

topics:

• API's

- Java GUI API: applets, interface components, events
- Java Graphics API: applications, drawing basics

cis20.1-fall2007-sklar-lecll.1

applets (1).

- Java programs can run as *applications* or *applets*
- application:
  - executed using the *java* command
  - server and client can be the same machine or different machines
  - $\mbox{ client invokes JVM}$  which interprets classes and runs them

• applet:

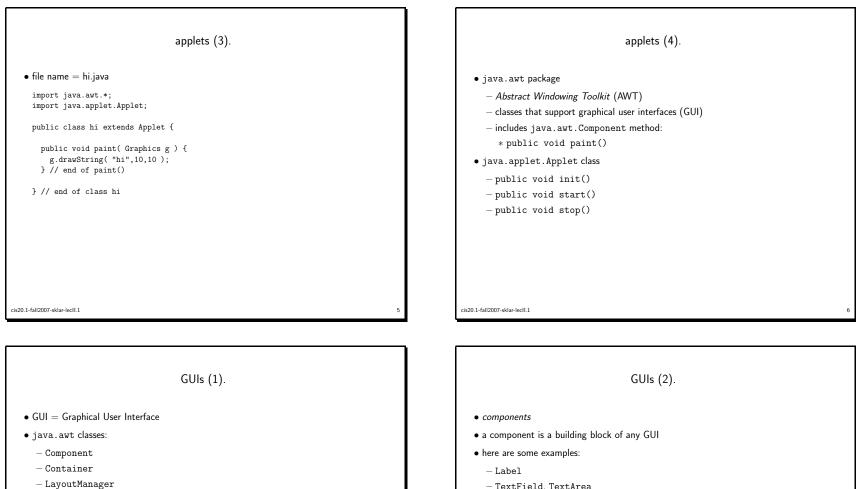
- must be executed using a browser, like Netscape, or the appletviewer command
- server sends applet to the client, in the form of class files; applet invokes JVM which interprets classes and runs them on the client

- there are two parts:

- \* an HTML file used to invoke the applet
- \* Java class file(s) that contain the applet code

APIs
API = application programmer interface
provides an interface for programmers between a standard language and a specialized hardware device and/or operating system and/or components of a language.
we will use Java as an example
and examine the GUI and Graphics components of the Java API

# 

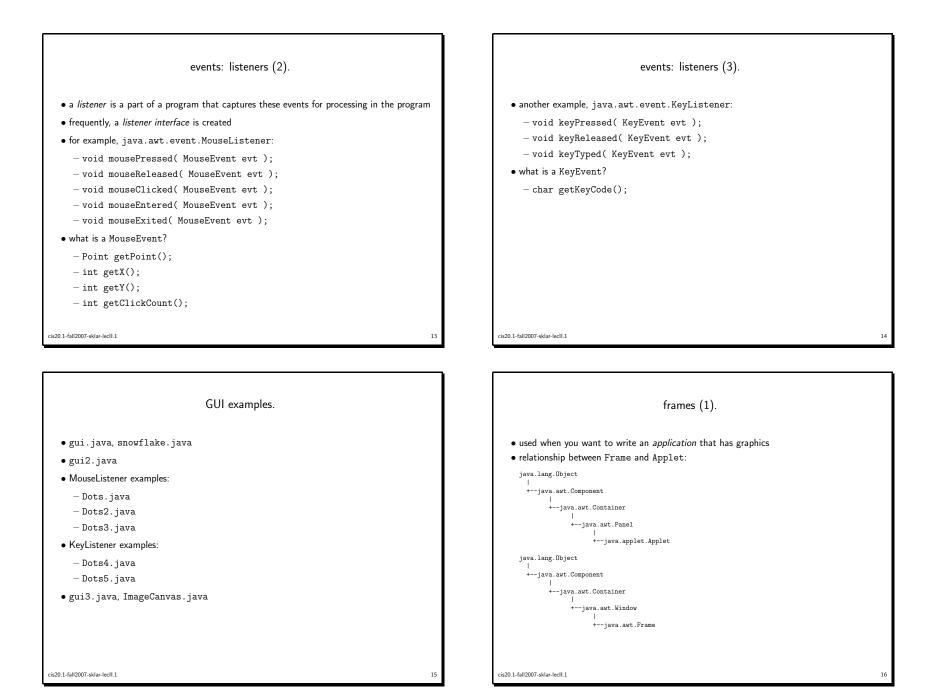


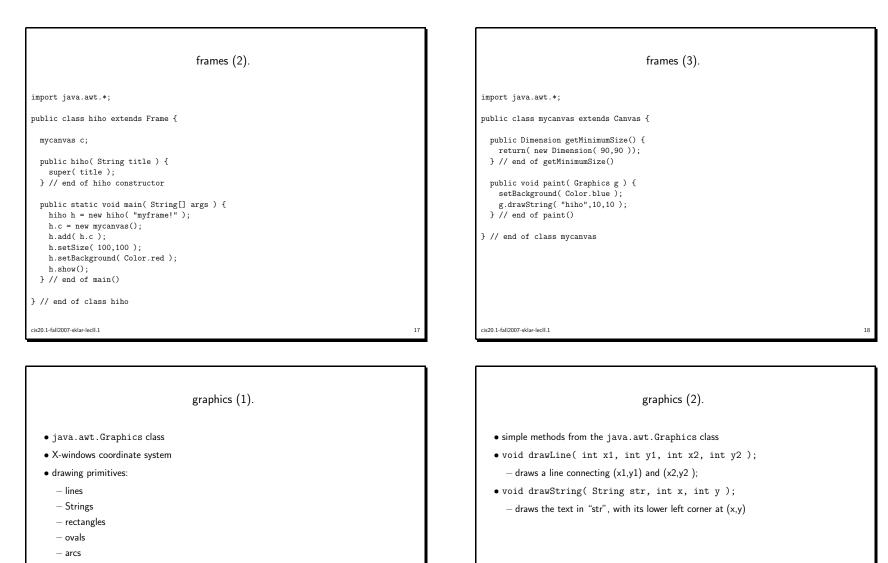
- -Event
- java.awt.event classes:
  - ActionListener
  - ItemListener
  - KeyListener
  - -MouseListener
  - -MouseMotionListener

cis20.1-fall2007-sklar-lecll.1

GUIs (2).
G

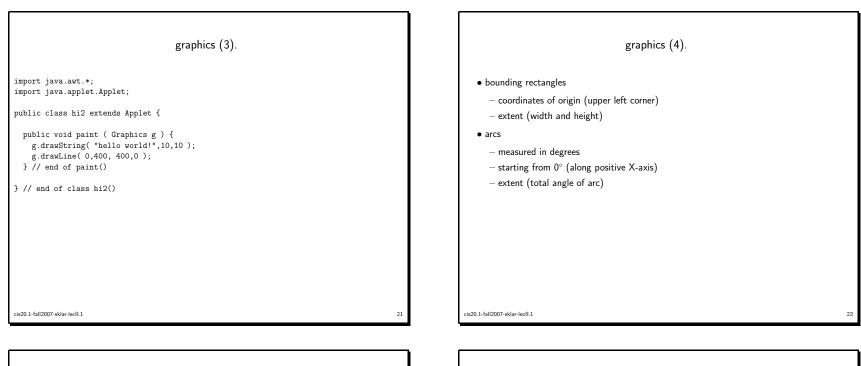
GUIs (3).	GUIs (4).
<ul> <li>containers</li> <li>a container is a special component that can hold other components</li> <li>here are some examples: <ul> <li>Applet</li> <li>Frame</li> <li>Panel</li> </ul> </li> </ul>	<ul> <li>layout managers</li> <li>a layout manager describes where the components are laid out within a given container</li> <li>you need to "set" the layout manager for each container</li> <li>you can "nest" containers (and their layour managers)</li> <li>BorderLayout — simplest layout manager</li> <li>looks like this: <ul> <li>north</li> <li>west center east</li> <li>south</li> </ul> </li> <li>GridBagLayout — more complex layout manager; but gives you the most control</li> </ul>
cis20.1-fall2007-sklar-lecll.1 9	cis20.1-fall2007-sklar-lecll.1 10
GUIs (5).	events (1).
<ul> <li>listeners</li> <li>are interfaces</li> <li>you need to implement the appropriate listener(s), depending on what events you want to handle</li> <li>then you need to override each method in the interface</li> <li>e.g., for a KeyListener, you need: <ul> <li>keyPressed()</li> <li>keyTyped()</li> <li>keyReleased()</li> </ul> </li> <li>the body of a method can be empty, if you don't want to do anything when a given event occurs</li> </ul>	<ul> <li>an event represents some action on the part of the user</li> <li>user-generated events are entered either through the mouse or the keyboard</li> <li>examples: <ul> <li>mouse pressed</li> <li>mouse released</li> <li>mouse clicked</li> <li>mouse entered</li> <li>mouse exited</li> <li>mouse moved</li> <li>mouse dragged</li> </ul> </li> </ul>
cis20.1-fall2007-sklar-lecll.1 11	cis20.1-fall2007-sklar-lecll.1 12





color

cis20.1-fall2007-sklar-lecll.1



23

# graphics (5).

- methods from the java.awt.Graphics class for drawing outlines of shapes
- void drawRect( int x, int y, int width, int height );
  - draws a rectangle with its upper left corner at (x,y), extending the specified "width" and "height"
- void drawOval( int x, int y, int width, int height );
  - $-\operatorname{draws}$  an oval circumscribed in the bounding rectangle with its upper left corner at (x,y), extending the specified "width" and "height"
- void drawArc( int x, int y, int width, int height, int startAngle, int arcAngle );
  - draws an arc whose oval is circumscribed in the bounding rectangle with its upper left corner at (x,y), extending the specified "width" and "height", where the arc starts at the "startAngle", measured in degrees (where  $0^{\circ}$ ) is horizontal along the positive x-axis), extending for "arcAngle" degrees

graphics (6).
<pre>import java.awt.*; import java.applet.Applet;</pre>
public class hi3 extends Applet {
<pre>public void paint ( Graphics g ) {    g.drawRect( 10,300,25,25 );    g.drawOval( 10,250,25,25 );    g.drawArc( 10,200,25,25,45,90 ); } // end of paint()</pre>
} // end of class hi3()
cis20.1-fall2007-sklar-lecll.1

### graphics (7).

- methods from the java.awt.Graphics class for drawing filled shapes
- void fillRect( int x, int y, int width, int height );
  - draws a filled rectangle with its upper left corner at (x,y), extending the specified "width" and "height"
- void fillOval( int x, int y, int width, int height );
  - draws a filled oval circumscribed in the bounding rectangle with its upper left corner at (x,y), extending the specified "width" and "height"
- void fillArc( int x, int y, int width, int height, int startAngle, int arcAngle );
  - draws a filled arc whose oval is circumscribed in the bounding rectangle with its upper left corner at (x,y), extending the specified "width" and "height", where the arc starts at the "startAngle", measured in degrees (where  $0^{\circ}$ ) is horizontal along the positive x-axis), extending for "arcAngle" degrees

cis20.1-fall2007-sklar-lecll.1

#### graphics (9).

- java.awt.Color class
- color is defined using the "RGB" methodology
- "Red", "Green", "Blue"
- each is an integer between 0 and 255, where 0 means no color and 255 means maximum color
- so white is: red=255 green=255 blue=255 or the ordered triple (255,255,255)
  - and black is: red=0 green=0 blue=0
  - and red is: red=255 green=0 blue=0
  - $\mbox{ and green is: red=0 green=255 blue=0}$
  - $\mbox{ and blue is: red=0 green=0 blue=255}$

• make up your own colors...

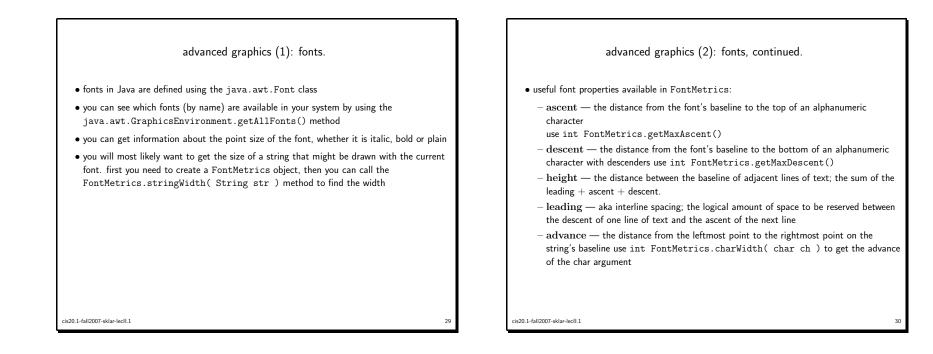
- graphics (8).methods from the java.awt.Graphics class for drawing polygons
- void drawPolygon(int[] xPoints, int[] yPoints, int nPoints);
   draws a closed polygon defined by arrays of x and y coordinates
- void drawPolygon( Polygon p );
  - draws the outline of a polygon defined by the specified Polygon object
- void drawPolyline( int[] xPoints, int[] yPoints, int nPoints );
   draws a sequence of connected lines defined by arrays of x and y coordinates
- the first two have counterparts for drawing filled polygons:
  - void fillPolygon(int[] xPoints, int[] yPoints, int nPoints); - void fillPolygon( Polygon p );

cis20.1-fall2007-sklar-lecll.1

## graphics (10).

- even more methods from the java.awt.Graphics class
  - void setColor( Color color );
  - \* sets the foreground (pen) color to the specified color
  - void fillRect( int x, int y, int width, int height );
    - $\ast$  draws a filled rectangle with its upper left corner at (x,y), extending the specified "width" and "height"
  - void fillOval( int x, int y, int width, int height );
  - $\ast$  draws a filled oval circumscribed in the bounding rectangle with its upper left corner at (x,y), extending the specified "width" and "height"
  - void fillArc( int x, int y, int width, int height, int startAngle, int arcAngle );
    - \* draws a filled arc whose oval is circumscribed in the bounding rectangle with its upper left corner at (x,y), extending the specified "width" and "height", where the arc starts at the "startAngle", measured in degrees (where  $0^{\circ}$ ) is horizontal along the positive x-axis), extending for "arcAngle" degrees

cis20.1-fall2007-sklar-lecll.1



# advanced graphics (3): images.

- you can load images from a URL and draw them
- load them using java.applet.getImage( URL url ) for an applet or java.awt.Toolkit.getImage( URL url ) for an application
- draw them using Graphics.drawImage() there are a number of versions of this method
- note that an Image is a Java object unto itself, defined in the java.awt package
- also note that URL is a Java object that must be instantiated prior to using either of the getImage() methods

#### advanced graphics (4): animation.

- computer animation is kind of like an old-fashioned flip book
- you need to draw the object(s) being animated repeatedly, in each new location
- each time, you calculate the new position of the object(s) and redraw
- you can either redraw the entire scene
- or you can only redraw the object(s) that are moving
- but in the second case, you need to "erase" the object first, then move it to its new location and redraw
- the erasing part can be tricky if the background is not solid

cis20.1-fall2007-sklar-lecll.1

### advanced graphics (5): GridBagLayout.

• GridBagLayout

cis20.1-fall2007-sklar-lecll.1

- you place components in the container in "rows" and "columns"
- you can specify the number of rows and columns
- you can specify the spacing between each row and/or column
- you can specify how a component is placed within its row/column, if it is smaller than the space allocated
- note that the height of an entire row is uniform, even if the components in each column are of different heights
- and the same for the width of a column
- all these are specified using a GridBagConstraints object

advanced graphics (6): GridBagLayout, continued.

GridBagConstraints( int gridx, int gridy, int int, gridwidth gridheight, double weightx, double weighty, int anchor, int fill, Insets insets, int ipadx, int ipady );

- gridx, gridy specify the location of the component, starting from (0,0)
- gridwidth, gridheight specify how many columns/rows the component occupies
- weightx, weighty specify how to distribute extra horizontal and vertical space
- anchor specifies where to place a component when it is smaller than its display area (e.g., CENTER, NORTH, NORTHEAST, ...)
- fill specifies whether to resize a component if it is smaller than its display area (e.g., NONE, HORIZONTAL, VERTICAL, BOTH)
- insets specifies minimum amount of space between a component and the edges of its display area (external padding)
- ipadx, ipady specifies how much space to add to the minimum width and height of the component (internal padding)