.     PrintWriter outFile = new PrintWriter
                          (new FileWriter (args[1]));
6.     String line;
7.     String[] token;
8.     while ((line = fileIn.readLine()) != null) {
9.         token = line.replaceAll("[^a-zA-Z ]","")
                .toLowerCase().split("\\s+");
10.         String lcString = "";
11.         for(int i = 0; i < token.length; i++) {
12.           lcString += token[i];
13.         }
14.         outFile.println(lcString);
15.     }
16.     fileIn.close();
17.     outFile.close();
18.   }
19. }
```

Review of Swing Components

- JButton:
  - *Constructor* takes String parameter which is text on button
  - *Generates ActionEvent* when button clicked
  - *Registration of Listener – addActionListener* in constructor
  - *Events:* can call getActionCommand or getSource() on event
  - *Changes:* can setText to be different, can disable button
- JLabel:
  - *Constructor* takes String which is text written
  - *No events generated*
  - *Changes:* can setText to be different
  - Can hold images instead of just text.
- JTextField:
  - *Constructor* takes int which is preferred number of characters, String, String and int, or nothing
  - *Generates an ActionEvent* when return is pressed
  - *Changes:* can setText to be different

10

Swing Components

- JPanel: fundamental class in Swing.

  The basic *JPanel* is just a blank rectangle. There are at least two different ways to make use of a *JPanel*:

  1. **add other components** to the panel
  2. **draw something** in the panel.

  JPanels can be used as drawing surfaces:

  1) define a class that is a subclass of *JPanel* and
  2) write a paintComponent method in that class to draw the desired content in the panel. Defining this method is overriding the method of the same name in the superclass, so the first line must call super.paintComponent();

11

Class Exercise

- Create a GUI that has a single square JButton in the exact center of the window. When clicked by user, the background color of this JPanel turns red.

  After class, see CenterButton.java

- Go over lab 8 – writing the program of assignment 5.

12

Mouse Events

- Mouse events require 2 different interface implementations, depending on what you want to do
- The simplest mouse events are defined in the MouseListener interface:
  public void mousePressed(MouseEvent evt);
  public void mouseReleased(MouseEvent evt);
  public void mouseClicked(MouseEvent evt);
  public void mouseEntered(MouseEvent evt);
  public void mouseExited(MouseEvent evt);
- The second type of mouse listener is the MouseMotionListener interface:
  public void mouseDragged(MouseEvent evt);
  public void mouseMoved(MouseEvent evt);

4 Steps of Event Handling

Four Steps of ActionEvent Handling

1. import java.awt.event.*
2. have "implements ActionListener"
3. define the event-handling methods
4. addActionListener to appropriate components

Four Steps of Mouse Event Handling

1. import java.awt.event.*
2. "implements MouseListener"
3. define the event-handling methods
4. addMouseListener to appropriate components

Rule of Drawing

Drawing rule: (often violated) all drawing shall henceforth and forever be done in a paintComponent() method.

In real-life programming of drawing applications, many people (including the author of our book) violate this rule by obtaining the Graphics content (if class extends JPanel) like so:

Graphics g = getGraphics(); // Graphics context for drawing directly out of any method in the class, not just the paintComponent method.

If you do use the above command to access a Graphics component, you need to use g.dispose() in the method before it ends.

What if I don't want to use ALL the MouseListener methods?

```
private static class RepaintOnClick implements MouseListener {

        public void mousePressed(MouseEvent evt) {
            Component source = (Component)evt.getSource();
            source.repaint();
        }

        public void mouseClicked(MouseEvent evt) { }
        public void mouseReleased(MouseEvent evt) { }
        public void mouseEntered(MouseEvent evt) { }
        public void mouseExited(MouseEvent evt) { }

    }
```

You either need to put in empty method bodies for each event or…

What if I don't want to use ALL the MouseListener methods?

```
private static class RepaintOnClick extends MouseAdapter
    implements MouseListener
{

        public void mousePressed(MouseEvent evt) {
            Component source = (Component)evt.getSource();
            source.repaint();
        }

 }
```

Use the MouseAdapter class, which fills the mouse methods in for you behind the scenes. Note: MouseAdapter is a class, not an interface. Also note: anonymous inner classes don't have to be of interface type. They can also be a class type.

Drawing in GUIs (cont)

- The paintComponent method is only called once by the system. If you want to re-call it, use the "repaint()" method.
- To set the color of text, use name.setForeground(Color). This can be called on any component.
- The shapes you can draw are listed in our on-line textbook, in section 6.2.4.
- In a Graphics context, you need to do
  g.setColor(Color.BLUE); // or some other color
  before you draw the object. It can only be set to one color at a time.

18

Limitations of JPanels

- A JPanel is not a component that can be displayed on its own. That's why I created the main method in SimpleDraw…to call the zero-parameter constructor of SimpleDraw and create a JFrame to hold the components, including the JPanel.
- Notice that the JFrame and the constructor have no connection to the JPanel where the drawing takes place, but they can occupy the same space.

19

Adding KeyListeners

- public void keyPressed(KeyEvent evt);
- public void keyReleased(KeyEvent evt);
- public void keyTyped(KeyEvent evt);

20