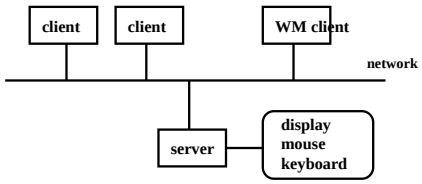
Graphical user interfaces

- **examples**
  - X Window system
  - Java Swing
  - Visual Basic, C#
  - Tcl/Tk (maybe)
  - Javascript
- **fundamental ideas**
  - interface components: widgets, controls, objects, …
  - methods, properties
  - events: loops and callbacks
  - geometry and layout management
  - use of hierarchy, inheritance
- **the GUI is the biggest chunk of code in many applications**
  - libraries try to make it easier
  - development environments and wizards and builders try to make it easier
  - it's still hard

X Windows (Bob Scheifler, Jim Gettys, mid-1980's)

- **client-server over a network**
  - works on single machine too, with IPC

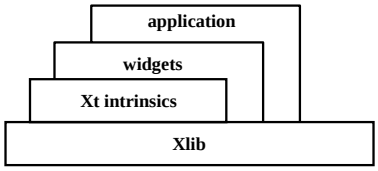


- **variants:**
  - X terminal (e.g., SunRay): server is standalone
  - workstation: server is on same processor as clients
  - remote clients, local server
  - Exceed: server on PC, clients on (usually) Unix
- **window manager is just another client, but with more properties**
  - clients have to let the window manager manage
  - InterClient Communications Conventions Manual

X Windows model (www.x.org)

- **server runs on the local machine**
  - accepts network (or local) client requests and acts on them
  - creates, maps and destroys windows
  - writes and draws in windows
  - manages keyboard, mouse and display
  - sends keyboard and mouse events back to proper clients
  - replies to information requests
  - reports errors
- **client application**
  - written with X libraries (i.e. Xlib, Xt)
  - uses the X protocol to
    send requests to the server
    receive replies, events, errors from server
- **protocol messages**
  - requests: clients make requests to the server
    e.g., Create Window, Draw, Iconify, ...
  - replies: server answers queries ("how big is this?")
  - events: server forwards events to client
    typically keyboard or mouse input
  - errors: server reports request errors to client

X Windows programming model



- **Xlib provides client-server communication**
- **intrinsics provide basic operations for building and combining widgets**
- **widgets implement user interface components**
  - buttons, labels, dialog boxes, menus, …
  - multiple widget sets, e.g., Motif
- **application uses all of these layers**

Xlib: bottom level library

- **basic mechanisms for**
  - requests from client to server: "draw on window", "how big is this?"
  - replies from server to client: "this big"
  - events from server to client: "button 1 pushed", "window exposed"
  - error reports: "out of memory"
- **basic Xlib-level client program**
  - connect client to server: XOpenDisplay()
  - get info about screen, compute desired size for window;
    hints, not mandatory; the WM is in charge
  - create window: XCreateSimpleWindow()
  - set standard properties for window manager
    sizes, window name, icon name, ...
  - select events to be received and discarded
  - create "graphics context" for storing info on color, depth, ...
    things that don't change from request to request
    server caches this info to cut down traffic
  - display window: XMapWindow()
    causes it to appear; up to this point it hasn't
  - loop on events

Events

- **client registers for events it cares about**
- **events occur asynchronously**
- **queued for each client**
- **client has to be ready to handle events any time**
  - mouse buttons or motion
  - keyboard input
  - window moved or reshaped or exposed
  - 30-40 others
- **information comes back to client in a giant union XEvent**

```
Xevent myevent;
for (;;) {
        XNextEvent(mydisplay, &myevent);
        switch (myevent.type) {
        case ButtonPress: …
        …
```

Hello world in X toolkit/ Motif widgets

```
#include <Xm/XmAll.h>

void main(int argc, char *argv[])
{
    Widget toplevel, main_w, button;
    XtAppContext app;
    XtSetLanguageProc(NULL, NULL, NULL);
    toplevel = XtVaAppInitialize(&app, "main", NULL, 0,
                                 &argc, argv, NULL, NULL);
    main_w = XtVaCreateManagedWidget("main_w",
        xmMainWindowWidgetClass,
        toplevel, XmNscrollingPolicy,
        XmAUTOMATIC, NULL);
    button = XtVaCreateWidget("Hello World",
        xmLabelWidgetClass, main_w, NULL);
    XtManageChild(button);
    XtRealizeWidget(toplevel);
    XtAppMainLoop(app);
}

/* cc x.c -lXm -lXt -lX11 */
```

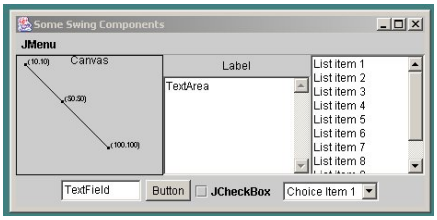
Hello world in GTk (plus ça change…)

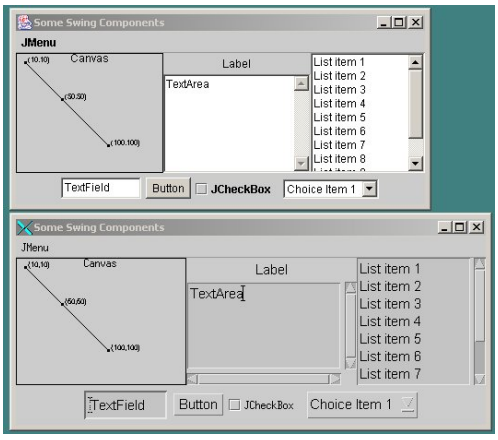
```
#include <gtk/gtk.h>

static void hello( GtkWidget *widget, gpointer data ) {
    g_print ("Hello World\n");
}

static gboolean delete_event( GtkWidget *widget, GdkEvent  *event,
                              gpointer  data ) {
    g_print ("delete event occurred\n");
    return TRUE;
}

static void destroy( GtkWidget *widget, gpointer data ) {
    gtk_main_quit ();
}

int main( int argc, char *argv[] ) {
    GtkWidget *window;
    GtkWidget *button;
    gtk_init (&argc, &argv);
    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
    g_signal_connect (G_OBJECT (window), "delete_event",
                      G_CALLBACK (delete_event), NULL);
    g_signal_connect (G_OBJECT (window), "destroy",
                      G_CALLBACK (destroy), NULL);
    gtk_container_set_border_width (GTK_CONTAINER (window), 10);
    button = gtk_button_new_with_label ("Hello World");
    g_signal_connect (G_OBJECT (button), "clicked",
                      G_CALLBACK (hello), NULL);
    g_signal_connect_swapped (G_OBJECT (button), "clicked",
                              G_CALLBACK (gtk_widget_destroy),
                              G_OBJECT (window));
    gtk_container_add (GTK_CONTAINER (window), button);
    gtk_widget_show (button);
    gtk_widget_show (window);
    gtk_main ();

    return 0;
}
```

Graphical user interfaces in Java

- **interfaces built from components**
  - buttons, labels, text areas, lists, menus, dialogs, …
  - canvas: graphics for drawing and image rendering
- **each component has**
  - properties: size, position, visibility, text, font, color, …
  - methods: things it will do, e.g., change properties
  - events: external stimuli it responds to
- **containers: hold components and containers**
- **layout managers: control size, placement of objects within a container**

(Campione & Walrath *Java Tutorial*)





Component object hierarchy

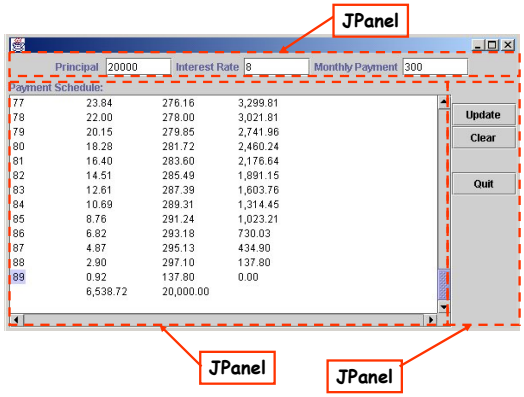
Object
-> Component
-> Container
-> Jcomponent
-> Jpanel
-> Jlabel
-> JButton
-> JTextComponent
-> JTextField
-> JPasswordField
-> JTextArea
-> …

- **containers hold components and containers**
  - used to build up nested structures
  - JFrame: top-level window
  - Jpanel: general container for components & containers
    put JPanels in a JFrame
  - JMenuBar for menubar across top of frame

- **individual components like Jbutton, Jtext…**
  - respond to events
  - have methods for other behaviors
  - have get and set methods for accessing properties
    like size, color, font

Layout hierarchy

- **JFrame holds one or more JPanels**
- **JPanel holds components and other Jpanels**
- **JPanel used for layout**
  - add() method adds components
  - uses a LayoutManager that lays out components
  - layout manager can be set to one of several



Events

- **stuff happens**
  - mouse motion, button push, button release, …
  - scrollbar fiddled
  - keyboard keypress, release, shift key, etc.
  - component got or lost focus
  - window iconified, uniconified, hidden, exposed, moved, reshaped, killed
  - etc.
- **each such event is passed to event-handling mechanism in the program**
- **program can decide what to do with it**

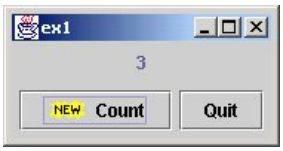
Events

- **components register to receive (listen for) events that they are interested in**
- **a thread watches for events like button push, mouse motion or click, key down or up, …**
- **each event is passed to listener(s) that registered for it**
- **obj.addActionListener(this)**
  - e.g., a Jpanel calls button.addActionListener(this)
  - tells **obj** to notify **this** container when event happens
    i.e., sets up a callback
  - usually called by container that contains object that will get the event
- **void actionPerformed(ActionEvent e) { … }**
  - called from component where event occurs (e.g., a button instance) when it does
    the callback happens
  - handler determines type or instance that caused event, does appropriate action
- **different kinds of listeners for different sources**
  - keyboard, mouse, mouse motion, window, …

Example 1: Buttons and labels

- after it starts:

- after Count button is pushed 3 times:



- after Quit button is pushed:

Example 1 events, layout

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

public class ex1 extends JFrame
                 implements ActionListener {
        int count;
        JLabel lab;
        JButton bcount, bquit;

public static void main(String[] args) {
        ex1 a = new ex1();
}

ex1() {
        setTitle("ex1");
        JPanel p1 = new JPanel();
        lab = new JLabel("Counter");
        p1.add(lab);
        bcount = new JButton("Count",
                             new ImageIcon("new.gif"));
        bcount.addActionListener(this);
        bquit = new JButton("Quit");
        bquit.addActionListener(this);
        JPanel p2 = new JPanel();
        p2.add(bcount);  p2.add(bquit);
        getContentPane().setLayout(new BorderLayout());
        getContentPane().add(p1, BorderLayout.NORTH);
        getContentPane().add(p2, BorderLayout.SOUTH);
        pack();
        show();
```

Example 1, continued

```
public void actionPerformed(ActionEvent ae) {
        System.out.println(ae.getActionCommand());
        if (ae.getActionCommand().equals("Count")) {
                                        // by content
                count++;
                lab.setText(Integer.toString(count));
        } else if (ae.getSource() == bquit) {
                                        // by object name
                System.exit(0);
        }
}
```

- **five steps to set up a GUI component:**
  - declare an object, like Button
  - create it with new
  - add it to a container
  - add an ActionListener to catch events
  - handle events in actionPerformed
- **information is spread all over the place**

Layout managers

- **control size, position, padding, stretching & shrinking, etc., for containers**
- **each container has a default layout manager**
  - change with setLayout method
    `jp.setLayout(new BorderLayout())`
  - or at creation
    `JPanel jp = new JPanel(new BorderLayout());`
- **FlowLayout**
  - fills area left to right in rows
  - each row can be centered, left or right adjusted
- **BorderLayout**
  - fills North, South, East, West, and Center
- **GridLayout**
  - regular array of rows and columns
  - specify number of each
- **CardLayout**
  - multiple windows that all occupy the same space
  - like tabs in Microsoft interfaces
- **GridBagLayout**
  - complicated grid layout
  - weighting factors on component areas, stretching, etc.

Flow Layout

- default for Panels


```
public class layout1 extends JFrame {

public static void main(String[] args) {
  layout1 a = new layout1();

  a.setTitle("layout1: flow");
  JPanel p = new JPanel();
  p.add(new Button("One"));
  p.add(new Button("Two "));
  p.add(new Button("Three"));
  p.add(new Button("Four"));
  p.add(new Button("Five"));
  a.getContentPane().add(p);
  a.pack();
  a.show();
}

}
```

Border Layout

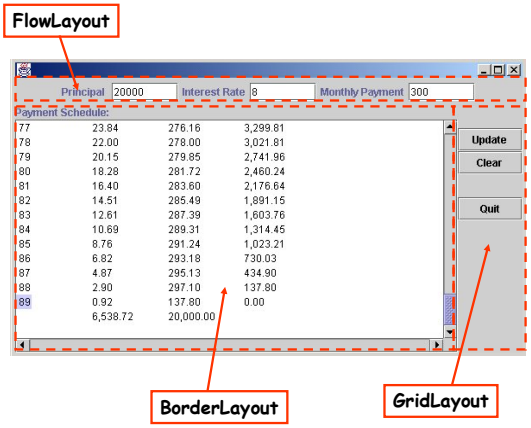

```
public class layout2 extends JFrame {

public static void main(String[] args) {
  layout2 a = new layout2();

  a.setTitle("layout2: border");
  JPanel p = new JPanel();
  p.setLayout(new BorderLayout());
  p.add(new JButton("north button"),
                        BorderLayout.NORTH);
  p.add(new JButton("south button "),
                        BorderLayout.SOUTH);
  p.add(new JButton("east"), BorderLayout.EAST);
  p.add(new JButton("westernmost button"),
                        BorderLayout.WEST);
  p.add(new JButton("center button"),
                        BorderLayout.CENTER);
  a.getContentPane().add(p);
  a.pack();
  a.show();
}
```

Grid Layout


```
public class layout3 extends JFrame {

public static void main(String[] args) {
  layout3 a = new layout3();

  a.setTitle("layout3: grid");
  JPanel p = new JPanel();
  p.setLayout(new GridLayout(3,2));
        p.add(new Button("One"));
        p.add(new Button("Two "));
        p.add(new Button("Three"));
        p.add(new Button("Four"));
        p.add(new Button("Five"));
  a.getContentPane().add(p);
  a.pack();
  a.show();
}
```

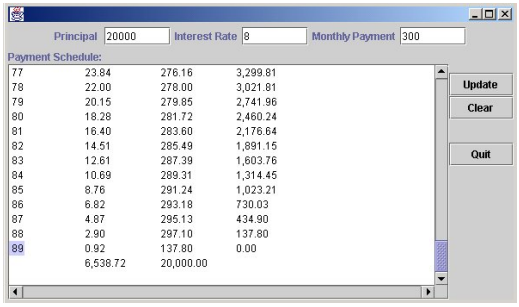
Layout hierarchy

- **JFrame holds one or more JPanels**
- **JPanel holds components and other Jpanels**
- **JPanel used for layout**
  - add() method adds components
  - uses a LayoutManager that lays out components
  - layout manager can be set to one of several



Example 2: Text components

- **JTextField**
  - single line for input
  - main interesting event is pushing Return
- **JTextArea**
  - multiple lines; can add scrolling
  - edit in place
  - change size and font for whole area but not parts
  - fancier JTextComponents for editing, display of different sizes and fonts, HTML, etc.



Example 2 code excerpts

```
class mtg extends JFrame implements ActionListener {
  JLabel lprin = new JLabel("Principal ");
  JTextField tprin = new JTextField(7);
  JLabel lrate = new JLabel("Interest Rate");
  JTextField trate = new JTextField(7);
  JLabel lmpay = new JLabel("Monthly Payment");
  JTextField tmpay = new JTextField(7);

  JLabel lsched = new JLabel("Payment Schedule:");
  JTextArea tpay = new JTextArea(15, 45);

  JButton update = new JButton("Update");
  JButton clear = new JButton("Clear");
  JButton quit = new JButton("Quit");

public static void main(String[] args) {
  mtg m = new mtg();
}
```

Example 2, page 2

```
mtg() {
  addWindowListener(new WindowAdapter() {
        public void windowClosing(WindowEvent e) {
                System.exit(0);
        }
  });

  // top row of entry boxes
  JPanel ptop = new JPanel();
  ptop.add(lprin);
  ptop.add(tprin);
  ptop.add(lrate);
  ptop.add(trate);
  ptop.add(lmpay);
  ptop.add(tmpay);
  tprin.setToolTipText("Enter principal amount");
  trate.setToolTipText("Enter yearly interest rate …");
  tmpay.setToolTipText("Enter monthly payment");

  // text area for payment schedule
  JScrollPane jsp = new JScrollPane(tpay,
     JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,
     JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS);
  JPanel pctr = new JPanel(new BorderLayout());
  pctr.add(lsched, BorderLayout.NORTH);
  pctr.add(jsp, BorderLayout.CENTER);
```

Anonymous inner classes

```
JLabel label = new JLabel("0");
JButton button = new JButton("Lookup");
button.addActionListener(
   new ActionListener() {
      public void actionPerformed(ActionEvent e) {
          n++;
          label.setText(n);
      }
   }
);
```

- an unnamed class defined inside another class

```
class foo implements ActionListener {
   public void actionPerformed(ActionEvent e) {...}
}
button.addActionListener(new foo());
```

- "Anonymous inner classes can be confusing at first, but once you're used to them, they make the code clearer by keeping the implementation of an event handler close to where the event handler is register." (Campione&Walrath, 359)

Example 2, page 3

```
 // buttons on right
  JPanel pr = new JPanel(new GridLayout(0,1));
  pr.add(new JLabel()); // spacer
  pr.add(update);
  pr.add(clear);
  pr.add(new JLabel()); // spacer
  pr.add(quit);
  JPanel pright = new JPanel(new BorderLayout());
  pright.add(pr, BorderLayout.NORTH); // pack at top

  update.addActionListener(this);
  clear.addActionListener(this);
  quit.addActionListener(this);
  update.setToolTipText("Update payment schedule");
  clear.setToolTipText("Clear payment schedule");

  // overall layout
  Container cp = getContentPane();
  cp.add(ptop, BorderLayout.NORTH);
  cp.add(pctr, BorderLayout.CENTER);
  cp.add(pright, BorderLayout.EAST);

  pack();
  show();
}
```

Example 2, page 4

```
public void actionPerformed(ActionEvent e) {
  if (e.getSource() == quit) {
        System.exit(0);
  } else if (e.getSource() == update) {
        tpay.setText(pay());
  } else if (e.getSource() == clear) {
        tpay.setText("");
  }
}
```

Example 2, page 5

```
String pay() {
  double mp = Double.parseDouble(tmpay.getText());
  double prin = Double.parseDouble(tprin.getText());
  double mrate = Double.parseDouble(trate.getText())
                                                / 12 / 100;
  double totint = 0;
  double totprin = 0;
  String s = "";
  for (int i = 1; i <= 500; i++) {
        double Int = prin * mrate;
        double dp = mp - Int;   // decrease of principal
        if (prin - dp > 0) {
                prin -= dp;
        } else {
                dp = prin;
                prin = 0;
        }
        s += " "+ i +"\t"+ f2(Int) +"\t"+
                        f2(dp) +"\t"+ f2(prin) +"\n";
        totint += Int;
        totprin += dp;
        if (prin <= 0)
                break;
  }
  s += "\t"+ f2(totint) +"\t"+ f2(totprin) +"\n";
  return s;
}
```

Example 2, page 6

```
String f2(double f) {  // %.2f, so help me
  NumberFormat form =
                NumberFormat.getInstance();
  form.setMinimumFractionDigits(2);
  form.setMaximumFractionDigits(2);
  //form.setMinimumIntegerDigits(6);
  return form.format(f);
}
```

Example 3: Notepad editor

- **menus**
- **file dialog**
- **I/O from file system**

Example 3, page 2

```
public class notepad extends JFrame
                    implements ActionListener {
  int last;
  JTextField tf = new JTextField(60);
  JTextArea ta = new JTextArea(15,60);
  JMenuBar mb = new JMenuBar();
  JMenu mfile;

public static void main(String[] args) {
  notepad a = new notepad();
}

notepad() {
  addWindowListener(new WindowAdapter() {
        public void windowClosing(WindowEvent e) {
              System.exit(0);
        }
  });

  setTitle("notepad");
```

Example 3, page 3

```
  setJMenuBar(mb);
  mfile = new JMenu("File");
  JMenuItem mi;
  mfile.add(mi = new JMenuItem("Open"));
        mi.addActionListener(this);
  mfile.add(mi = new JMenuItem("Save"));
        mi.addActionListener(this);
  mfile.add(mi = new JMenuItem("Quit"));
        mi.addActionListener(this);
  mb.add(mfile);

  tf.addActionListener(this);

  getContentPane().setLayout(new BorderLayout());
  getContentPane().add(tf, BorderLayout.NORTH);
  JScrollPane jsp = new JScrollPane(ta,
    JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,
    JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS);
  getContentPane().add(jsp, BorderLayout.CENTER);
  pack();
  show();
}
```

Example 3, page 4

```
public void actionPerformed(ActionEvent e) {
  if (e.getSource() == tf) {
        find(tf.getText());
  } else if (e.getSource() instanceof JMenuItem) {
        String b = e.getActionCommand();
        if (b.equals("Quit")) {
              System.exit(0);
        } else if (b.equals("Open")) {
              openfile();
        } else if (b.equals("Save")) {
              savefile(ta.getText());
        }
  }
}
public void find(String pat) {          // find next pat in ta
  String s = ta.getText(); // where to search
  if (last + pat.length() >= s.length())
        last = 0;
  int n = s.indexOf(pat, last); // look to end
  if (n == -1) {
        last = 0;
        n = s.indexOf(pat, 0);      // look from beginning
  }
  if (n >= 0) {
        ta.setSelectionColor(Color.red);
        ta.getCaret().setSelectionVisible(true);
        ta.select(n, n+pat.length());
        last = n + 1;
  }
}
```

JFileChooser open file

```
public void openfile() {
  JFileChooser jfc = new JFileChooser();
  jfc.showOpenDialog(this);
  if (jfc.getSelectedFile() == null)   // cancelled
        return;

  File fil = jfc.getSelectedFile().getAbsoluteFile();
  String f = fil.getAbsolutePath(); // attach
  directory name
  try {
        FileInputStream in =
                    new FileInputStream(f);
        byte [] data = new byte [in.available()];
        in.read(data);
        ta.setText(new String(data, 0));
  } catch (IOException e) {
        ta.setText("Can't open file " + f);
  }
}
```

JFileChooser save file

```
public void savefile(String s) {
  JFileChooser jfc = new JFileChooser();
  jfc.showSaveDialog(this);
  if (jfc.getSelectedFile() == null)   // cancelled
        return;

  File fil = jfc.getSelectedFile().getAbsoluteFile();
  String f = fil.getAbsolutePath(); // attach
  directory name
  try {
        FileOutputStream out =
                    new FileOutputStream(f);
        out.write(s.getBytes());
        out.close();
  } catch (FileNotFoundException e) {
        System.err.println(e + " can't open " + f);
  } catch (IOException e) {
        System.err.println(e + " savefile error");
  }
}
```

Applets

- **run Java code in browser**
- **HTML applet tag loads code into a web page**

```
<applet
   code="scribble.class" width=500 height=300>
</applet>
```

- **default security restrictions on applets**
  - can't access client file system
  - can't run processes on client
  - can't create unrestricted top-level windows
  - can't make the interpreter quit
  - etc., etc.
- **can use Socket() in an applet**
  - but can only open socket to system that applet came from
- **with Swing plug-in, can do somewhat more**
  - but still quite restricted

Applet scribble (from *Java in a Nutshell*)

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

public class scribble extends Applet {
        int last_x = 0, last_y = 0;

        boolean mouseDown(Event e, int x, int y) {
                last_x = x; last_y = y;
                return true;
        }
        boolean mouseDrag(Event e, int x, int y) {
                Graphics g = getGraphics();
                g.drawLine(last_x, last_y, x, y);
                last_x = x; last_y = y;
                return true;
        }
}
```

HTML to access an applet

```
<html>
<title> Scribble </title>
<body>
If you're running Java, you should see the
  scribble applet below.
<P>
<applet code="scribble.class"
              width=800 height=600>
</applet>
</body>
</html>
```

- parameters from applet tag are available to applet