Object Oriented Programming and Design in Java

Session 6
Instructor: Bert Huang
Announcements

- Homework 1 due Wednesday Feb. 17th 11 AM
- Lauren's office hours moved to 8:30-10:30 PM (just this week)
- For fastest email queries, email all TAs and me
  - {bert@cs., jwg2116@, lep2128@, yh2315@}@columbia.edu
Review

- Introduction to Java graphics
- Swing classes: JFrame, JComponent, JButton, JTextField, JPanel
- ActionListener interface
- Graphics: Graphics2D
Today’s Plan

• Named ActionListeners
• Timers
• Interfaces and polymorphism
  • Examples: List, Comparator, Collection, Iterator
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class GraphicsTester2 {
    public static void main(String [] args) {
        JFrame frame = new JFrame();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setLayout(new FlowLayout());

        JButton myButton = new JButton("I'm a JButton");
        final JTextField myTextField = new JTextField("I'm a JTextField");
        final JLabel myLabel = new JLabel("I'm a JLabel");

        myButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent event) {
                myLabel.setText(myTextField.getText());
            }
        });

        // continued in box ->
        frame.add(myButton);
        frame.add(myTextField);
        frame.add(myLabel);
        frame.pack();
        frame.setVisible(true);
    }
}
/**
 * This ActionListener object sets a JLabel to a textField's contents
 * @author bert
 */

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.*;

public class SetTextListener implements ActionListener {

    public SetTextListener(JTextField textField, JLabel label) {
        myLabel = label;
        myTextField = textField;
    }

    public void actionPerformed(ActionEvent event) {
        myLabel.setText(myTextField.getText());
    }

    private JLabel myLabel;
    private JTextField myTextField;
}
import javax.swing.*;
import java.awt.*;

public class GraphicsTester2 {
    public static void main(String[] args) {
        JFrame frame = new JFrame();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setLayout(new FlowLayout());

        JButton myButton = new JButton("I'm a JButton");
        JTextField myTextField = new JTextField("I'm a JTextField");
        JLabel myLabel = new JLabel("I'm a JLabel");

        myButton.addActionListener(
                new SetTextListener(myTextField, myLabel));

        frame.add(myButton);
        frame.add(myTextField);
        frame.add(myLabel);
        frame.pack();
        frame.setVisible(true);
    }
}
Timer

- Invisible Swing component that can call ActionListener based on time
  - `new Timer(int delay, ActionListener listener)`
  - `addActionListener(ActionListener listener)`
  - `start()`
  - `setRepeats(boolean b) // default true`
  - `setDelay(int delay) // milliseconds`
Why Interfaces?

• Interchangeable parts are essential in modern engineering
• Allows tools and parts to be used for various applications
• Without establishing standard interfaces, every part must be custom-built for each application
Interfaces of Screws

Flat

Philips
Designing Interfaces

• Parts of your code do the same thing to objects of similar classes
  • but don't want to combine the classes

• The similar classes can implement an interface, then consolidate redundant code to work with the interface type
public interface FlatHeadScrew
{
    public void turnClockWise(FlatHeadScrewDriver driver);
    public void turnCounterClockWise(FlatHeadScrewDriver driver);
}

public class WoodScrew implements FlatHeadScrew
{
    public void turnClockWise(FlatHeadScrewDriver driver)
    { /* ... */ }

    public void turnCounterClockWise(FlatHeadScrewDriver driver)
    { /* ... */ }
}
ArrayLists and LinkedLists

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

A → B → C → D
Review: Generics

• Classes with `<SomeType>` in their definition are **generic**
• Keyword inside the brackets is a placeholder for a type
• Objects are instantiated with a particular type, e.g., `ArrayList<Integer>`
List<T> Interface

• Includes methods:
  • boolean add(int index, T o)
  • Object get(int index)
  • boolean remove(int index)
  • int size()

• Implemented by
  ArrayList<T>, LinkedList<T>
Collections.sort()

- Collections.sort(List<T> list, Comparator<T> c);

- Sorts list according to Comparator c

- Comparator<T> objects define comparison metrics for types

- e.g., sort Rectangle2D.Double by:
  - Left edge, top edge, distance to (0,0)
Comparator<T> Interface

- int compare(T object1, T object2)
  - returns positive int if object1 > object2
  - returns negative if object1 < object2
  - returns zero if equal
Reusable Code

- Using the same code, we can sort a combination of

<table>
<thead>
<tr>
<th>Sort a</th>
<th>according to</th>
</tr>
</thead>
<tbody>
<tr>
<td>LinkedList</td>
<td>LeftEdgeComparator TopEdgeComparator DistFromOriginComparator</td>
</tr>
<tr>
<td>ArrayList</td>
<td></td>
</tr>
</tbody>
</table>
Polymorphism in Collections.sort()

- Polymorphism - ability to work with multiple shapes
- Collections.sort treats LinkedLists and ArrayLists as List objects
- If it did not, code would have to be specifically written for each kind of list
Collection Interface

- More general than List: a Collection stores a set of objects, but does not have to be **ordered**
- LinkedList, ArrayList, Stack, Queue
- Methods include: add(Object o), remove (Object o), boolean contains(Object o)
- Iterator iterator();
Iterator Interface

• An Iterator<T> lets you look at one element at a time from a Collection<T>

• boolean hasNext(), T next()

• Using Iterators, you can write code that doesn't know what kind of Collection you have
Iterators Preserve Encapsulation

• Iterator user doesn't know how the items are stored

• Iterating through linked list:
  • Do work on current node
  • Go to current.next()

• Need to know linked list structure, and private next() links
Interface Relationships

- Collections.sort() sorts object implementing List
- using an object that implements Comparator
- List extends Collection
- Collection includes iterator(), returns an object that implements Iterator
Anonymous Classes

• Anonymous ActionListener objects are of some class
• but that class is only used once
• Anonymous classes can lead to shorter code in these cases
• Can use final variables in local scope
Reading

- Horstmann Ch. 4
- Next class, Horstmann Ch. 5