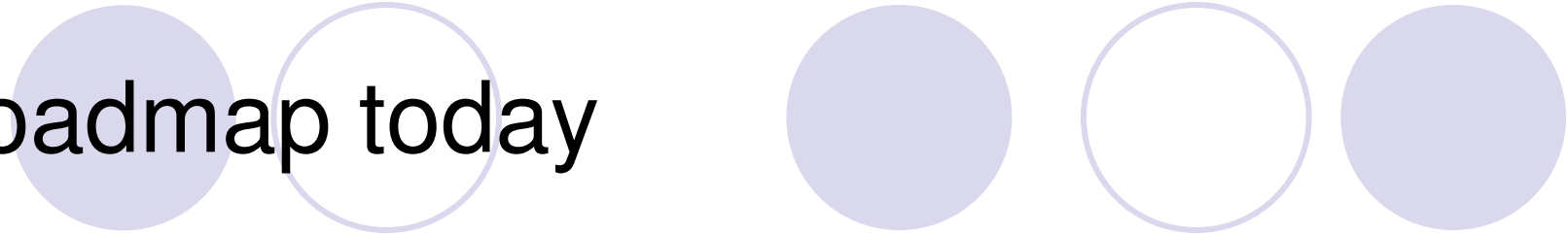
CS3101-3
Programming Language - JAVA

Fall 2004

Oct.13rd

Roadmap today



- Review
- Inner class
- Multi-threading
- Graphics

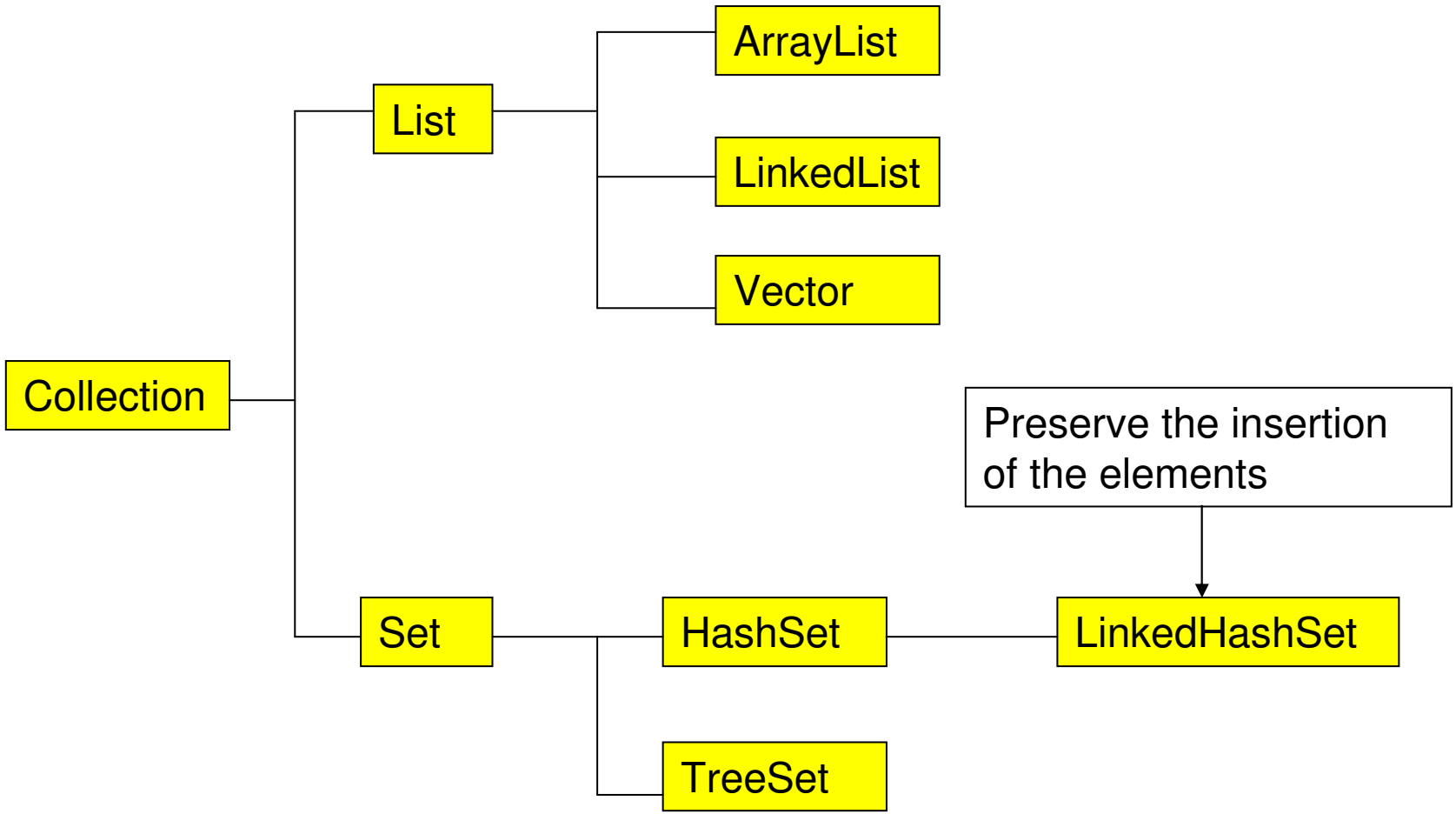
Containers

- Hold a group of objects
- Significantly increase your programming power
- All perform bound checking
- array: efficient, can hold primitives
- Collection: a group of individual elements
  - List, Set
- Map: a group of key-value object pairs
  - HashMap
- Misleading: sometimes the whole container libraries are also called collection classes

Collection: hold one item at each location

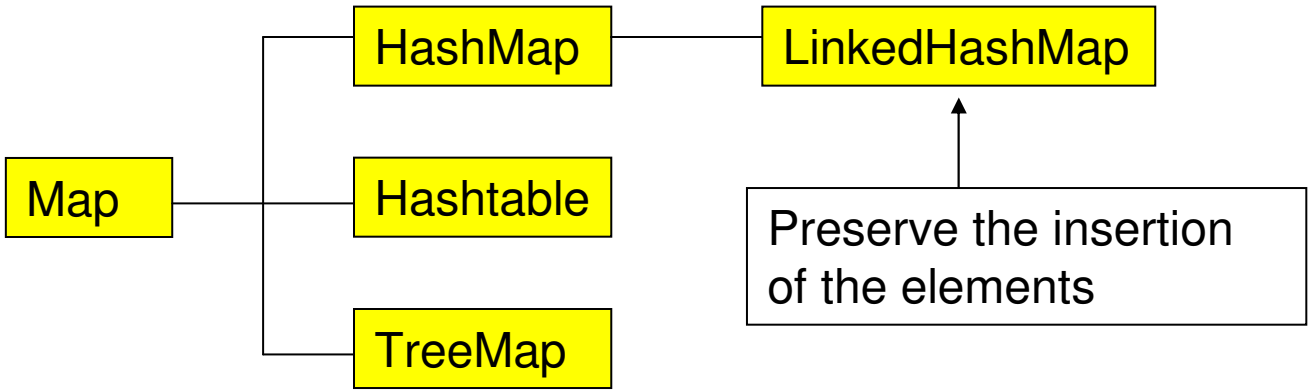
List: items in order

Set: no duplicates, no ordering



Map: key-value pairs, fast retrieval

no duplicate keys, no ordering



Disadvantages of container

- Cannot hold primitives
  - Have to wrap it
- Lose type information when put object into container
  - Everything is just Object type once in container
- Have to do cast when get it out
  - You need to remember what's inside
- Java do run time type check
  - ClassCastException

Iterator object

- Access method regardless of the underlying structure
- Generic programming
  - Can change underlying structure easily
- “light-weight” object
  - Cheap to create
- Can move in only one direction

Iterator constraints

- Container.**iterator**() returns you an Iterator, which is ready to return the first element in the sequence on your first call to **next**()
- Get the next object in the sequence with **next**()
- Set there are more objects in the sequence with **hasNext**()
- Remove the last element returned by the iterator with **remove**()
- Example: revisit CatsAndDogs.java

Java I/O: lowest level abstraction

- InputStream/OutputStream class
  - Byte-oriented
  - read() return the byte got read
  - write(int b) writes one byte out
- Can obtain from console, file, or socket
- Handle them in essentially the same way
- Write your code to work with Streams and won't care if it's talking to a file or a system on the other side of the world

Serialization

- An important feature of Java
- Convert an object into a stream of byte, and can later deserialize it into a copy of the original object
- Takes care of reassembling objects
- Need to cast type when read back
- Any object as long as it ***implements Serializable interface***
  - No method inside

Java networking

- Java.net.
- Socket, ServerSocket
- Grab the associated streams
  - InputStream is = s.getInputStream();
  - OutputStream os = s.getOutputStream();
- URL class
  - InputStream is = u.openStream();

Inner class

- Our internal implementation of a class is private
- Information hiding
  - Outside users don’t know what we are doing to provide our services
  - We can change our implementation without affecting others
- Private methods, fields

Inner class

- What if we want to use classes for our internal implementation?
  - **Inner classes!**
  - **A class defined inside another class**
- Similar to the methods or fields of class
  - Can be static, public, private
- Can be private so no other class can use it
- Usable by enclosing class

Example

- InnerClassTest.java

Why use inner class?

- Inner class can access to all the elements of the enclosing class
- Inner classes can be hidden from other classes by using private
- Anonymous inner classes are handy sometimes
- Convenient when you are writing event-driven programs

Anonymous class

- A local class without a name
- Defined and instantiated in a single *expression* using the **new** operator.

```
public void start (){
    ActionListener listener = new ActionListener{
            public void actionPerformed(ActionEvent e){
                    System.out.println("get the event "+e);
            }
   }; //Need to give a ; here, since it's an expression
}
```

When to use anonymous class?

- The class has a short body
- Only one instance of the class is needed
- The class is used right after it is defined
- The name of the class does not make your code any easier to understand

Process

- Programs often need to be doing multiple things at once
  - Talking on a network conn, process the data, listen to input, …, etc
- A process is a heavyweight OS-level unit of work
  - OS will assign piece of processor time to each processes, and take away the CPU when time is up
  - Processes are protected from each other

Threads

- Threads are lightweight, application-level form of multi-tasking
- Also preemptive (when time is up, switch to another one)
- But not protected: all can see the same memory
  - Which means much faster communication than classic “interprocess communication (IPC)”

All Java programs are MT

- You have the main() thread, and
- The garbage collector thread
- And any other thread by yourself

Creating a thread

- 1. Extend Thread class and override run()

  ```
  class Mythread extends Thread{…}
  Thread t = new MyThread();
  ```

- 2. Implement Runnable (have a run() method)

  ```
  class myRunnable implements Runnable { … }
  Thread t = new Thread(new myRunnable());
  ```

- Then start it (do NOT call run() directly)

  ```
  t.start();
  ```

- Threads are better than Runnables, but you may already be extending another class

Example

- SimpleThread.java
- The results are random, since 3 threads are running at the same time.

Example

- Multi-thread Network Server

Problems

- What is one thread is traversing a list while another thread is modifying it?
- Can easily reach inconsistent states
- Even a simple increment or decrement can cause problems
- Only one thread is running at a time, but
  - We don’t know which will be the next
  - We don’t know when it will be interrupted
  - All decided by CPU

More problems

- Debugging multithread code is a nightmare!
- We need ***synchronization*** mechanisms for the ***critical sections***

Solution

- Java is one of the first widespread languages to provide synchronization primitives at the language level
- Writing a correct multithreading program in Java is much easier than in many other languages
  - Which is not to say that it's easy!
- Java runtime has list of runnable threads

How to synchronize

- Need some “lock” so that only one thread can hold at one time
- Only the thread holding the lock can enter the critical section
- When done or waiting, give up the lock
- Another thread requests the lock has to wait until the first one is done and release the lock
- Thus, we can guarantee that only one thread can access the same data structure at any time
- So, what’s a lock?

Instance synchronization

- Answer: any object in Java can function as a lock
- If some data structure needs to be protected, one generally uses the instance itself as the lock

Synchronized keyword

```
public void swap (Object[] array, int index1, int index2){
        synchronized(array){
                Object temp = array[index1];
                array[index1]=array[index2];
                array[index2]=temp;
        }
}
```

- synchronized takes the lock object as its parameter
- Fine-grained locking
  - Multiple thread can enter swap() method, but just wait lock for that block

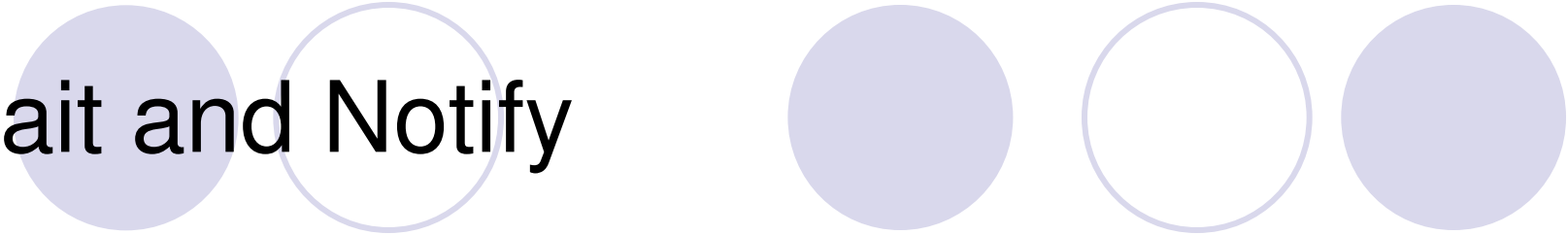
Easier way

- Make the whole method **synchronized**
- If a thread is inside one of the synchronized methods, all other threads are blocked from entering any of the synchronized methods of the class until the first thread returns from its call

```
synchronized void f() { /* ... */ }
synchronized void g(){ /* ... */ }
```

- if **f( )** is called for an object, **g( )** cannot be called for the same object until **f( )** is completed and releases the lock
  - there is a single lock that is shared by all the **synchronized** methods of a particular object

Wait and Notify

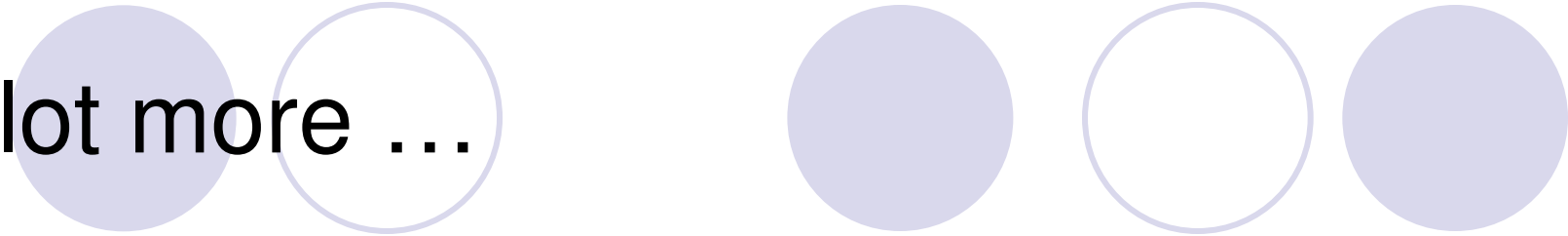


- Can call wait()/notify() on any object
  - Methods of Object class
- wait() will block the current thread until another thread calls notify() on the same object
- Must hold the lock on the object to call wait() or notify()
  - Which means they must be used within a synchronized method or block
- wait() means you give up your lock temporarily
- Can wait(2000) to time out after 2 secs

Example

- Queue.java
- The consumer has to wait until the producer push some object into the queue

A lot more …



- Operating Systems spends a lot of time on this topic
- JDK1.5 has lots of cool new multithreading features
- Producer-consumer queues, thread pools, etc.

Graphics User Interface (GUI)

- AWT – Abstract Window Toolkit
  - Peer-based approach
  - When you create a textbox on Java window, an underlying “peer” textbox created and handle the text input
- Swing
  - Underlying system just provides a blank window, and Java paints everything on top
  - Use AWT event model
  - Slower than AWT, but more robust

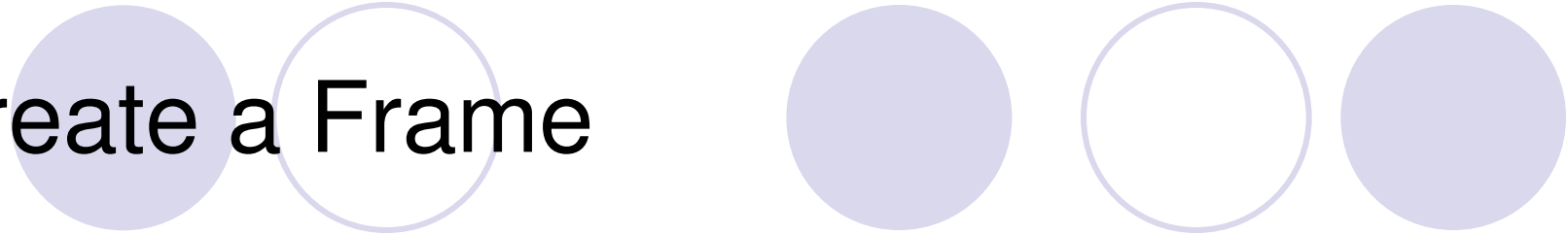
Graphics from cunix

- You need an X window server running on your local machine
- http://www.cs.columbia.edu/crf/crf-guide/resources/software/xwin32.html
  - Commercial, limited to Columbia CS machine
- http://www.cygwin.com/xfree/
  - Open source, no restrictions. Installation is clumsier

If you have problem

- Try using a tunneling ssh terminal client
- Teraterm+TTSSH
- http://www.cs.columbia.edu/crf/crf-guide/resources/software/ttssh

Create a Frame



- The top-level window, which is not contained inside another window
- JFrame in swing
  - Most Swing components start with “J”
- One of the few Swing component that is not painted on a canvas
  - Directly drawn by the user’s windows system

Draw an empty Frame

```
import javax.swing.*;

public class SimpleFrameTest
{
   public static void main(String[] args)
   {
      SimpleFrame frame = new SimpleFrame();
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      frame.show();
   }
}

class SimpleFrame extends JFrame
{
   public SimpleFrame()
   {
      setSize(300,200);
   }
}
```

Frame result

JFrame

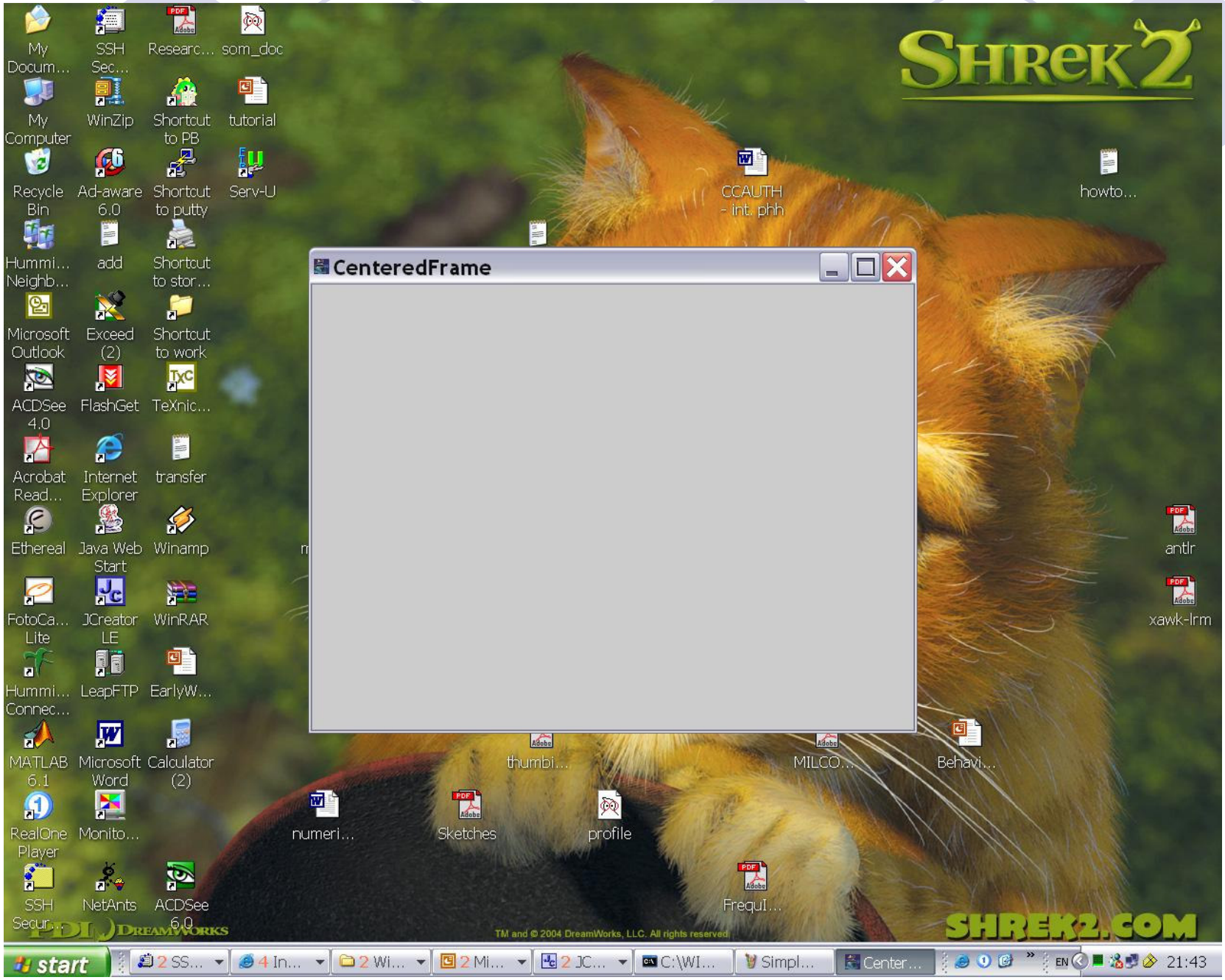
- setSize() gives the frame size.
- frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  - Define what should happen when user close this frame
  - By default, a frame is hidden when the user closes it, but program does not terminate
- frame.show()
  - Starts to show the frame.
  - Frames start their life invisible
  - show() is deprecated in JDK5.0, need to use frame.setVisible(true);
- By default, position on the upper-left corner

Positioning a Frame

```
class CenteredFrame extends JFrame
{
   public CenteredFrame()
   {
      // get screen dimensions
      Toolkit kit = Toolkit.getDefaultToolkit();
      Dimension screenSize = kit.getScreenSize();
      int screenHeight = screenSize.height;
      int screenWidth = screenSize.width;

      // center frame in screen
      setSize(screenWidth / 2, screenHeight / 2);
      setLocation(screenWidth / 4, screenHeight / 4);

      // set frame icon and title
      Image img = kit.getImage("icon.gif");      setIconImage(img);
      setTitle("CenteredFrame");
   }
}
```


CenteredFrame
My Docum...
SSH Sec...
Researc...
som_doc
My Computer
WinZip
Shortcut to PB
tutorial
Recycle Bin
Ad-aware 6.0
Shortcut to putty
Serv-U
Hummi... Neighb...
add
Shortcut to stor...
Microsoft Outlook
Exceed (2)
Shortcut to work
ACDSee 4.0
FlashGet
TeXnic...
Acrobat Read...
Internet Explorer
transfer
Ethereal
Java Web Start
Winamp
FotoCa... Lite
JCreator LE
WinRAR
Hummi... Connec...
LeapFTP
EarlyW...
MATLAB 6.1
Microsoft Word
Calculator (2)
RealOne Player
Monito...
SSH Secur...
NetAnts
ACDSee 6.0
CCAUTH - int. phh
howto...
thumbi...
MILCO...
Behavi...
antlr
xawk-lrm
numeri...
Sketches
profile
FrequI...
SHREK2
SHREK2.COM
TM and © 2004 DreamWorks, LLC. All rights reserved.
start
2 SS...
4 In...
2 Wi...
2 Mi...
2 JC...
C:\WI...
Simpl...
Center...
21:43

Display information inside frame

- Frames in Java are designed to be containers for other components like button, menu bar, etc.
  - You can directly draw onto a frame, but it's not a good programming practice
- Normally draw on another component, called panel, using JPanel

Add onto a frame

Before JDK5, get the content pane of frame first, then add component on it

```
Container contentPane = getContentPane();
Component c = …;
contentPane.add(c);
```

After JDK5, you can directly use frame.add(c);

```
class NotHelloWorldFrame extends JFrame
{
   public NotHelloWorldFrame()
   {
      setTitle("NotHelloWorld");      setSize(300, 200);
      // add panel to frame
      NotHelloWorldPanel panel = new NotHelloWorldPanel();
      Container contentPane = getContentPane();
      contentPane.add(panel);
   }
}

class NotHelloWorldPanel extends JPanel
{
   public void paintComponent(Graphics g)
   {
      super.paintComponent(g);
      g.drawString("Not a Hello, World program", 75, 100);
   }
}
```


NotHelloWorld
Not a Hello, World program

paintComponent()

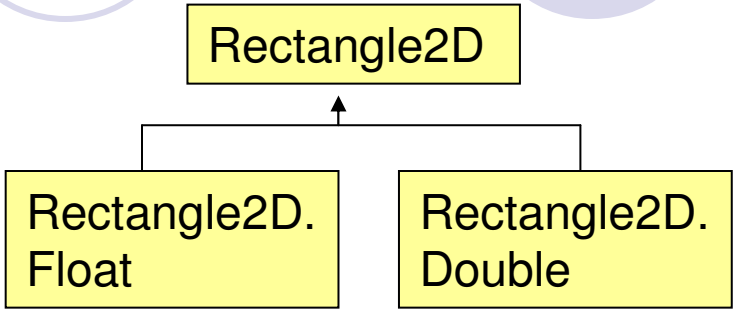
- paintComponent() is a method of JComponent class, which is superclass for all nonwindow Swing components
- Never call paintComponent() yourself. It's called automatically whenever a part of your application needs to be drawn
  - User increase the size of the window
  - User drag and move the window
  - Minimize then restore
- It takes a Graphics object, which collects the information about the display setting

Java 2D

- Powerful set of 2D graphics
- Need to obtain Graphics2D class, which is a subclass of Graphics class
- Line2D, Rectangle2D, Ellipse2D classes
- If you are using a JDK with Java 2D enabled, methods like paintComponent() automatically get object of Graphics2D, just need to cast it

```
public void paintComponent(Graphics g){
      Graphics2D g2 = (Graphics2D) g;
}
```

Float vs. Double



- static inner class, but just use them as normal classes
- When use .Float object, supply the coordinates as float number
- When use .Double object, supply the coordinates as double number
- Just for use of easy, no need to convert between float and double numbers

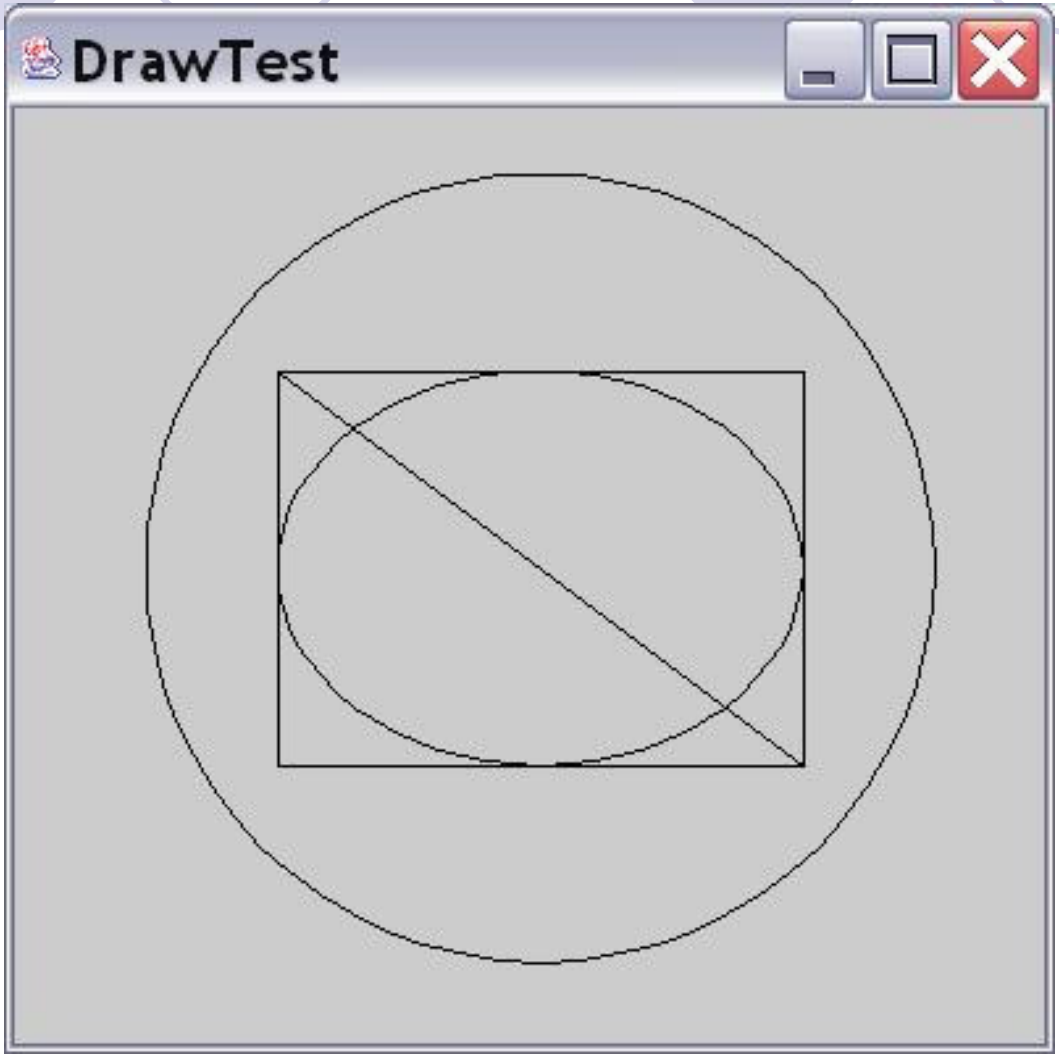
```
class DrawPanel extends JPanel{
   public void paintComponent(Graphics g){
      super.paintComponent(g);
      Graphics2D g2 = (Graphics2D)g;

      // draw a rectangle
      double leftX = 100;      double topY = 100;      double width = 200;      double height = 150;
      Rectangle2D rect = new Rectangle2D.Double(leftX, topY, width, height);
      g2.draw(rect);

      // draw the enclosed ellipse
      Ellipse2D ellipse = new Ellipse2D.Double();
      ellipse.setFrame(rect);
      g2.draw(ellipse);

      // draw a diagonal line
      g2.draw(new Line2D.Double(leftX, topY, leftX + width, topY + height));

      // draw a circle with the same center
      double centerX = rect.getCenterX();      double centerY = rect.getCenterY();
      double radius = 150;
      Ellipse2D circle = new Ellipse2D.Double();
      circle.setFrameFromCenter(centerX, centerY, centerX + radius, centerY + radius);
      g2.draw(circle);
   }
}
```

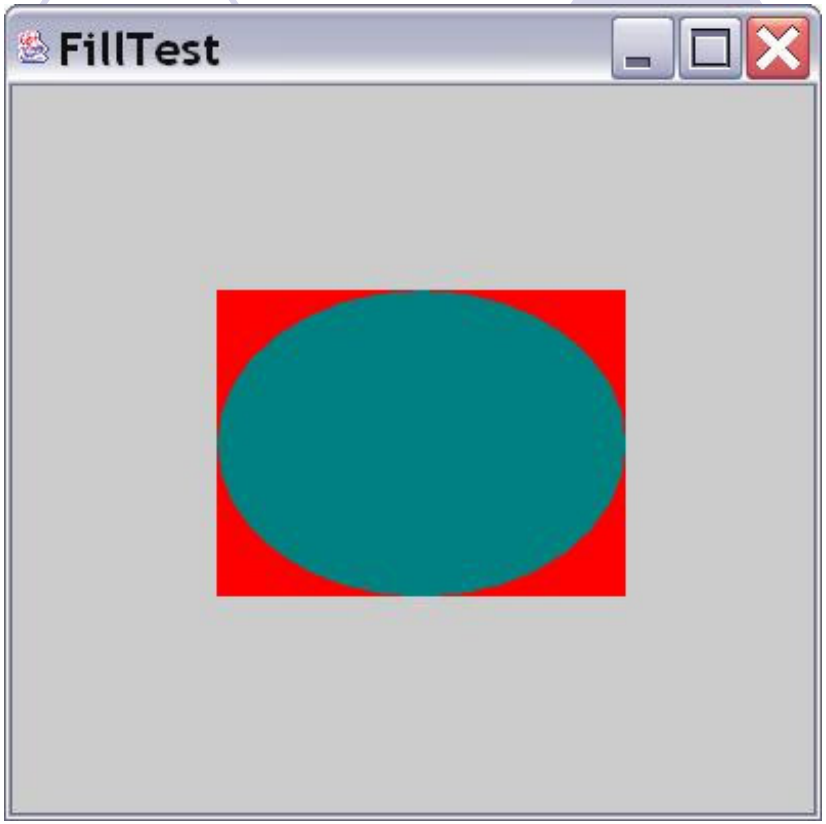

DrawTest

Fill in Color

```
class FillPanel extends JPanel{
   public void paintComponent(Graphics g){
      super.paintComponent(g);
      Graphics2D g2 = (Graphics2D)g;

      // draw a rectangle
      double leftX = 100;      double topY = 100;
      double width = 200;      double height = 150;
      Rectangle2D rect = new Rectangle2D.Double(leftX, topY, width, height);
      g2.setPaint(Color.RED);
      g2.fill(rect);

      // draw the enclosed ellipse
      Ellipse2D ellipse = new Ellipse2D.Double();
      ellipse.setFrame(rect);
      g2.setPaint(new Color(0,  128, 128)); // a dull blue-green
      g2.fill(ellipse);
   }
}
```



Can also use the following methods from Component class:

void setBackground (Color c)

void setForeground (Color c)

```
class FontPanel extends JPanel{
   public void paintComponent(Graphics g){
      super.paintComponent(g);
      Graphics2D g2 = (Graphics2D)g;

      String message = "Hello, World!";
      Font f = new Font("Serif", Font.BOLD, 36);
      g2.setFont(f);
      g2.drawString(message, 35, 100);
      g2.setPaint(Color.GRAY);
  }
}
```

Event handling

- Any OS supporting GUIs constantly monitors events such as keystrokes and mouse clicks, then report to program.
- ActionEvent - EventListener
- http://java.sun.com/docs/books/tutorial/uiswing/events/index.html
  - Study by samples

Event model

- Button generates ActionEvents when someone clicks on it
- Other objects say “let me know” by calling the button’s addActionListener() method, which means “please let me know when you are clicked”

Typical format

```
ActionListener listener = new myListener;
//ActionListener is an interface with method actionPerformed

JButton button = new JButton("OK");
button.addActionListener(listener);

class myListener implements ActionListener{
    public void actionPerformed(ActionEvent e){
            //reaction to button click goes here
    }
}
```

```
class ButtonPanel extends JPanel{
   public ButtonPanel(){
      JButton yellowButton = new JButton("Yellow");
      JButton blueButton = new JButton("Blue");
      add(yellowButton);      add(blueButton);

      // create button actions
      ColorAction yellowAction = new ColorAction(Color.YELLOW);
      ColorAction blueAction = new ColorAction(Color.BLUE);

      // associate actions with buttons
      yellowButton.addActionListener(yellowAction);
      blueButton.addActionListener(blueAction);
}

private class ColorAction implements ActionListener   {
      public ColorAction(Color c){
         backgroundColor = c;      }
      public void actionPerformed(ActionEvent event) {
         setBackground(backgroundColor);      }
      private Color backgroundColor;
   }
}
```


ButtonTest
Yellow
Blue


ButtonTest
Yellow
Blue


ButtonTest
Yellow
Blue

Anonymous inner class

```
Public ButtonPanel(){
        makeButton("yellow", Color.YELLOW);
        makeButton("bule", Color.BLUE);
}

Void makeButton(String name, Color c){
        JButton b = new JButton(name);
        add(b);
        button.addActionListener(new ActionListener(){
          public void actionPerformed(ActionEvent e){
                setBackground(c);
          }
        });
}
```

Semantic and low-level events

- Semantic event is one that express what the user is doing
  - Button click
  - Adjust scrollbar
- Low-level events are those events that make this possible
  - Mouse down, mouse up or keystroke
  - Dragging a mouse

Semantic and low-level events

- Semantic event
  - ActionEvent (button click, menu selection, ENTER in text field)
  - AdjustmentEvent (adjust the scroll bar)
  - ItemEvent (select from a set of checkbox or list items)
  - TextEvent (textfield or text area content changed)
- Low-level event
  - ComponentEvent (component resize, move, hidden)
  - KeyEvent
  - MouseEvent
  - FocusEvent (a component got focus, lost focus)
  - WindowEvent (window activated, iconified, closed)
  - ContainerEvent (component added or removed)