CSE373: Data Structures and Algorithms

Java Review and GUI Construction

Steve Tanimoto
Autumn 2016

Topics for Today

- Using Java arrays to implement queues (with the "circular array" technique). --serving in part as a brief review of Java.
- *A sample Java + Swing application.*
- *JFrame, JPanel, JMenu, JMenuItem*
- *The ActionListener interface.*

Why cover this material now?

- Get us all back into Java programming.
- Cover just enough about Swing so that we can do the programming assignments without Swing being an obstacle.
- The visual aspects of our assignments make them more fun, more realistic, and in some cases easier to debug.

The Queue ADT (review from lecture 1)

- Data: a collection of items, allowing repeated elements, in a particular order (which is based on the FIFO access discipline).
- Two primary methods:
  - enqueue(elt): inserts an element at the tail of the queue.
  - dequeue(): removes and returns an element from the head of the queue.

Class CircularArrayQueue

- In order to implement our queue ADT using an array, we will need members for the array itself, for the front, back and next "cursors" (indices), and then some extra items related to the size and the resizing of the array.
- For simplicity here, let's assume the elements are of type int.

```
public class CircularArrayQueue {
   private int[] array; // Where the elements are actually stored.
   private int front, back, next; // The "cursors" or "indexes".
   private int qsize; // Actual number of elements in the queue.
   private int arraySize; // Current capacity of the array.
   private static int initialArraySize = 2; // Starting capacity.
```


A Constructor

- We'll use a small initial size, in part so that it is easy to test the resizing mechanism.

```
 public CircularArrayQueue() {
   array = new int[initialArraySize];
   for(int i = 0; i < initialArraySize; i+=1)
        {array[i]=-1;} // Initialize the array.
   arraySize = initialArraySize;
   qsize = 0;
   front = 0;
   back = -1;
   next = 0;
}
```


The enqueue Method

```
// Insert an element into the queue at the "next" array location.

public void enqueue(int elt) {
    if (qsize==arraySize)
        { doubleTheArraySize(); }
    array[next] = elt;
    back = next;
    next = (next + 1) % arraySize;
    qsize += 1;
}
```

The dequeue Method

```
// Remove and return the element at the head (front) of the queue.

public int dequeue() throws Exception {
   if (qsize==0) {
       throw(new Exception("Empty Queue Exception")); }
   int elt = array[front];
   front = (front + 1) % arraySize;
   qsize -= 1;
   return elt;
}
```


The Method doubleTheArraySize

```
// Allocate a new array with double the size of the old one, and copy the
// queue elements into the new array. Then throw away the old array.
void doubleTheArraySize() {
  int[] tempArray = new int[2 * arraySize];
  for(int i = 0; i < 2 * arraySize; i+=1)
        {tempArray[i]=-1;} // Initialize the array.
  int ti = 0;
  int oi = front;
  for(int i = 0; i < arraySize; i += 1) {
       tempArray[ti] = array[oi];
       oi = (oi + 1) % arraySize;
       ti += 1;
  }
  front = 0; // replace the cursor values with positions in the new array.
  back = ti - 1;
  next = back + 1; // No need to worry about wraparound here,
       // because the elements use the first half of the new array.
  array = tempArray; // Remove the old array by replacing it with the new one.
  arraySize = 2 * arraySize;
}
```


Comments on this implementation

- Handles only elements of type int.
- Starts with a small initial size. Assuming most queues will have more than 10 elements, it would probably be better to start with an initial size of 10, saving time-consuming calls to doubleTheArraySize.
- Some applications may not require the array-resizing code.
- Never shrinks the arrays. That could be added in order to save space in some use cases.
- Other methods may be desired, such as getSize(), getArraySize(), printState(), isEmpty(), get(index i), etc.
- This implementation makes minimal use of Java utility classes, and therefore serves as a more transparent implementation of this basic data structure than one based on heavier Java infrastructure such as ArrayList.

Other names and views for this technique can easily be found online. For example...

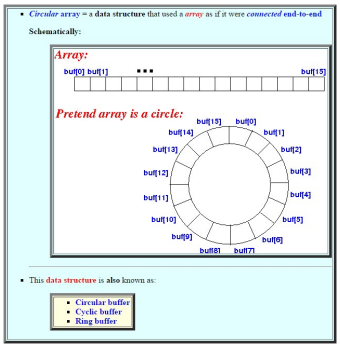
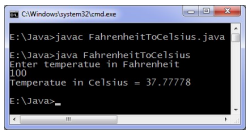


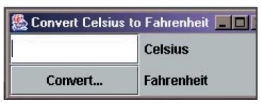
Image source: http://www.mathcs.emory.edu/~cheung/Courses/171/Syllabus/8-List/array-queue2.html

Graphical User Interfaces in Java

- GUI = Graphical User Interface.
- Compare a textual interface with a GUI.



Textual Interface



Graphical User Interface

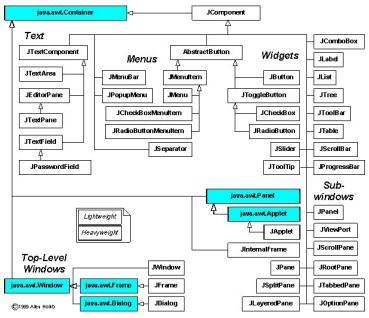
Credits: http://www.programmingsimplified.com/java/source-code/java-program-to-convert-fahrenheit-to-celsius http://www2.sys-con.com/itsg/virtualcd/java/archives/0804/dutta/index.html

Java Swing

- (My personal interpretation of the name: SoftWare for INteractive Graphics.*)
- Swing is a library of components that can work together, permitting complex GUIs to be constructed.
- The two main things to learn about are:
  - The Components themselves and how that relate to each other (mainly about the Graphics)
  - The Event model (mainly about the User Interaction)

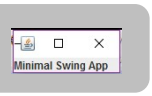
*According to the Oracle website: The real inspiration for the name was something up and coming in the San Francisco area in the later 1990s: Swing dancing. Originally, the name for the technology was "the Java Foundation Classes" (JFC). https://blogs.oracle.com/thejavatutorials/entry/why_is_swing_called_swing

SWING Components (Class Hierarchy)



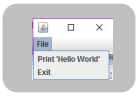
http://www.holub.com/goodies/images/swing_component_hierarchy.gif

A Minimal Swing Application




```
import javax.swing.*;

public class MinimalSwingApp extends JFrame {
   JLabel label;

   public MinimalSwingApp() {
        label = new JLabel("Minimal Swing App");
        add(label);
   }

   public static void main(String[] args) {
        MinimalSwingApp a = new MinimalSwingApp();
        a.pack();
        a.setVisible(true);
   }
}
```

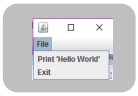

A Small Swing App with a Menu




```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JFrame;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;

public class MenuDemo extends JFrame implements ActionListener {

   JMenuBar mb;
   JMenu fileMenu;
   JMenuItem hwItem, exitItem;
```

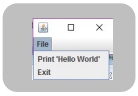
•••

MenuDemo (continued)




```
   // constructor:
   public MenuDemo() {
        mb = new JMenuBar();
        fileMenu = new JMenu("File");
        hwItem = new JMenuItem("Print 'Hello World'");
        exitItem = new JMenuItem("Exit");

        fileMenu.add(hwItem);
        fileMenu.add(exitItem);
        mb.add(fileMenu);
        setJMenuBar(mb);
        hwItem.addActionListener(this);
        exitItem.addActionListener(this);
   }
```


MenuDemo (continued)




```
   // Handler for menu-item selections:
   @Override
   public void actionPerformed(ActionEvent e) {
        JMenuItem whichItem = (JMenuItem) e.getSource();
        if (whichItem==hwItem) {
                System.out.println("Hello World!");
        }
        if (whichItem==exitItem) {
                System.exit(0);
        }
   }

   // main method to actually create and show the MenuDemo window.
   public static void main(String[] args) {
        MenuDemo md = new MenuDemo();
        md.pack();
        md.setVisible(true);
   }
}
```


Assignment 1

Add Undo and Redo capabilities to a Java Swing App you are given.
This app is operated by menu commands.
Here is an excerpt from the starter code, showing how the menus are set up.

```
//   ImageEnhancer.java ...

 private void createMenu() {
    menuBar = new JMenuBar();
    fileMenu = new JMenu("File");
    editMenu = new JMenu("Edit");
    imageMenu = new JMenu("Image");
    exitItem = new JMenuItem("Exit");
    undoItem = new JMenuItem("Undo");
    redoItem = new JMenuItem("Redo");
    darkenItem = new JMenuItem("Darken");
    blurItem = new JMenuItem("Blur");
    sharpenItem = new JMenuItem("Sharpen");
    photoNegItem = new JMenuItem("Photo-negative");
    thresholdItem = new JMenuItem("Threshold");
```

```
    exitItem.addActionListener(this);
    undoItem.addActionListener(this);
    redoItem.addActionListener(this);
    darkenItem.addActionListener(this);
    blurItem.addActionListener(this);
    sharpenItem.addActionListener(this);
    photoNegItem.addActionListener(this);
    thresholdItem.addActionListener(this);

    fileMenu.add(exitItem);
    editMenu.add(undoItem);
    editMenu.add(redoItem);
    imageMenu.add(darkenItem);
    imageMenu.add(blurItem);
    imageMenu.add(sharpenItem);
    imageMenu.add(photoNegItem);
    imageMenu.add(thresholdItem);
    menuBar.add(fileMenu);
    menuBar.add(editMenu);
    menuBar.add(imageMenu);
```

`createMenu()` *(continued)*

```
  undoItem.setEnabled(false);
  redoItem.setEnabled(false);

}
```

Your program will adjust the "enablement" of these items depending on whether the user has anything that can be undone or redone at the time.

Another Excerpt: a method for responding to user menu selections.

```
// We handle menu selection events here: //
public void actionPerformed(ActionEvent e) {
 // CSE 373 Students: Add code in this method to save the current buffered image for
 // undoing and dispose of any redoable actions.
 // Also add code to enable and disable the Undo and Redo menu items, and to process
 // these items when the user selects them.
 //System.out.println("The actionEvent is "+e);
 // This can be useful when debugging.
 if (e.getSource()==exitItem) { System.exit(0); }
 if (e.getSource()==blurItem) { blur(); }
 if (e.getSource()==sharpenItem) { sharpen(); }
 if (e.getSource()==darkenItem) { darken(); }
 if (e.getSource()==photoNegItem) { photoneg(); }
 if (e.getSource()==thresholdItem) { threshold(); }
 gWorking.drawImage(biFiltered, 0, 0, null); //Draw the pixels from biFiltered into biWorking
 repaint(); // Ask Swing to update the screen.
 printNumbersOfElementsInBothStacks(); // Report on the states of the stacks.
 return;
}
```

And add code to handle the undoItem and redoItem selections.

Swing Conclusion

- Java's Swing library offers a rich set of widgets for constructing graphical user interfaces (GUI's).
- Containers and layouts can be used to organize the spatial structure of the interface.
- Basic interaction with menus and buttons can be handled using the ActionListener interface.
- Additional information is available at http://docs.oracle.com/javase/tutorial/uiswing/
- Although we are using Swing in some of our apps, it is just a tool and you only need to learn about those parts of Swing you will be modifying or those that directly support them.