Object Oriented Programming and Design in Java

Session 15

Instructor: Bert Huang

Announcements

- Homework 3 out. Due **Monday**, Apr. 5th
- Office hour change

Sun	Mon	Tue	Wed	Thu	Fri
John 1-3	Class 11-12:15		Class 11-12:15 Bert 2-4 Yipeng 4-6		Lauren 11-1

Review

- Generics
  - Generic types
  - Generic methods
  - Type bounds and wildcards
  - Type erasure

Today’s Plan

- Frameworks
- The Applet Framework
- The Collections Framework

Frameworks

- Sets of cooperating classes that implement mechanisms essential for a particular problem domain
- Application frameworks implement services common to a certain type of application
- Programmers subclass some framework classes and implement additional functionality specific to the target application

Packages

- Typically, framework classes can be stored in packages
- javax.swing.*, java.awt.*, java.applet.*
- Allows clients to import easily

Notes on Packages

- Not hierarchical (java.awt does not include java.awt.geom)
- Naming convention is to use reverse-order internet domain name:
  - edu.columbia
  - then use whatever convention your organization prefers (e.g., UNI)

Inversion of Control

- Most of the work is done by the framework, as in the template method and strategy patterns
- The programmer doesn’t need to be concerned with control flow, just the specifics of the applications

Swing and AWT

- Frameworks allow graphical interfaces
- Frameworks handle communication with operating system, display and input devices
- Clients design interface, decide what to do on user input

Applets

- Framework for GUI programs for websites
- Framework handles communication with web browser
  - parameter retrieval
  - starting and stopping

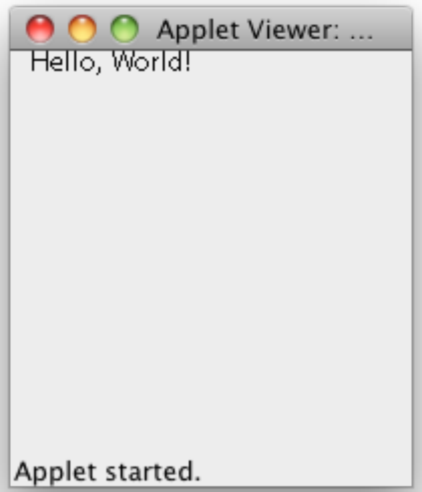
Hello World Applet

```
import java.applet.*;
import java.awt.*;


public class HelloWorldApplet extends Applet
{

   public void paint(Graphics g)
   {
      g.drawString("Hello, World!", 10, 10);
   }

}
```



```
Hello, World!
```

```
Applet started.
```

Applet Methods

- init() // initializes data
- start() // called when applet loaded and
  // when user restores browser window
- stop() // called when user leaves
  // browser window (minimize, tabs)
- destroy() // called when browser exits
- paint() // called when applet window needs
  // repainting.

BannerApplet

- The Applet Framework allows web sites to embed applets and pass parameters using HTML

```html
<applet code="BannerApplet.class" width="300" height="100">
<param name="message" value="Hello, World!"/>
<param name="fontname" value="Serif"/>
<param name="fontsize" value="64"/>
<param name="delay" value="10"/>
</applet>
```

```java
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
import java.awt.font.*;
import java.awt.geom.*;
import javax.swing.*;

public class BannerApplet extends Applet {
   public void init() {
      message = getParameter("message");
      String fontname = getParameter("fontname");
      int fontsize = Integer.parseInt(getParameter("fontsize"));
      delay = Integer.parseInt(getParameter("delay"));
      font = new Font(fontname, Font.PLAIN, fontsize);
      Graphics2D g2 = (Graphics2D) getGraphics();
      FontRenderContext context = g2.getFontRenderContext();
      bounds = font.getStringBounds(message, context);

      timer = new Timer(delay, new ActionListener() {
            public void actionPerformed(ActionEvent event) {
               start--;
               if (start + bounds.getWidth() < 0)
                  start = getWidth();
```

```java
public class BannerApplet extends Applet {
   public void init() {
      message = getParameter("message");
      String fontname = getParameter("fontname");
      int fontsize = Integer.parseInt(getParameter("fontsize"));
      delay = Integer.parseInt(getParameter("delay"));
      font = new Font(fontname, Font.PLAIN, fontsize);
      Graphics2D g2 = (Graphics2D) getGraphics();
      FontRenderContext context = g2.getFontRenderContext();
      bounds = font.getStringBounds(message, context);

      timer = new Timer(delay, new ActionListener() {
            public void actionPerformed(ActionEvent event) {
               start--;
               if (start + bounds.getWidth() < 0)
                  start = getWidth();
               repaint();
            }
         });
   }

   public void start() { timer.start(); }

   public void stop() { timer.stop(); }
```

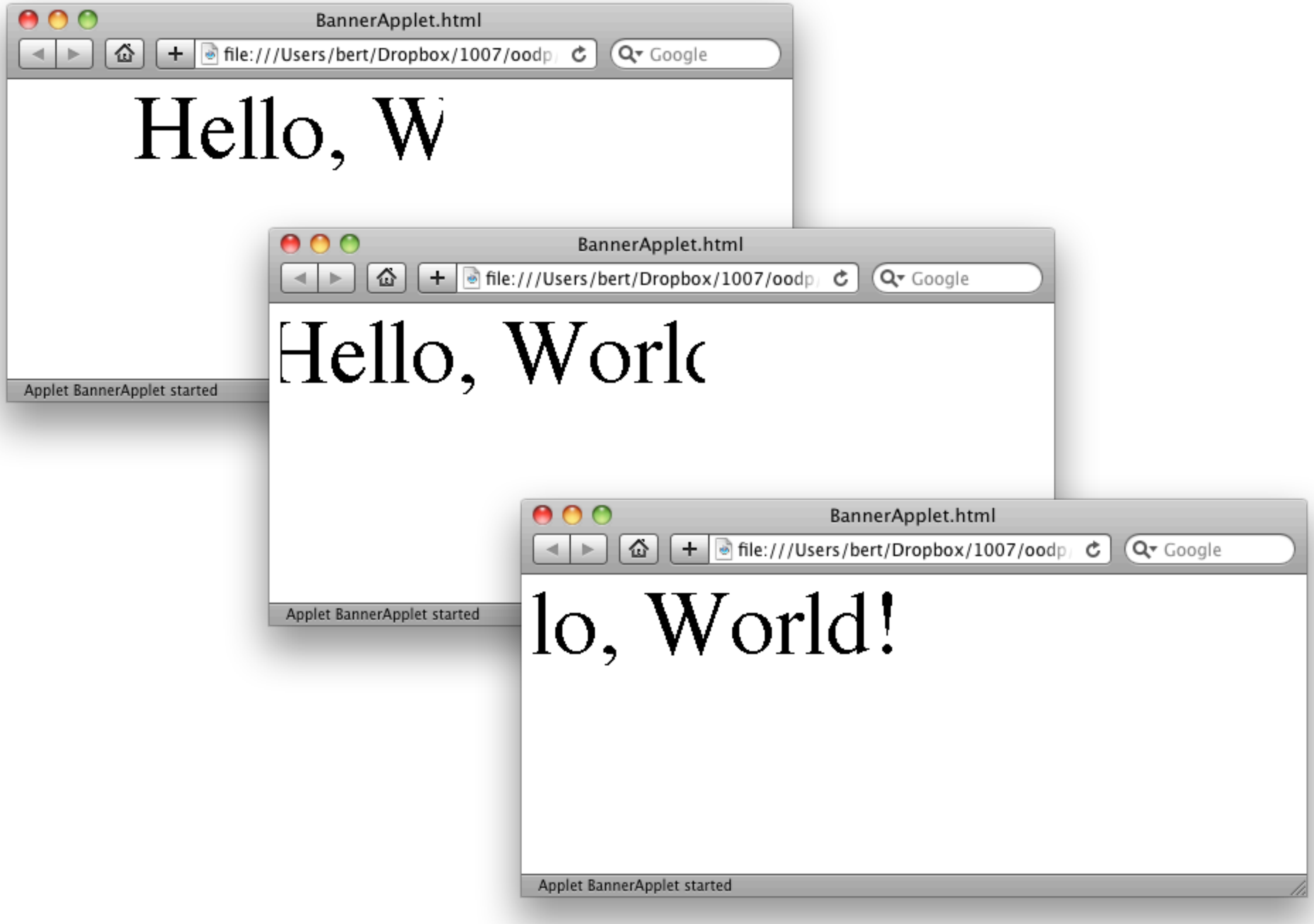
```
                if (start + bounds.getWidth() < 0)
                   start = getWidth();
                repaint();
             }
          });
    }

    public void start() { timer.start(); }

    public void stop() { timer.stop(); }

    public void paint(Graphics g) {
       g.setFont(font);
       g.drawString(message, start, (int) -bounds.getY());
    }

    private Timer timer;
    private int start;
    private int delay;
    private String message;
    private Font font;
    private Rectangle2D bounds;
 }
```


BannerApplet.html
file:///Users/bert/Dropbox/1007/oodp
Google

```
Hello, W
```

Applet BannerApplet started

BannerApplet.html
file:///Users/bert/Dropbox/1007/oodp
Google

```
Hello, Worl
```

Applet BannerApplet started

BannerApplet.html
file:///Users/bert/Dropbox/1007/oodp
Google

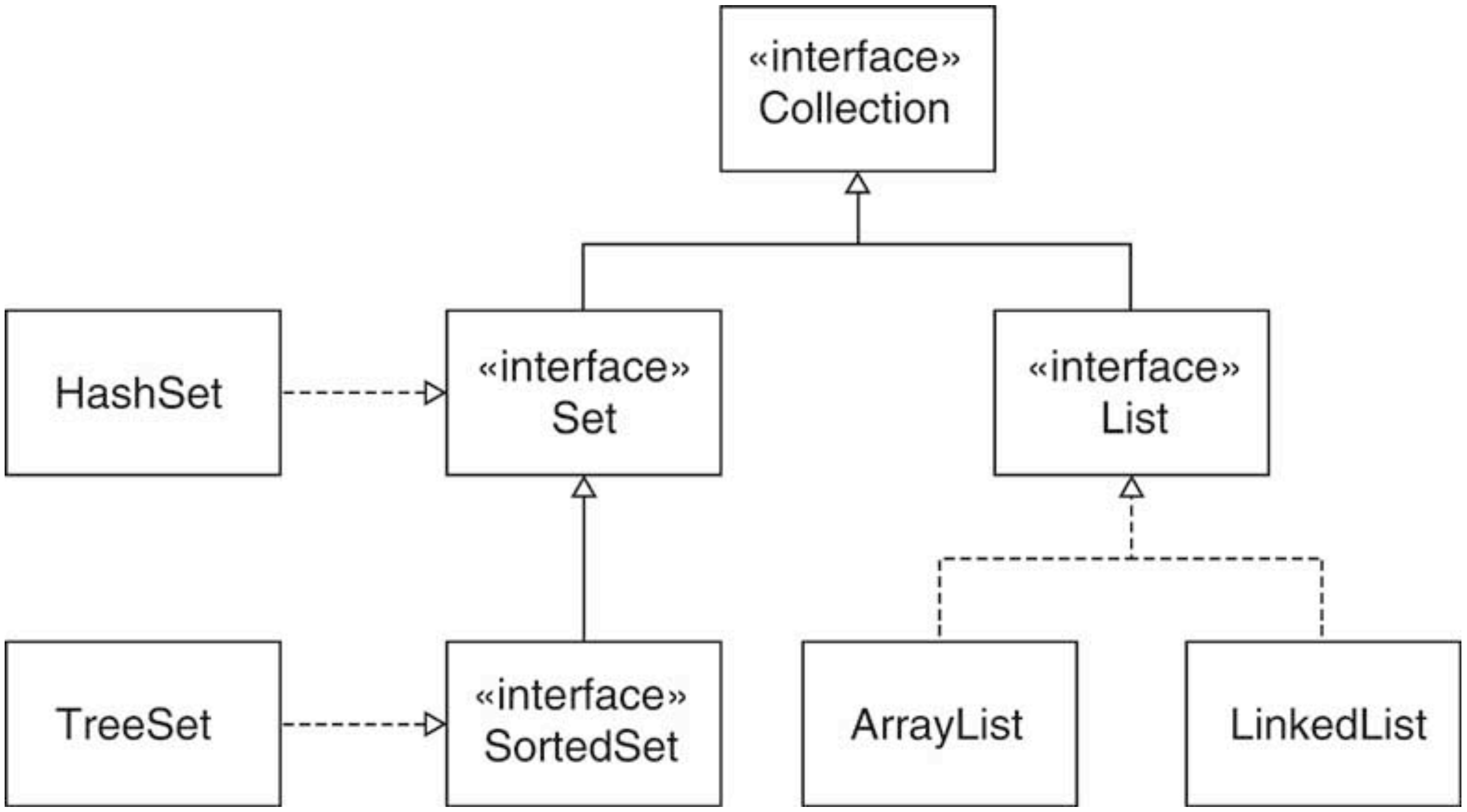
```
lo, World!
```

Applet BannerApplet started

Collections

- Framework for aggregating data structures, such as Lists and Sets
- Includes various built-in classes such as ArrayList, LinkedList, HashSet
- But allows easy construction of custom data structures

Collections Class Diagram



Collection<E> Interface

- `boolean add(E o)`
- `boolean addAll( Collection<? extends E> c)`
- `void clear()`
- `boolean contains(Object o)`
- `boolean containsAll (Collection<?> c)`
- `boolean equals(Object o)`
- `int hashCode()`
- `boolean isEmpty()`
- `Iterator<E> iterator()`
- `boolean remove(Object o)`
- `boolean removeAll (Collection<?> c)`
- `boolean retainAll (Collection<?> c)`
- `int size()`
- `Object[] toArray()`
- `<T> T[] toArray(T[] a)`

AbstractCollection

- Uses template-method pattern to define all Collection methods in terms of each other
- Client needs only to implement abstract `int size()` and `Iterator<E> iterator()`

-
  ```
  public Object[] toArray()
  {
      Object[] result = new Object[size()];
      Iterator<E> e = iterator();
      for (int i=0; e.hasNext(); i++)
          result[i] = e.next();
      return result;
  }
  ```

The Set<E> Interface

- The Set interface extends Collection, but adds no more methods
- Conceptually, Sets are a subclass of collections, so designers decided to make separate subinterface
- No duplicates, no order information

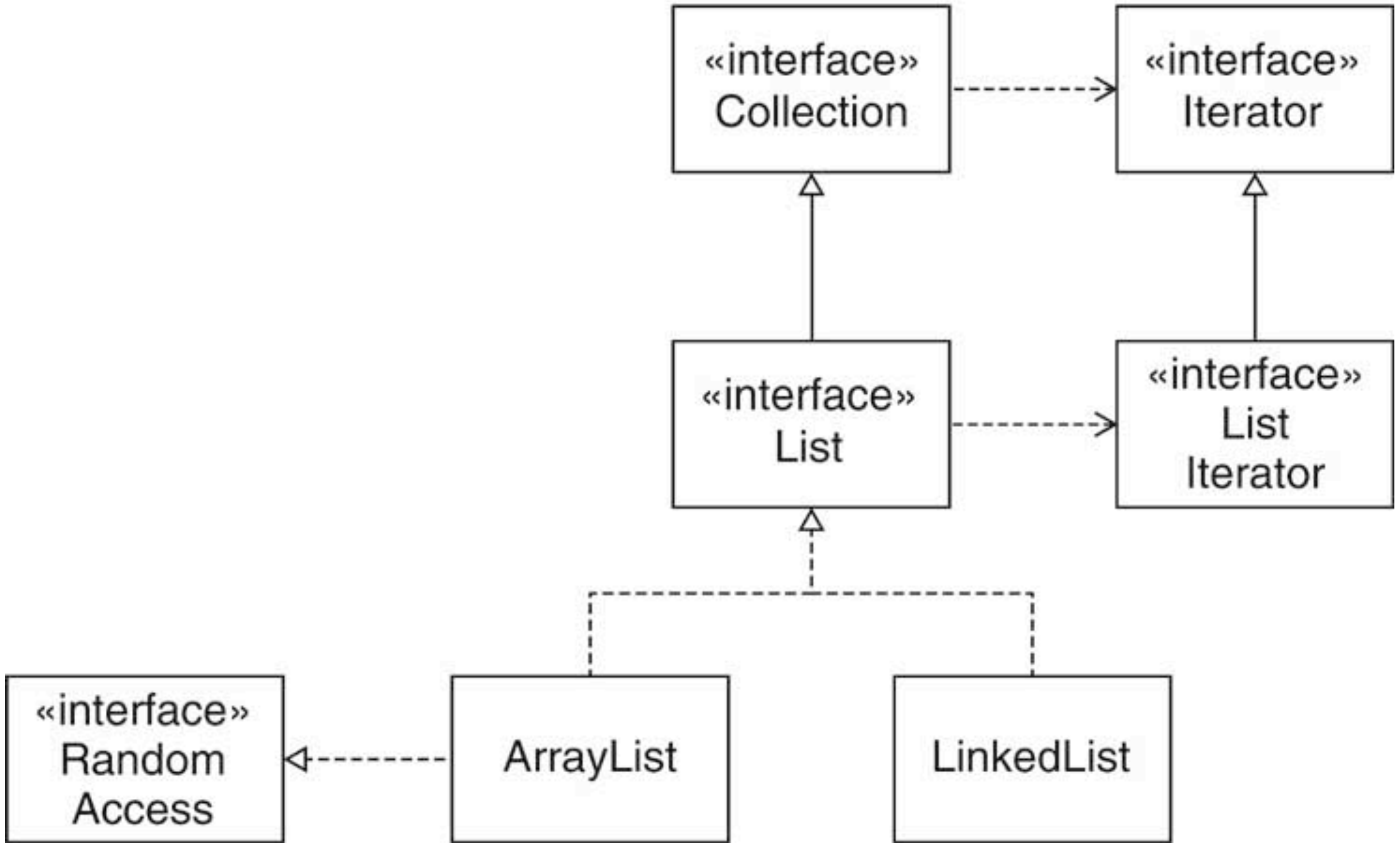
The List<E> Interface

- Extends Collection, adding the following
  - `void add(int index, E element)`
  - `boolean addAll(int index, Collection<? extends E> c)`
  - `E get(int index)`
  - `int indexOf(Object o)`
  - `ListIterator<E> listIterator()`
  - `ListIterator<E> listIterator(int index)`
  - `E remove(int index)`
  - `E set(int index, E element)`
  - `List<E> subList(int fromIndex, int toIndex)`

ListIterator<E> Interface

- Iterator<E>:
  - `boolean hasNext()`
  - `E next()`
  - `void remove()`
- ListIterator<E> adds
  - `int nextIndex()`
  - `int previousIndex()`
  - `boolean hasPrevious()`
  - `E previous`
  - `void add(E obj)`
  - `void set(E obj)`

Random Access



Reading

- Horstmann 8.1-8.3
- http://java.sun.com/docs/books/tutorial/collections/index.html