CS1007: Object Oriented Design and Programming in Java

Lecture #11

Feb 21

Shlomo Hershkop
shlomo@cs.columbia.edu

1

Outline

- Interfaces
- Inheritance
- Graphics

- Reading: Chapter 4.10, 5-5.4.1
- Next time: 5.4.2-5.8

2

Announcements

- Midterm review on Thursday
  - Will go over study strategy
  - Will release sample exams
  - Will do quick overview at end of class
- Chapter 1 – 5.2 (page 179)
- Anything else we covered in class

3

From last time

- We discussed interface objects using a icon as an example
- Please take a look at the code from the book
- Understand it enough to be able to read interface code

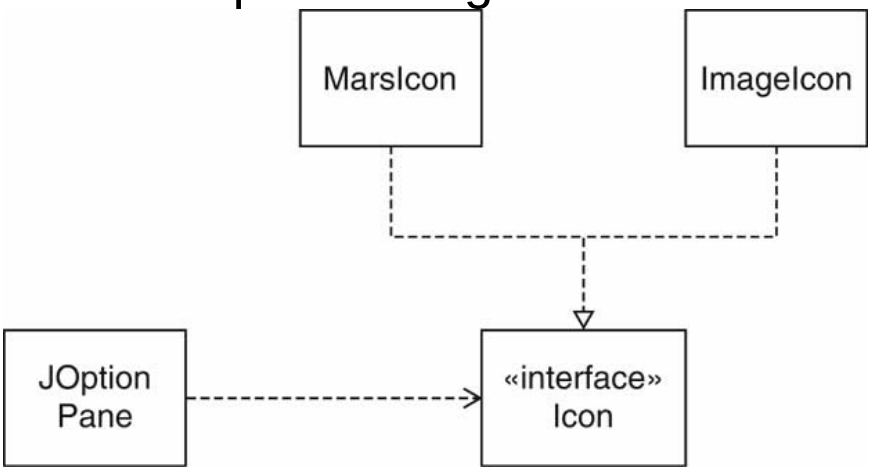
4

Using it

```
01: import javax.swing.*;
02:
03: public class IconTester
04: {
05:    public static void main(String[] args)
06:    {
07:       JOptionPane.showMessageDialog(
08:             null,
09:             "Hello, Mars!",
10:             "Message",
11:             JOptionPane.INFORMATION_MESSAGE,
12:             new MarsIcon(50));
13:       System.exit(0);
14:    }
15: }
16:
```

5

The Icon Interface Type and Implementing Classes



6

Fundamentals again

- Polymorphism
- Inheritance
- Functional inheritance
- Encapsulation

7

Polymorphism

- What is polymorphism?

8

Polymorphism

- Ability of objects to react differently to the same method

- Example:
  - `something.toString()`
    will depend on what the something is…

9

Polymorphism

- `public static void showMessageDialog(...Icon ourIcon)`
- `showMessageDialog` shows
  - icon
  - message
  - OK button

- `showMessageDialog` must compute size of dialog
- width = icon width + message size + blank size
- How do we know the icon width?

```
int width = ourIcon.getIconWidth();
```

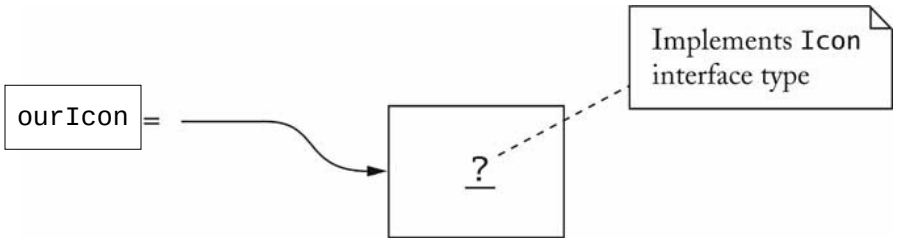
10

PolyMorphism

- showMessageDialog doesn't know which icon is passed
  - ImageIcon?
  - MarsIcon?
  - . . .?
- The actual type of anIcon is not `Icon`
- There are no objects of type `Icon`..WHY??

11

A Variable of Interface Type



12

So…

- Which getIconWidth method is called?
- Could be
  - MarsIcon.getIconWidth
  - ImageIcon.getIconWidth
  - . . .
- Depends on object to which anIcon reference points, e.g.

```
showMessageDialog(..., new MarsIcon(50))
```

- Polymorphism: Select different methods according to actual object type

13

Benefits

- Stronger OO Design
- Loose coupling
  - showMessageDialog decoupled from ImageIcon
  - Doesn't need to know about image processing
- Extensibility
  - Client can supply new icon types

14

Inheritance

- Is the process of extending a class to do something specific
  - Specializing a more general class
  - Extend usefulness of a specific class
  - Torture students as a teaching aid

15

Example

- BufferedReader class in java
  - Can read a single char
  - Can read a String
  - Can read n characters into a buffer
- Want to extend to allow us to read an Integer.

16

```
public class newBR extends BufferedReader
{
   public Integer readInt() throws…..

}
```

- Base class = super class = BufferedReader
- Derived class = subclass = newBR

- Can add variables, methods and will have all resources of super class

17

Redefining methods

- In subclass, can redefine methods
  - Called polymorphism
  - May not change simple return type
    - Exception derived returned types
    - New to java 5
  - Can not change final methods in parents class
  - Can make parent private methods, public (less restrictive only)

18

Overriding vs overloading

- Overriding:
  - When redefine method with exact arguments and return type in subclass
- Overloading:
  - Adding a method with the same name but new number of arguments
  - Result in 2 methods available in the subclass

19

Access rules

- Private variable in the base class are not accessible in the derived class.
- So how do we manipulate them?

20

Encapsulation

- `protected` modifier:
  - Allows access to variable/method only in same, derived and package classes.
  - Weak protection compared to public/private modifiers

21

- A Ferrari and mini-van are both car types
- What makes them different?
- How are they the same?

22

Inheritance II

- Ability in a programming language to extend a base object to specialize in some task

- Do we want to go really fast?
- Do we want to hold family + dog ?

23

Quick question?

- Why not just copy paste specialized code?

- i.e. completely build mini-van and then copy and chip away everything which isn't a Ferrari ?

24

Use of base object

- Can talk about everything as base object (limited but useful)
- If everything is copy-paste upgrade or bug will have a domino effect
- Scaling…we can work on much larger set of objects

25

Iterators

- Iterator is an object which allows you to step through (and maybe modify) a collection of objects in some order without knowing the underlying representation.
- Allows loose coupling between collections and users.
- Simplest example:
  - Step through an array with a counter
  - Problem: controlling multiple access from different objects

26

Iterator<T> interface

- standard Iterator interface type

```
public interface
  Iterator<LineItem>
{
    boolean hasNext();
    LineItem next();
    void remove();
}
```

27

Danger

- Should next() return
  - Value
  - Reference

28

Comparing two objects

- What is the best way to compare two different objects?

29

The `Comparable` Interface Type

- Collections has static sort method:

```
ArrayList<E> a = . . .
Collections.sort(a);
```

30

- Objects in list must implement the `Comparable` interface type

```
public interface Comparable<T>
{
 int compareTo(T other);
}
```

- Notice the generic type T

31

- `object1.compareTo(object2)` returns
  - Negative number if `object1` less than `object2`
  - 0 if objects identical
  - Positive number if `object1` greater than `object2`
- sort method compares and rearranges elements

```
if (object1.compareTo(object2) > 0) . . .
```

- `String` class implements `Comparable<String>` interface type: lexicographic (dictionary) order

32

Some code

```
01: /**
02:    A country with a name and area.
03: */
04: public class Country implements Comparable<Country>
05: {
06:    /**
07:       Constructs a country.
08:       @param aName the name of the country
09:       @param anArea the area of the country
10:    */
11:    public Country(String aName, double anArea)
12:    {
13:       name = aName;
14:       area = anArea;
15:    }
16:
```


testing

```
01: import java.util.*;
02:
03: public class CountrySortTester
04: {
05:    public static void main(String[] args)
06:    {
07:       ArrayList<Country> countries = new
   ArrayList<Country>();
08:       countries.add(new Country("Uruguay", 176220));
09:       countries.add(new Country("Thailand", 514000));
10:       countries.add(new Country("Belgium", 30510));
11:
12:       Collections.sort(countries);
13:       // Now the array list is sorted by area
14:       for (Country c : countries)
15:          System.out.println(c.getName() + " " + c.getArea());
16:    }
17: }
```

The Comparator interface type

- How can we sort countries by name?
- Can't implement Comparable twice!
- Comparator interface type gives added flexibility

```
public interface Comparator<T>
{
   int compare(T obj1, T obj2);
}
```

- Pass comparator object to sort:

```
Collections.sort(list, comp);
```

35

- Comparator object is a function object
- This particular comparator object has no state
- State can be useful, e.g. flag to sort in ascending or descending order

36

The Comparator interface type

```
public class CountryComparatorByName implements
   Comparator<Country>
{
public int compare(Country country1, Country
   country2)
{
return
   country1.getName().compareTo(country2.getName());
}

}
```

37

```
public class ComparatorTester
{
    public static void main(String[] args)
    {
    ArrayList<Country> countries = new ArrayList<Country>();
    countries.add(new Country("Uruguay", 176220));
    countries.add(new Country("Thailand", 514000));
    countries.add(new Country("Belgium", 30510));
    Comparator<Country>  comp = new CountryComparatorByName();
    Collections.sort(countries, comp);
      // Now the array list is sorted by area
       for (Country c : countries)
      System.out.println(c.getName() + " " + c.getArea());
  }
 }
```

38

Anonymous Classes

- In general No need to name objects that are used only once

```
Collections.sort(countries,
   new CountryComparatorByName());


Comparator<Country> comp = new
   Comparator<Country>()
   {
      public int compare(Country country1, Country
  country2)
      {
         return
  country1.getName().compareTo(country2.getName());
      }
   };
```

39

- anonymous new expression:
  - defines anonymous class that implements Comparator
  - defines compare method of that class
  - constructs one object of that class
- Cryptic syntax for very useful feature

40

Example

```
modeling.addActionListener(
  new ActionListener() {
     public void actionPerformed(ActionEvent evt) {
                try {
                    if (update_all_labels == true) {
```


`public interface ActionListener extends EventListener`

- The listener interface for receiving action events. The class that is interested in processing an action event implements this interface, and the object created with that class is registered with a component, using the component's addActionListener method. When the action event occurs, that object's actionPerformed method is invoked.

`void actionPerformed(ActionEvent e)`

Java Frame Class

- Frame window has decorations
  - title bar
  - close box
  - provided by windowing system

```
JFrame frame = new JFrame();
frame.pack();
frame.setDefaultCloseOperation(JFrame
  .EXIT_ON_CLOSE);
frame.setVisible(true);
```

43

Layout Managers

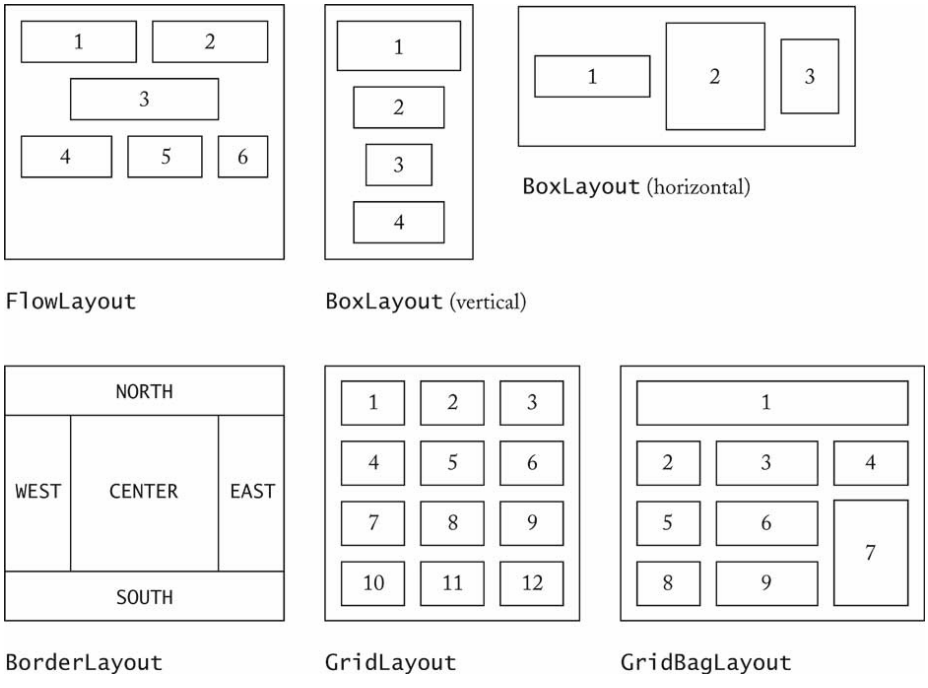
- User interfaces made up of components
- Components placed in containers
- Container needs to arrange components
- Swing doesn't use hard-coded pixel coordinates
- Advantages:
  - Can switch "look and feel"
  - Can internationalize strings
- Layout manager controls arrangement

44

Basic Managers

- `FlowLayout`:
  - left to right, start new row when full
- `BoxLayout`:
  - left to right or top to bottom
- `BorderLayout`:
  - 5 areas, Center, North, South, East, West
- `GridLayout`:
  - grid, all components have same size
- `GridBagLayout`:
  - complex, like HTML table

45



Adding components

- Construct components

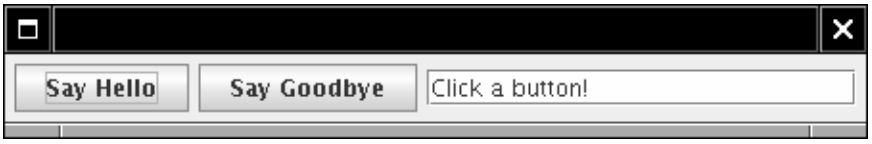
```
JButton helloButton = new JButton("Say Hello");
```

- Set frame layout

```
frame.setLayout(new FlowLayout());
```

- Add components to frame

```
frame.add(helloButton);
```



47

```
09      JFrame frame = new JFrame();
10:     JButton helloButton = new JButton("Say Hello");
11:     JButton goodbyeButton = new JButton("Say Goodbye");
12:
13:     final int FIELD_WIDTH = 20;
14:     JTextField textField = new JTextField(FIELD_WIDTH);
15:     textField.setText("Click a button!");
16:
17:     frame.setLayout(new FlowLayout());
18:
19:     frame.add(helloButton);
20:     frame.add(goodbyeButton);
21:     frame.add(textField);
22:
23:     frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
24:     frame.pack();
25:     frame.setVisible(true);
```

48

Reacting to user input

- Previous code buttons don't have any effect
- Add listener object(s) to button
- Belong to class implementing ActionListener interface type

```
public interface ActionListener
{
   int actionPerformed(ActionEvent event);
}
```

- Listeners are notified when button is clicked

49

Actions

- Add action code into actionPerformed method
- Gloss over routine code

```
helloButton.addActionListener(new
   ActionListener()
   {
      public void actionPerformed(ActionEvent event)
      {
          textField.setText("Hello, World");
     }
   });
```

- When button is clicked, text field is set

50

Back to scopes

- Remarkable: Inner class can access variables from enclosing scope
  e.g. textField
- Can access enclosing instance fields, local variables
- Local variables must be marked final

```
final JTextField textField = ...;
```

51

How to react

- Constructor attaches listener:

```
helloButton.addActionListener(listener);
```

- Button remembers all listeners
- When button clicked, button notifies listeners

```
listener.actionPerformed(event);
```

- Listener sets text of text field

```
textField.setText("Hello, World!");
```

52

Related work

- Write helper method that constructs objects
- Pass variable information as parameters
- Declare parameters final

```
public static ActionListener createGreetingButtonListener(
   final String message)
{
 return new
    ActionListener()
    {
       public void actionPerformed(ActionEvent event)
       {
          textField.setText(message);
       }
    };
}
```

53

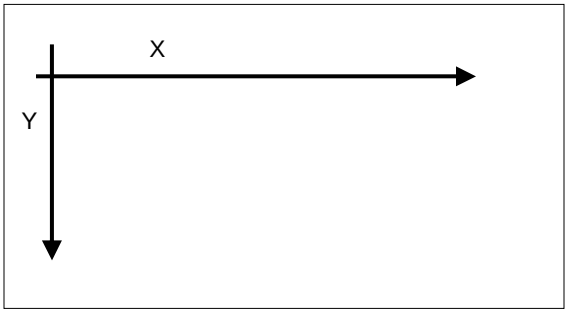
Low Level Java Drawing

- All drawing done on graphic context objects
- Area to draw on
- It is instantiated to the paint() or update() method

54

Coordinate System

- `(0,0)` on top left corner
- X increase to right
- Y increase downwards
- `(0,object.getwidth() )` is upper right corner



55

Drawing Shapes

- paintIcon method receives graphics context of type Graphics
- Actually a Graphics2D object in modern Java versions

```
public void paintIcon(Component c, Graphics g, int
  x, int y)
{
 Graphics2D g2 = (Graphics2D)g;
 . . .
}
```

- Can draw any object that implements Shape interface

```
Shape s = . . .;
g2.draw(s);
```

56

Next Time

- Do Reading (through 5.4)
- We will do low level paint and review for midterm next class

57