CS1007: Object Oriented Design and Programming in Java

Lecture #8
Oct 4
Shlomo Hershkop
shlomo@cs.columbia.edu

Outline

• Unit Testing
• Sorting Algorithms
• Polymorphism
• Interfaces
• Basic graphics
• Layout managers
• Anonymous Classes

Feedback

• From lab experience
  – Please see us if you need more help with anything in Java
  – Any comments?

HW2

• Help you work with UML and design specs
• More practical java work

• Out Today
• Programming Due: Oct 17 11:59pm.
• Written Part Due: Class Oct 18

• Note: For those who missed handing in written part 1, see TA
3 proposals in Testing

• Don’t:
  – Hope to be bought out before anyone realizes
    • Requirements: ticket to get out of town once they realize
  • Assemble everything and then test
    – See above
  • Test individual components before integrating
    – Ability to only test each piece separately, but allows
      you to work out many bugs early on.

Unit Testing

• Unit test = test of a single class
• Design test cases during implementation
• Run tests after every implementation change
• When you find a bug, add a test case that catches it

JUnit

• [http://www.junit.org/](http://www.junit.org/)
• Framework for running tests on your code
• Need to plan out tests not random
  – Code a special class to run tests on your
    other classes
  – Will explore this in next lab
• Need to realize advantages and disadvantages of this framework
  – It is only a TOOL!
JUnit

• Test class name = tested class name + Test
• Test methods start with test

```java
import junit.framework.*;
public class DayTest extends TestCase {
    public void testAdd() { ... }
    public void testDaysBetween() { ... }
    ... }
```

JUnit

• Each test case ends with assertion
• Test framework catches assertion failures

```java
public void testAdd() {
    Day d1 = new Day(1970, 1, 1);
    int n = 1000;
    Day d2 = d1.addDays(n);
    assertTrue(d2.daysFrom(d1) == n);
}
```

Shift gears

• Sorting integers

Sort

• A sorting algorithm takes an unordered array of objects and returns an array of objects in ascending or descending order.
• Ascending is defined by the programmer or object type.
• We can use integers as an example
  – Number line defines order
### Bubble Sort

- The bubble sort works by comparing each item in the list with the item next to it, and swapping them if required.
- The algorithm repeats this process until it makes a pass all the way through the list without swapping any items (in other words, all items are in the correct order).
- This causes larger values to "bubble" to the end of the list while smaller values "sink" towards the beginning of the list.

### Algorithm

- **Input:** List of integers $X_0, X_1, \ldots, X_n$
- **Output:** Ascending order
- 1. $i = 0$, swap =0
- 2. If $X_i > X_{i+1}$ swap $X_i$ and $X_{i+1}$ swap =1
- 3. $i = i + 1$
- 4. If $i < n$ goto step 2
- 5. If swap > 0 goto step 1
- 6. Return sorted list

### Pseudo code

```cpp
void bubbleSort(int numbers[], int array_size)
{
    int i, j, temp;
    for (i = (array_size - 1); i >= 0; i--)
    {
        for (j = 1; j <= i; j++)
        {
            if (numbers[j-1] > numbers[j])
            {
                temp = numbers[j-1];
                numbers[j-1] = numbers[j];
                numbers[j] = temp;
            }
        }
    }
}
```

### Example

- 6 9 3 1 8
How to analyze this algorithm

- Will be taught in data structures
- Enough to know:
  - Slowest sort in general
  - Run time:
    - Anyone know how many comparisons required?
  - Advantages:
    - For small number of items, ok to use
    - Simple

Polymorphism

- Definition:
  - Programming language’s ability to process objects independent of their data type or class with the same set of code
  - i.e. example draw a shape on the screen
    - Triangle
    - Square
    - Circle
  - Perfect for Object Oriented design

Interface

- Java interface defines methods which need to be implemented by anyone implementing the interface
- No implementation
- Implementing class must supply implementation of all methods

The Icon Interface Type

```java
public interface Icon {
    int getIconWidth();
    int getIconHeight();
    void paintIcon(Component c, Graphics g, int x, int y)
}
```
Example

- Use JOptionPane to display message:
  `JOptionPane.showMessageDialog(null, "Hello, World!");`
- Note icon to the left

![Hello, World!]

images

- Can specify arbitrary image file
  `JOptionPane.showMessageDialog(
      null, 
      "Hello, World!", 
      "Message", 
      JOptionPane.INFORMATION_MESSAGE, 
      new ImageIcon("globe.gif"));`

Displaying an Image

- What if we don't want to generate an image file?
- Fortunately, can use any class that implements Icon interface type
- ImageIcon is one such class
- Easy to supply your own class

![Hello, Mars!]

Marsicon.java

```java
import java.awt.*;
import java.awt.geom.*;
import javax.swing.*;

/**
 * An icon that has the shape of the planet Mars.
 */
public class MarsIcon implements Icon
{
    /** Constructs a Mars icon of a given size. */
    public MarsIcon(int aSize)
    {
        size = aSize;
    }

    public int getIconWidth()
    {
        return size;
    }

    public int getIconHeight()
    {
        return size;
    }

    public void paintIcon(Component c, Graphics g, int x, int y)
    {
        Graphics2D g2 = (Graphics2D) g;
        Ellipse2D.Double planet = new Ellipse2D.Double(x, y, size, size);
        g2.setColor(Color.RED);
        g2.fill(planet);
    }

    private int size;
}
```
Using it

```
import javax.swing.*;
public class IconTester {
    public static void main(String[] args) {
        JOptionPane.showMessageDialog(null, "Hello, Mars!", "Message", JOptionPane.INFORMATION_MESSAGE, new MarsIcon(50));
        System.exit(0);
    }
}
```

The Icon Interface Type and Implementing Classes

Polymorphism
- public static void showMessageDialog(...)Icon anIcon
- 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?
  ```java
  int width = anIcon.getIconWidth();
  ```

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
- anIcon belongs to a class that implements Icon
- That class defines a getIconWidth method
A Variable of Interface Type

anIcon = new Icon()

Implements Icon interface type

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

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

The Comparable Interface Type

• Collections has static sort method:
  ArrayList<E> a = ...;
  Collections.sort(a);
• Objects in list must implement the Comparable interface type
  public interface Comparable<T>
  {
    int compareTo(T other);
  }
• Interface is parameterized (like ArrayList)
• Type parameter is type of other
• `object1.compareTo(object2)` returns
  - Negative number if `object1` less than `object2`
  - 0 if `object1` 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

• `Country` class: compare countries by area

```java
import java.util.*;

class CountrySortTester {
    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));
        Collections.sort(countries);
        // Now the array list is sorted by area
        for (Country c : countries)
            System.out.println(c.getName() + " * " + c.getArea());
    }
}
```

The Comparator interface type

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

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

• Pass comparator object to sort:

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

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

```java
import java.util.*;
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());
        }
    }
}
```

Anonymous Classes

• No need to name objects that are used only once
  Collections.sort(countries,
                 new CountryComparatorByName());

• No need to name classes that are used only once
  Comparator<Country> comp = new Comparator<Country>() {
      public int compare(Country country1, Country country2) {
          return country1.getName().compareTo(country2.getName());
      }
  };
Anonymous Classes

- Commonly used in factory methods:
  ```java
  public static Comparator<Country> comparatorByName() {
    return new Comparator<Country>() {
      public int compare(Country country1, Country country2) {
        // ...}
    };
  }
  ```
  ```java
  Collections.sort(a, Country.comparatorByName());
  ```
- Neat arrangement if multiple comparators make sense
  (by name, by area, ...

Next Time

- Continue reading
- Meet Thursday in Class
- Will continue with more analysis of the graphics components of awt.
- Continue reading book