

Lecture 28 Applets I

Sample Applet code — /share/copy/aps105/Applets

Simple Applet Graphics

- in Java, graphics programs are *objects*!
- an *Applet* is a type of Java graphics program you run in a web browser.

```
import java.applet.*;
// to find 'Applet' class
import java.awt.*;
// to find 'Graphics' class

public class Hello extends Applet
{
    public void paint( Graphics g )
    {
        g.drawString( "Hello, world!", 50, 100 );
        g.drawString( "origin", 10, 10 );
    }
}
```

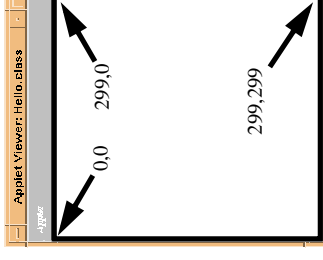
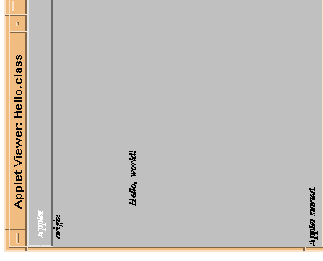
Running the Applet

- in “Hello.html”, write


```
<APPLET code="Hello.class" width=300 height=300>
</APPLET>
```
- gives a size (width, height) to the graphics area
- code names the program class we want to load
- use appletviewer to run it (don't use ‘java’)


```
appletviewer Hello.html &
```
- can also put this code in a web page!
- place Hello.class and Hello.html in your web space
- point web browser to the Hello.html file

Output



Explanation

- import lines tell Java where to look for other classes
- Applet is another class defined in the system
 - defines a type of object with instance variables, methods
 - Graphics is also another class, own methods & variables
- public class Hello extends Applet
- *inherits* (borrows) all methods, vars in the Applet class
- almost as if you had typed them in your Hello class
- you provide new methods to extend the Applet class
- class Hello must be public
- or appletviewer cannot find the class Hello

In appletviewer, try clicking on menu options:

- **Restart** to re-run your program
- **Reload** to reload the Program.class file
- **Clone** to make a clone copy of the running program (applet is an object, it can be cloned!)

Controlling your Applet

- appletviewer is a special version of the java program
- it runs your program, contains the JVM
- it doesn't look for a main() method
- appletviewer **controls** your program
- your program only does what appletviewer asks
 - it may ask you to paint() the graphics area
- your applet is like an ADT
 - it must provide an interface, so appletviewer can use it
- your program must have specific public methods
 - specific, predefined name, parameter list, intended goal
- if your Applet window is ever erased
 - eg: another window moved over top of it, then moved away
- appletviewer calls your the paint() method
- the paint() method is called **every time** it has to redraw the window contents!

Graphics Object

- the parameter to paint() is a Graphics object
- this is the “graphics area” where things get drawn
- lots of things you can do:


```
g.drawString( "Hello, world!", x, y );
g.drawLine( x_from, y_from, x_to, y_to );
g.drawRect( x, y, width, height );
g.drawOval( x, y, width, height );
g.fillRect( x, y, width, height );
g.fillOval( x, y, width, height );
```
- calls to drawOval(), drawRect(), fillOval(), fillRect()
 - x, y is **top left** corner of shape
- calls to drawString()
 - x, y is **bottom left** corner of text

Color and Dimension Objects

Draw some coloured boxes, scaled to the size of the window.

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

public class ColourBox extends Applet {
    private Color myPurple;

    public void init() {
        myPurple = new Color(219, 68, 219);
    }

    public void paint( Graphics g ) {
        Dimension d = getSize();
        // get window size

        g.setColor( Color.blue );
        g.fillRect( 0, 0, d.width, d.height );

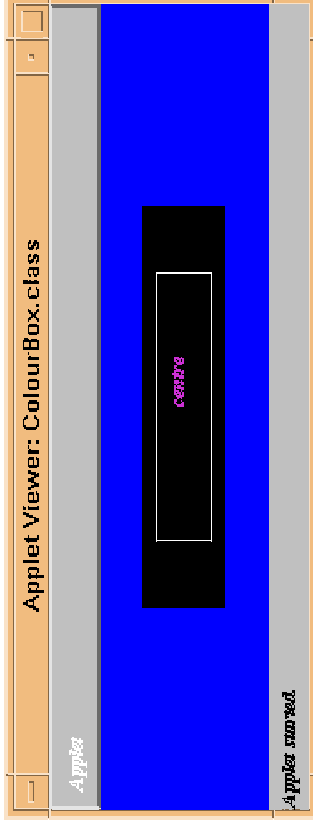
        g.setColor( Color.black );
        g.fillRect( d.width/4, d.height/4,
                  d.width/2, d.height/2 );

        g.setColor( Color.white );
        g.drawRect( d.width/3, d.height/3,
                  d.width/3, d.height/3 );

        g.setColor( myPurple );
        g.drawString( "centre", d.width/2,
                    d.height/2 );
    }
}
```

- can resize this window, and rectangles automatically resize

Output



Color.blue? Color class.... predefined colours:

```
Color.magenta, .yellow, .cyan
    .gray, .lightGray, .darkGray,
    .green, .orange, .pink, .red
```

- create your own colors

```
Color myColor = new Color(red,green,blue);
```

- r,g,b values 0..255
- NOTE: American spelling of **Color** is used

Dimension objects...

- only contains two integers: `.width, .height`
- used to get size of the window
- can scale all drawings to the size of the window!

- `init()` method

- called by appletviewer when the applet is first loaded
- we don't have a constructor... use `init()` method
- only called once
- here, we use it to initialize our *instance variables*

Input

- you cannot use `StdIn.getInt()` methods anymore
- will learn how to get input next lecture or two

Output

- you can still print debugging information to screen


```
System.out.println("debugging info");
```
- debugging info can be ugly on screen
- debugging output through `System.out.println()` won't be seen by a web browser (will it cause an error?)
- applet has 1 "status line" at bottom of window


```
showStatus( "display this status text" );
```
- good to use for
 - debugging in web browser
 - short messages to user
 - we will use this in the next lecture

More Graphics, Applet methods

- lots of methods, things you can do, eg:
 - `repaint()` method in applet
 - gives appletviewer a request
 - "some time in the near future, please erase my window and call my `paint()` method"
 - `repaint()` returns before the painting is actually done!
- to find them, use the online reference manuals
- <http://java.sun.com/products/jdk/1.2/docs/api/index.html> or:
 - course web page <http://www.ecf.utoronto.ca/~aps105w>
 - click on "Java Use in APS105" link, then click on "Java 2 SDK API" link

Lecture 29 Applets II: Events

Event-Driven Programming

- before
 - program was in control
 - `Stdin.getInt()` gets something from user:
 - program paused, waiting for user to type something
 - program resumed after pressing ENTER
- now
 - appletviewer is in control
 - your program does not wait for input
 - appletviewer does the 'waiting'
 - when mouse moved, key pressed
 - appletviewer calls a method in your program
 - this method call is called an *event*
 - program may *ignore* event, or it may *respond*
 - eg, draw something in window

Event-driven programming

- your program is not in control
- something else generates events
- appletviewer sends them to your program
- for each event, a method is called in your program
- do work required by the event quickly
 - there may be other events waiting
 - this method must finish before they can be processed
 - appletviewer is waiting, cannot do anything else!!
- when done, return from method
- control goes back to appletviewer
- another event may come again
 - respond to that event

1. Keyboard Events

- you can tell when a key is pressed or released
 - must write some special methods
 - `keyPressed()`
 - `keyTyped()`
 - `keyReleased()`
- you must do 3 things to get these keyboard events:
 1. include all 3 of these methods in your program
 2. add the following to class declaration
 - `implements KeyListener`
 3. call `addKeyListener(this);` in `init()` method
- appletviewer calls these methods for you...
 - pressing and releasing the 'a' key
 - calls `keyPressed()` method in your program
 - calls `keyTyped()` method
 - calls `keyReleased()` method
 - holding 'shift' key down
 - calls `keyPressed()`
 - can hold down for a long time
 - releasing calls `keyReleased()`
 - your `keyTyped()` method will not be called for 'shift'
 - holding the 'a' key down, repeatedly cycles
 - calls `keyPressed()`
 - calls `keyTyped()`
 - calls `keyReleased()`
- probably only use `keyPressed()` method
 - must supply other 2, don't put any code in them

NOTE: appletviewer requires you to press the **TAB** key before it will call any keyboard events.

Keyboard Events Program

```

import java.applet.*;
import java.awt.*;
import java.awt.event.*;
// to find 'KeyListener', 'KeyEvent' class
public class Keys extends Applet
    implements KeyListener
{
    public void init() {
        // tell appletviewer we want to know when key
        // events occur. it will call 'KeyListener'
        // methods in this object.
        addKeyListener(this);
    }

    public void paint( Graphics g ) {
        g.drawString( "Hello, world!", 50, 100 );
        g.drawString( "origin", 10, 10 );
    }
    // - - - - -
    // KeyListener methods
    public void keyPressed( KeyEvent e ) {
        showStatus( "key pressed" );
        System.out.println( e );
    }
    public void keyReleased( KeyEvent e ) {
        showStatus( "key released" );
        System.out.println( e );
    }
    public void keyTyped( KeyEvent e ) {
        showStatus( "key typed" );
        System.out.println( e );
    }
}

```

KeyEvent Class

- each of the keyXXX() methods contains a parameter
keyPressed(KeyEvent e)
 - parameter tells you about the event
 - which key was pressed?
 - what character was it?
 - which key was pressed?
 - e.getKeyChar()
 - returns 'E' if E is pressed
 - e.getKeyCode()
 - special integer value for each key
 - because 'shift', 'control' do not have char values!!!
- ```

int keycode = e.getKeyCode();
if(keycode == KeyEvent.VK_LEFT)
 // left arrow
if(keycode == KeyEvent.VK_SHIFT)
 // shift key
if(keycode == KeyEvent.VK_A)
 // the 'a' key (whether shifted or not)

```
- see KeyEvent class in API
  - list of all many key codes, methods you can use

## 2. Mouse Events

- when mouse button is pressed, released, clicked

```
mousePressed()
mouseReleased()
mouseClicked()
```

- clicked happens after a press+release

- when mouse enters, leaves the window

```
mouseEntered()
mouseExited()
```

- to observe these new mouse events

- add all 5 methods to your program
- add the following to class declaration
  - implements `MouseListener`
- call `addMouseListener(this)`; in `init()` method

- often, want to draw in window due to mouse action
- eg: remember where the mouse clicked
- call `repaint()`;

- the parameter (`MouseEvent e`) tells you
  - where the mouse was clicked / entered / exited
  - which mouse button was pressed
  - `int x = e.getX()`;
  - `int y = e.getY()`;
  - X/Y position of mouse pointer in your window
  - `Point p = e.getPoint()`;
  - returns `Point` object (page 785 of textbook)
  - `p.x, p.y` integers specifying x,y position
- `int mods = e.getModifiers()`;
- integer tells which mouse buttons were held down

## Example

- simple 'scribble' program
- use an array to store up to 10 mouse clicks
- draw a line between mouse click positions

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
// to find 'MouseListener', 'MouseEvent' class
```

```
public class Scribble extends Applet
implements MouseListener
```

```
{
```

```
// Point is a very simple class that has only
// .x and .y instance fields.
private Point[] mouseClicks;
private int numClicks;
```

```
public void init() {
 addMouseListener(this);
 mouseClicks = new Point[10];
 numClicks = 0;
}
```

```
public void paint(Graphics g) {
 for(int i=1; i < numClicks; i++) {
 g.drawLine(
 mouseClicks[i-1].x, mouseClicks[i-1].y,
 mouseClicks[i].x, mouseClicks[i].y);
 }
}
```

```
// -----
// MouseListener methods
```

```

public void mouseClicked(MouseEvent e) {
 if(numClicks < mouseClicks.length) {
 mouseClicks[numClicks] = e.getPoint();
 numClicks++;
 }
 repaint();
 // ask appletviewer to clear screen and call
 // paint(), but not done right away...
 // done later when appletviewer has time
}

// need the following, even though not used
public void mousePressed(MouseEvent e) { }
public void mouseReleased(MouseEvent e) { }
public void mouseEntered(MouseEvent e) { }
public void mouseExited(MouseEvent e) { }
}

```

### 3. Mouse Motion Events

- if mouse is in your window
- can detect when mouse is moved or dragged
- dragged is when moved *while holding button down*

```

mouseMoved(MouseEvent e)
mouseDragged(MouseEvent e)

```
- to implement these new mouse events
  - again, must add **both** methods to your program
  - add the following to class declaration

```

implements MouseMotionListener

```
  - add the following line to `init()` method

```

addMouseMotionListener(this);

```

### Combining Multiple Listeners

- can combine these mouse events with
  - mouse motion events
  - key events
  - any other type of events
- to do this...
  - add extra “listeners” to implements line
  - add all methods required
- see ‘Extra Lecture’ notes for
  - action events from text fields and buttons
  - pausing
  - timing
  - animation
  - threads (very advanced)



## Extra Lecture Applets III

Last Day — keyboard and mouse events

Note — this is not a part of the course, advanced material!

### Getting Text Input

- other ways of getting input from the user
- create input objects:
- TextField — one line to enter text input
- TextArea — multiple lines to enter text input
- Button — a push-button
- when user presses ENTER or pushes button
- a method is called, `actionPerformed()`
- the input object must know *who to notify*
- after creating the input object, tell it who is listening  
`inputObject.addActionListener( listener );`
- should have separate listening object for each source of input
- that object has `actionPerformed()` method
  - called when pushbutton is pressed for that Button, or
  - called when ENTER pressed in that TextField
- listening object can respond to input
- it can also tell the applet what happened
- to display the Label or Button or TextXXXX
- must add( ) it to the applet window
- to display text beside a TextField (say, as a prompt)
- Label — short text message
- must add( ) it to the applet window

### Single Text Field for Input

- here, the applet itself is the listening object
- more than one TextField/Button/TextArea?
- create separate listening objects (not applets!)
- see textbook for examples (Fahrenheit, Quotes)

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

public class TextEntry extends Applet
 implements ActionListener
{
 Label promptA; // remember the prompt
 TextField inputA;
 String stringA;

 // set up the graphical user interface components
 // add a label (prompt), then add an input box.
 public void init ()
 {
 promptA = new Label("Enter text:");
 add(promptA); // add to window so it is displayed

 inputA = new TextField ("type here", 10);
 inputA.addActionListener(this);
 // tell inputA to alert this applet when ENTER
 // is pressed
 add(inputA); // add to window so it is displayed
 }

 // whenever the user presses RETURN, this
 // method is called.
 public void actionPerformed(ActionEvent e)
 {
 String stringA = inputA.getText();
 // show the accepted text on the status line
 showStatus("Text entered: " + stringA);
 repaint ();
 }
}
```

## Pausing

- animation... controlling speed
- to pause, don't do this:
 

```
for(int i=0; i < 10000000; i++) {
 // do nothing
}
```
- proper way:
 

```
try {
 Thread.sleep(10); // 10 milliseconds
}
catch(InterruptedException e) {
 System.out.println("ignoring exception");
}
```
- this causes your program to “sleep” for 10 ms
- the try { } catch() { } block
- try doing first part, on error do catch() { } part
- try doing area within try { ... } block
- if an ‘exception’ (a special error condition) happens
  - a special InterruptedException object is ‘sent’
  - stop doing work in the try block
  - immediately jump to the catch() { ... } block
  - inside the catch(){...}
  - ‘e’ is the exception object sent
  - use it to determine type of error
  - we may attempt to ‘fix the problem’
  - we’ll ignore it here
  - see StdIn.java for another try/catch example

## “Stopwatch” Timing

```
long start = System.currentTimeMillis();
doSomeMethod();
long stop = System.currentTimeMillis();
double elapsedSeconds = (stop-start)/1000.0;
```

## Animation

- animating applets is difficult
- your program does one thing at a time:
  - calls mouseClicked(), returns.
  - calls paint(), returns.
  - mouseClicked() should always return quickly
  - if you're doing a “screen saver”
    - easiest type of animation
    - no user input to worry about
    - start animation in the mouseClicked() method
    - never have to **return** if you don't want too
    - not clean or proper, but “oh well”
  - next hardest animation
    - short animation clip reacting to user input
    - eg: firing a missile across the screen
    - when missile goes off screen, animation stops
    - can start animation in mouseClicked() method
    - do a return (soon!) when animation stops
    - can now get a new mouseClicked() call
  - hardest animation
    - eg: games like pong, arkanoid, pacman, space invaders
    - other objects moving “in the background”
    - eg: a ball, pacman ghosts, invading UFOs
    - if you don't click the mouse
      - your program isn't “running”
      - only runs when mouseClicked() called, or paint() called
      - called only in response to USER ACTIONS
    - how to work in background?
    - need to use “Thread” class
    - beyond scope of course to do this well....

## Threads

```

public class MyThread extends Thread {
 Applet theApplet;

 public MyThread(Applet a) {
 theApplet = a; // remember the applet
 }

 public void run() {
 // this method is called when the thread
 // starts running. do something!
 animate();
 }

 private void myPause(int ms) {
 try { Thread.sleep(ms); }
 catch(InterruptedException e) { }
 }

 private void animate() {
 // ... do something long here...
 while(true) {
 myPause(100); // wait 100ms
 a.repaint(); // ask applet to repaint!
 }
 }
}

```

...inside your applet:

```

MyThread myThread = new MyThread(this);
myThread.start(); // to start it, calls run()
myThread.stop(); // to stop it

```

The hard part about threads is communicating values between them. What if the Applet and the Thread want to both update the same variable (say, increment a counter) at the same time?