Exceptions and Java Graphics

Robert Keller
October 2010

Exceptions

- An exception is a kind of "extraordinary" exit from otherwise normal control flow.
- Exceptions are used to "catch" **occasional** situations for which:
  - code would get cluttered if we had to code checks for the situation repeatedly, **or**
  - the situation may come from inside a method to which we do not have access.
- Such situations are said to "throw" the exception.

Why we need to know about this

- It helps makes your code robust (insensitive to various failures).
- Many library methods throw exceptions; you need to know how to code for these methods.
- Indicating errors by calling immediate exit or embedding print-statements is clumsy, and sometimes not optimally helpful.

Exceptions Detail

- Exceptions could include:
  - Divide by 0
  - Arithmetic overflow
  - Error input-output operation
  - Bad input format
  - and others, including programmer-defined ones
- Exceptions in Java are implemented as Exception **objects**.
- Exceptions can carry **values** indicating the **cause** of the exception.
- Exceptions should **not** be used as a normal value-returning mechanism.

Exception Lingo

- When an exception occurs it is said to be
  - "thrown"
- If an exception is thrown inside a method, it can either be:
  - "caught" (the buck stops here), or
  - "passed" (hot potato)

Exception passing

- An exception not otherwise caught will eventually get passed to the **top-level main**, at which point it will either be:
  - reported, then ignored, or
  - cause the program to terminate, if sufficiently severe.
- Continuing to operate after an exception has been caught at the top level can be risky.

Throwables

- In Java, there is a more general interface, of which Exception is a special case:
  - **Throwable** is the interface
    - **Exception** is an **implementation**, typically used as a base class.
    - **Error** is another **implementation**, usually indicating a more serious internal error.

Exception Examples

CloneNotSupportedException

DataFormatException

GeneralSecurityException

IllegalAccessException

InterruptedException

IOException

RuntimeException

RuntimeException Sub-classes

ArithmeticException

ClassCastException

EmptyStackException

IllegalArgumentException

IndexOutOfBoundsException

NegativeArraySizeException

NoSuchElementException

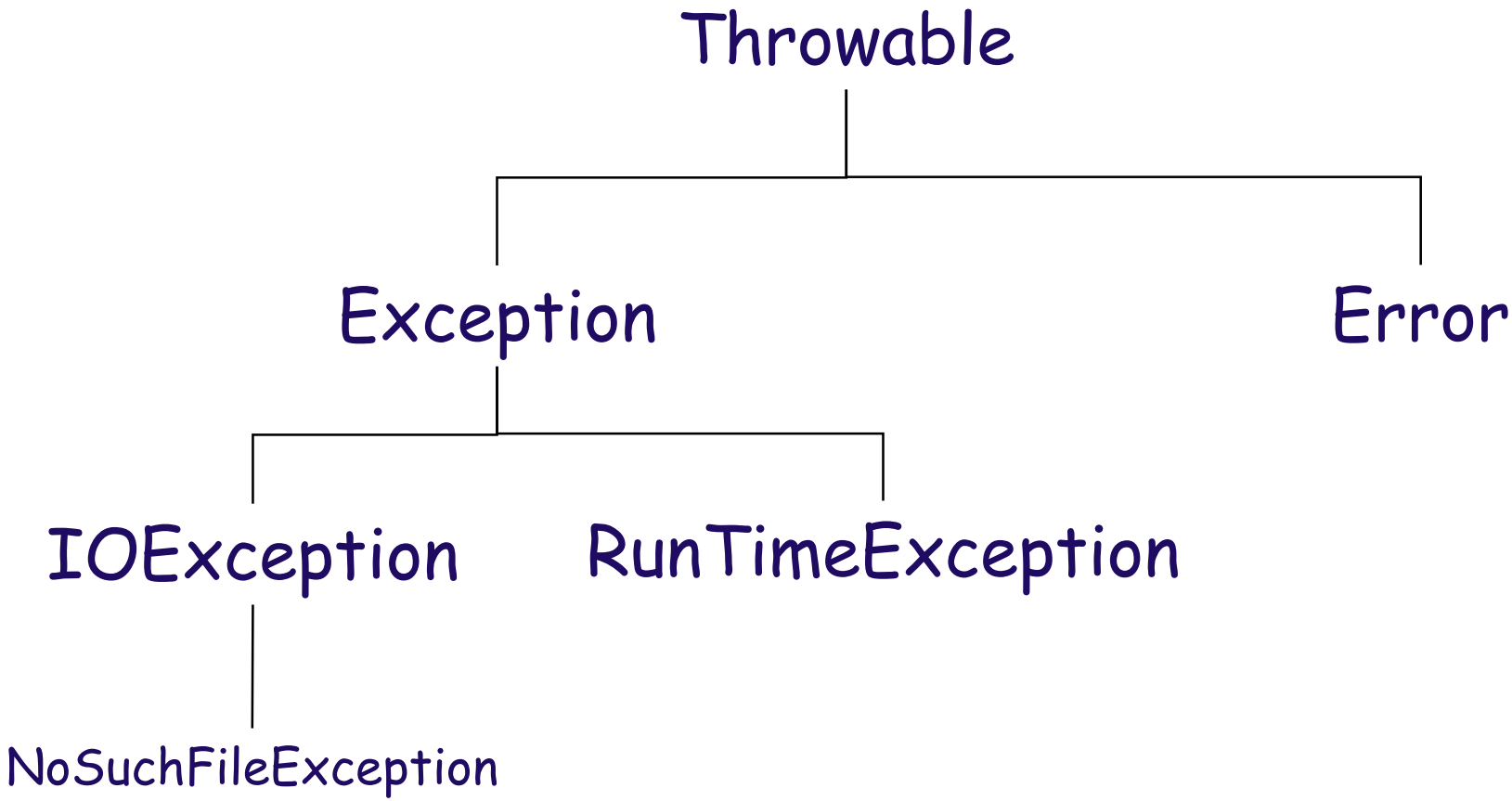
NullPointerException

SystemException

RunTimeExceptions do not need to be declared in the method head.

The programmer can construct subclasses of RunTimeExceptions.

Throwable Hierarchy



Typical Exception Handling

- Keywords are:
  - **`try`**: execute some code (known as a **try-block**) in which an exception might be thrown
  - **`catch`**: handle the exception if it is thrown
  - **`finally`**: *optional* code executed after a try-bock **whether or not** an exception was thrown

Problem: Opening a File

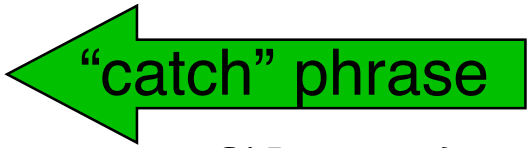
- A named file might not exist
- Attempting to open a non-existent file will throw an

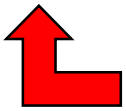
  `FileNotFoundException`

- Need to catch, or will not compile

Correct Version

```
InputStream inStream = System.in;

if( arg.length > 0 )
  {
  String filename = arg[0];
  try
    {
    inStream = new FileInputStream(filename);
    }
  catch( FileNotFoundException e )
    {
    System.err.println("*** unable to open file: " + filename);
    System.exit(1);
    }
  }
```





terminates program

Generally >1 Exception Type

```
try
  {
  . . .
  }
catch( ExceptionType1 e )
  {
  . . .
  }
catch( ExceptionType2 e )
  {
  . . .
  }
. . .

finally
  {
  . . .
  }
```

optional, always executed if present whether or not there is an exception

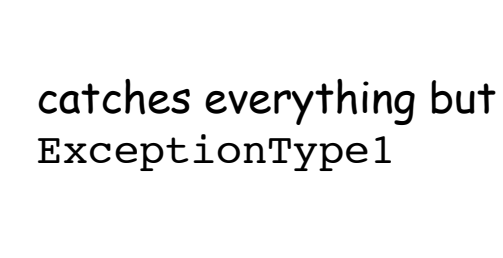
`finally` Code Example

```
class Contains
{
public static void main(String arg[])
  {
  if( arg.length != 2 )
    {
    System.err.println("usage: filename word"
    System.exit(1);
    }

  String filename = arg[0];
  String word = arg[1];

  FileInputStream stream = null;
  StreamTokenizer input;
  try
    {
    stream = new FileInputStream(filename);
    input = new StreamTokenizer(stream);
    boolean found = false;
    while( input.nextToken() != StreamTokenizer.TT_EOF )
      {
      if( word.equals(input.sval) )
        {
        found = true;
        break;
        }
      }
```

```
  catch( IOException e )
    {
    System.err.println("IO exception opening or reading file '
    }
  finally
    {
    if( stream != null )
      {
      try
        {
        stream.close();
        }
      catch( IOException e )
        {
        System.err.println("IO exception closing file " + file
        }
      }
    }
  }
}
```

Catch-all for Exceptions

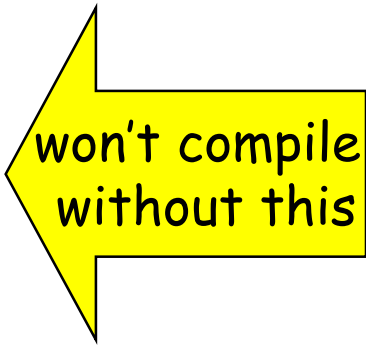
```
try
  {
  . . .
  }
catch( ExceptionType1 e )
  {
  . . .
  }
catch( Exception e )
  {
  . . .
  }
```



Declaring

If a method throws an exception, this fact must be **declared**:

```
void myMethod() throws MyException
{
… throw new MyException(msg);
}
```



Declaring

If a method *passes* on an exception, this fact must be **declared *as if*** the method throws it:

```
void myMethod() throws FileNotFoundException
{
inStream = new FileInputStream(filename);
}
```

Stopping the buck from being passed

If a method catches an exception, do **not** declare that it throws it, unless it does:

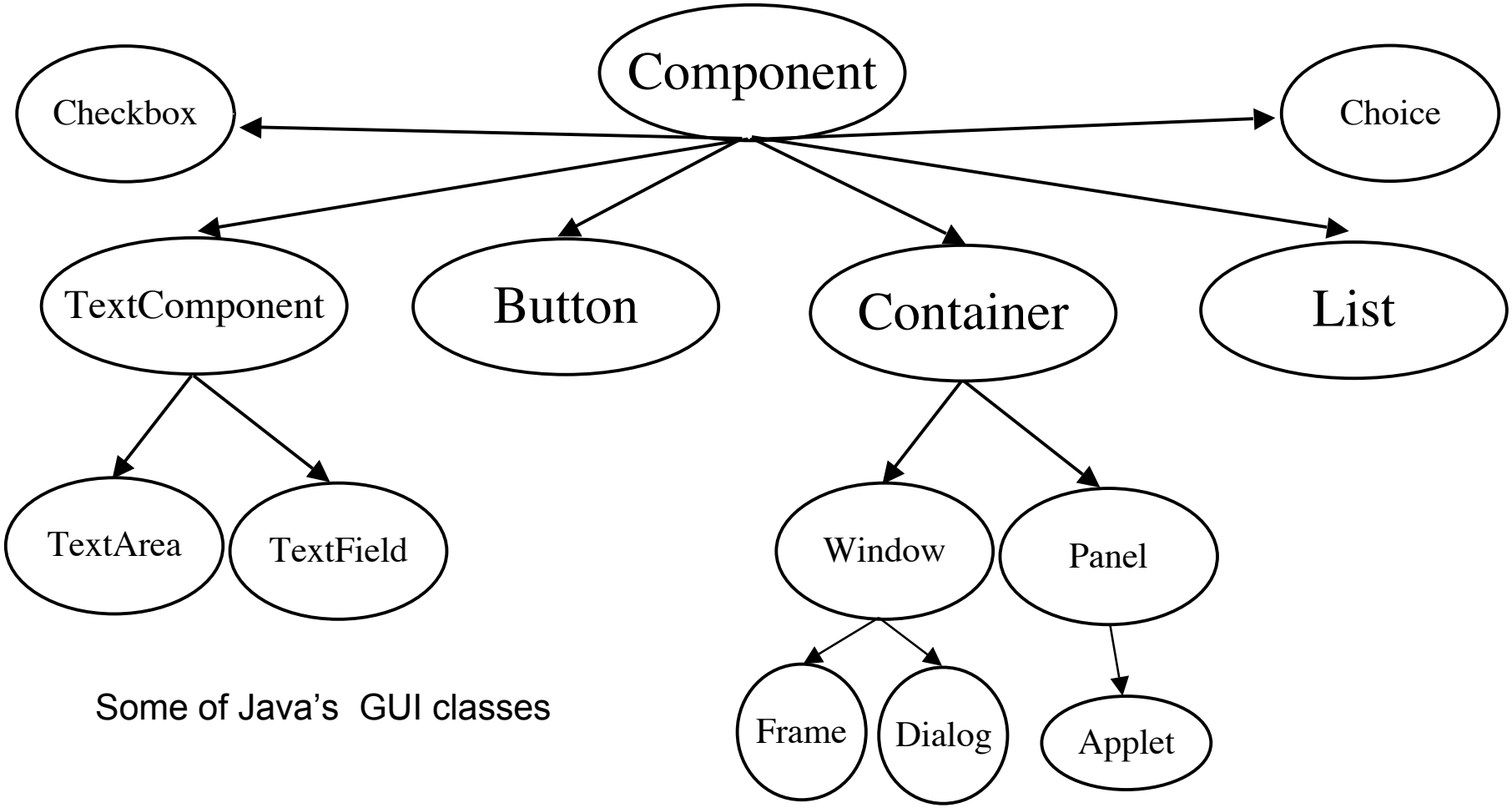
```
void myMethod() throws FileNotFoundException
{
try
  {
   inStream = new FileInputStream(filename);
  }
catch( FileNotFoundException e)
  {
  }
}
```

Exception on Declaration Rule

- A subclass of `RuntimeException` does not have to be declared.
- Example: `OpenlistException`

```
class OpenlistException extends RuntimeException
{
. . .
}
```

Inheritance Application: Graphics



Some of Java's GUI classes

Drawing Calls

- The drawing commands are encapsulated in the Graphics class (*graphics* is the data member's name)

---

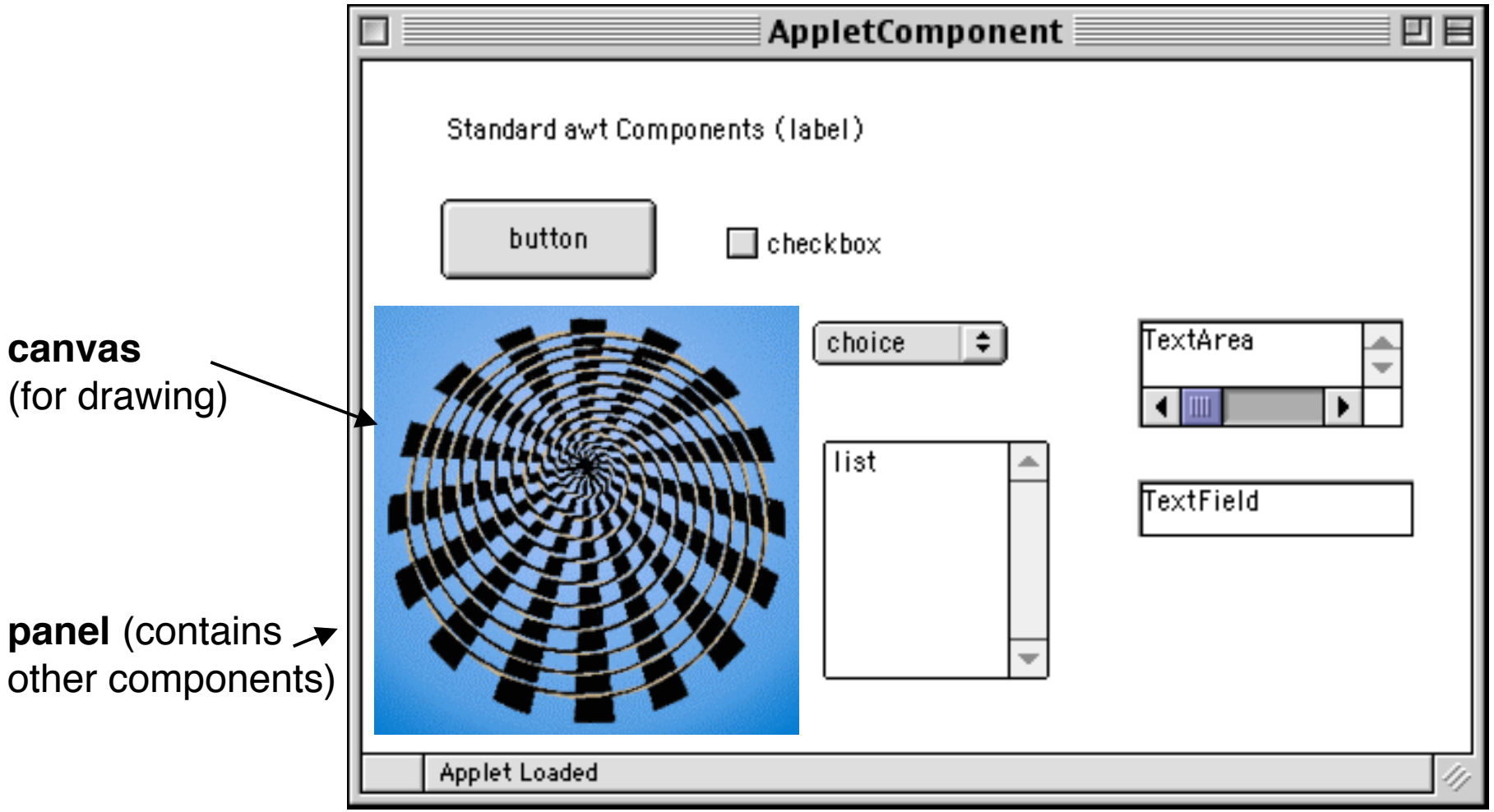
void setColor(Color c)

void fillRect(int x, int y, int width, int height)
void fillOval(int x, int y, int width, int height)
void fillPolygon(int[] xPoints, int[] yPoints, int nPoints)

Each of the above have "draw" versions:
drawRect, drawOval, drawPolygon

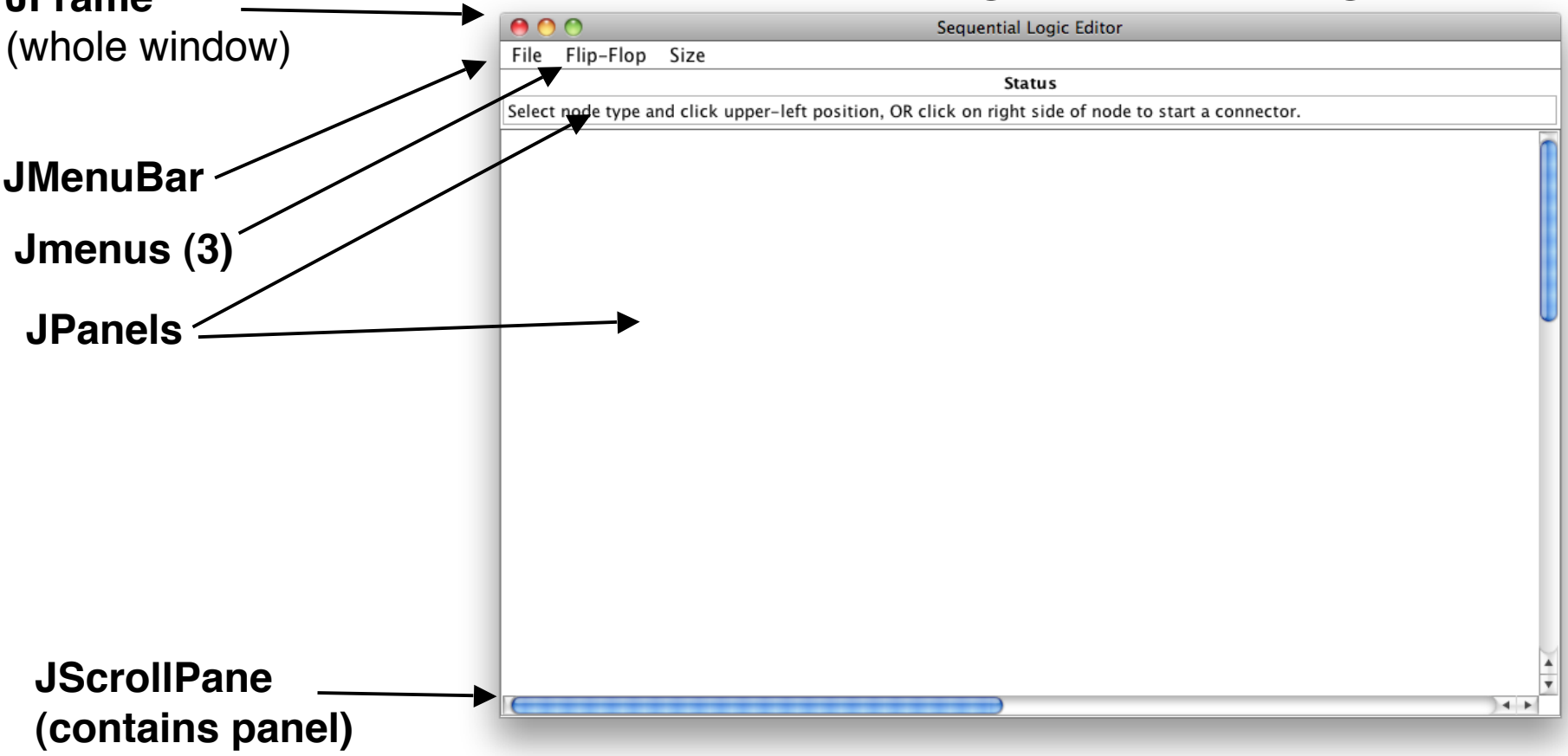
void drawString(String str, int x, int y)
void drawLine(int x1, int y1, int x2, int y2)
void drawImage(Image img, int x, int y, null)

AWT Graphics Components (old)



“Swing” Graphics Components

“J” Prefix designates Swing

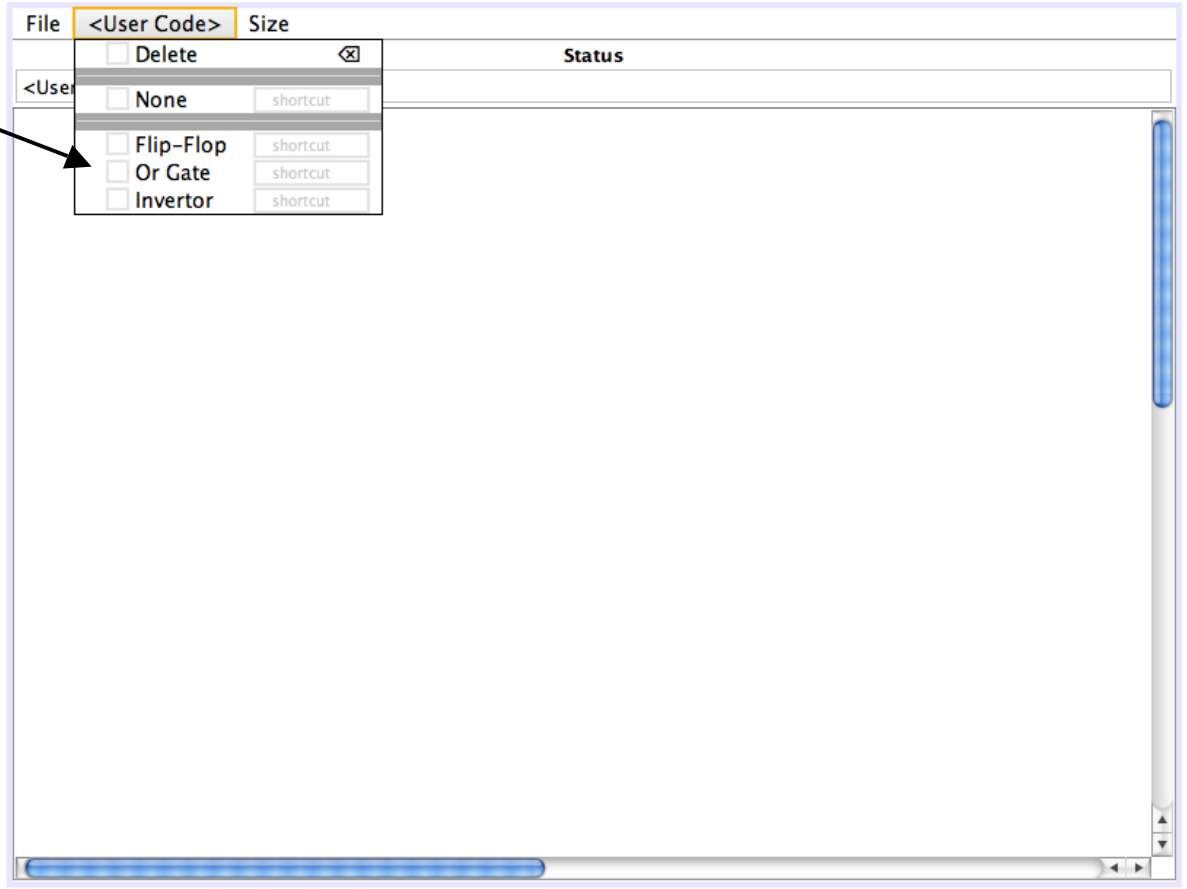


Customization by Inheritance


```
public class MainFrame extends javax.swing.JFrame
{
. . .
}
```

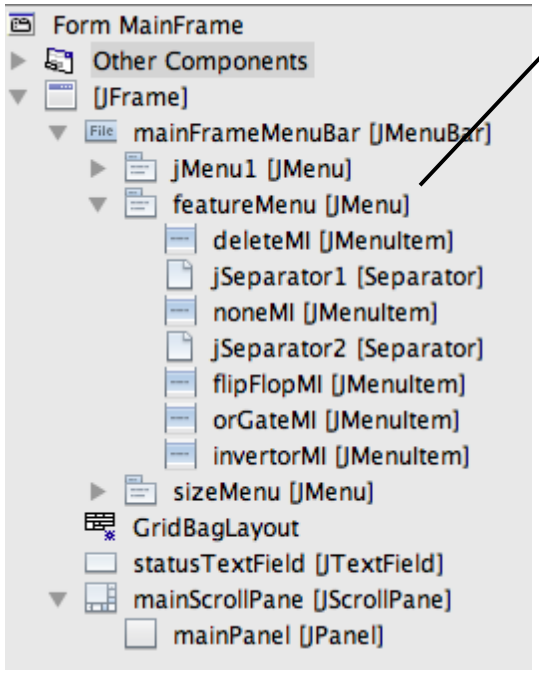
View from Netbeans GUI Builder

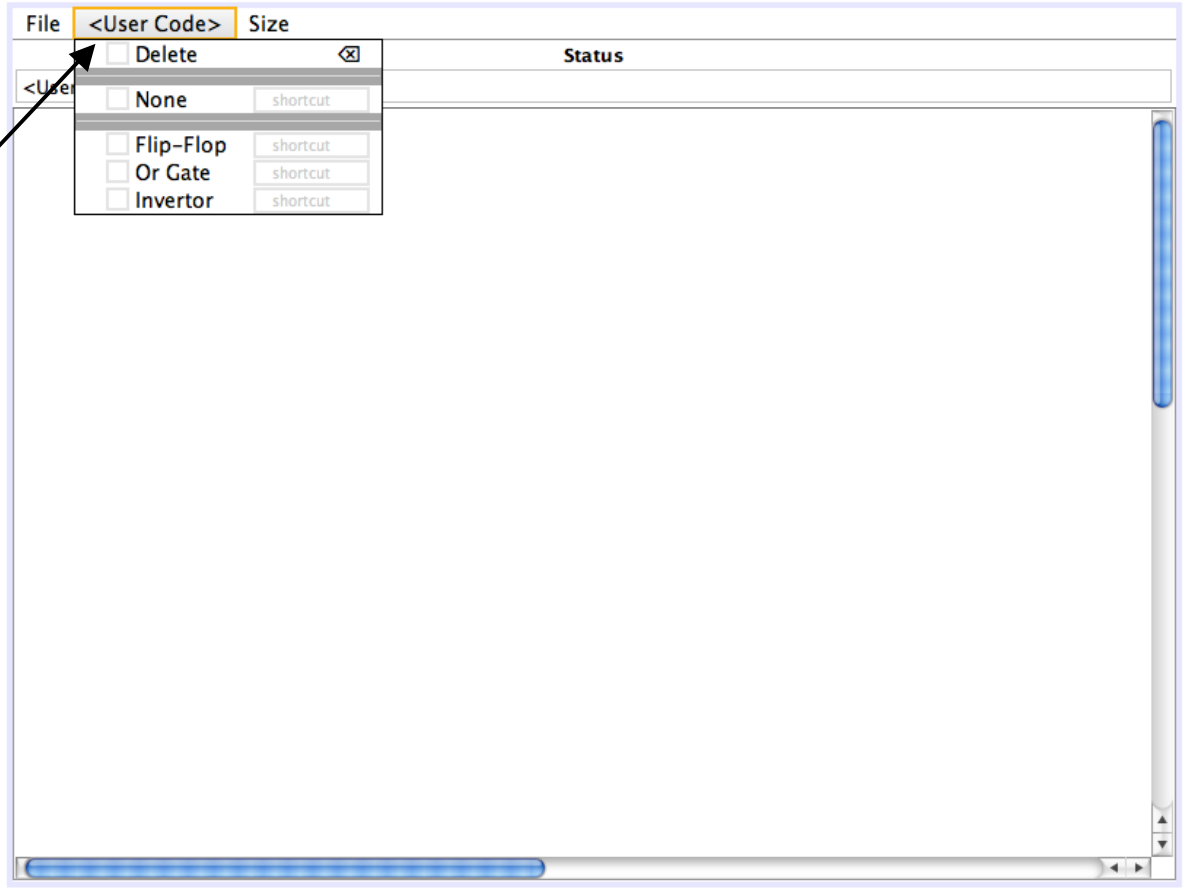
JMenuItems (5)



Design View with GUI Inspector

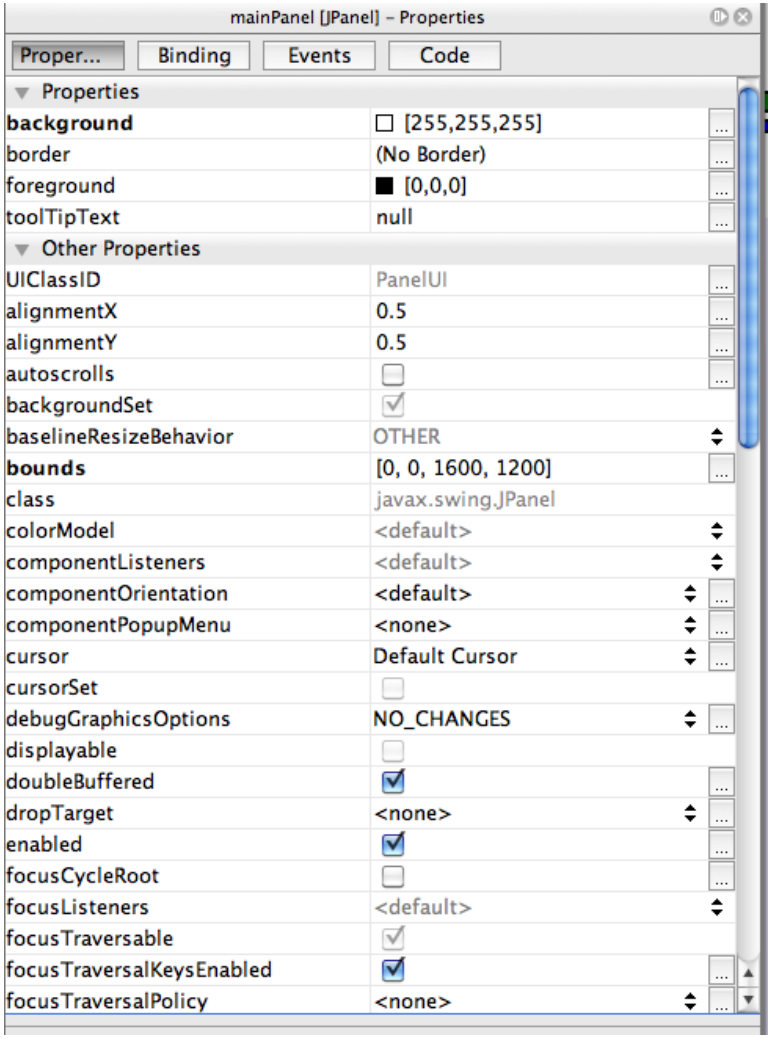
Navigator view



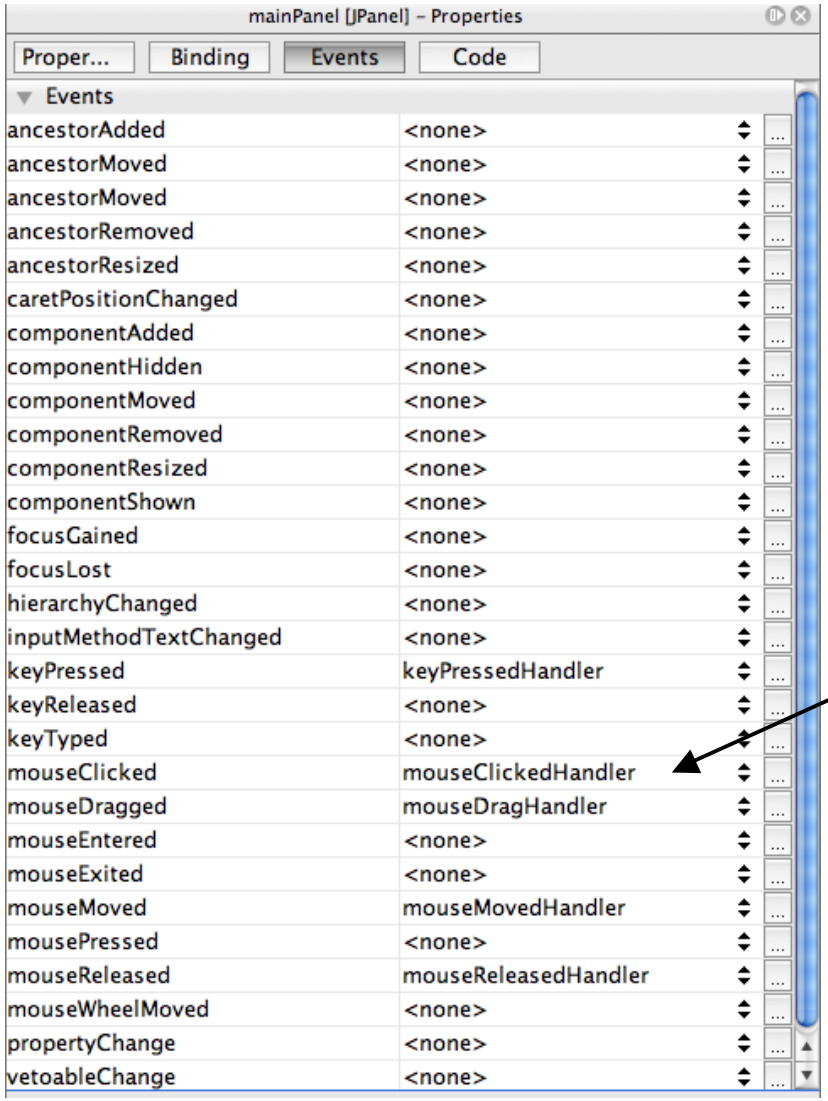


Properties View of Main Panel (partial)

A large amount of code is generated from this panel specification.



Events Properties for Main Panel



Some (Swing) Graphics-Related Interfaces

```
interface KeyListener
{
    void keyPressed(KeyEvent evt);
    void keyReleased(KeyEvent evt);
    void keyTyped(KeyEvent evt);
}

interface MouseListener
{
    void mousePressed(MouseEvent evt);
    void mouseDragged(MouseEvent evt);
        . . .
}

interface ActionListener
{
    void actionPerformed(ActionEvent evt);
}
```

Generated Code for Event Handling

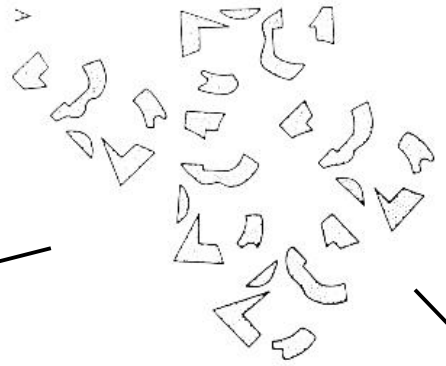
```
mainPanel.addMouseListener(new java.awt.event.MouseAdapter()
{
            public void mouseReleased(java.awt.event.MouseEvent evt) {
                mouseReleasedHandler(evt);
            }
            public void mouseClicked(java.awt.event.MouseEvent evt) {
                mouseClickedHandler(evt);
            }
        });

mainPanel.addMouseMotionListener(new java.awt.event.MouseMotionAdapter() {
            public void mouseMoved(java.awt.event.MouseEvent evt) {
                mouseMovedHandler(evt);
            }
            public void mouseDragged(java.awt.event.MouseEvent evt) {
                mouseDragHandler(evt);
            }
        });

mainPanel.addKeyListener(new java.awt.event.KeyAdapter() {
            public void keyPressed(java.awt.event.KeyEvent evt) {
                keyPressedHandler(evt);
            }
        });
```

User Code for Event Handling

```
/**
 * Handle mouse clicks, single and double
 @param evt
 */

private void mouseClickedHandler(java.awt.event.MouseEvent evt)
  {
  System.out.println("\nMouse clicked at " + evt.getX() + ", " + evt.getY();

    switch( evt.getClickCount() )
      {
        case 1: // Single Click
            analyzeSingleClick(evt);
            break;

        case 2: // Double Click
            analyzeDoubleClick(evt);
            break;
      }
}
```

Events

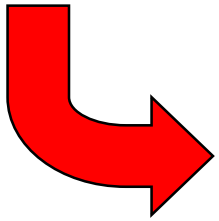
- Events are things that happen to a graphical application
  - Button Presses
  - Key Events
  - Text Entries
  - Mouse Events
- Each object receiving an event notifies its “Listener”
- The Listener then handles the event appropriately



```
public void keyPressed(KeyEvent evt)
{
    graphics.setColor(Color.white);
    graphics.fillRect(300,180,100,40);
    graphics.setColor(Color.black);
    graphics.drawString("Key " + evt.getKeyChar() + " pressed",300,200);
    repaint();
}
```

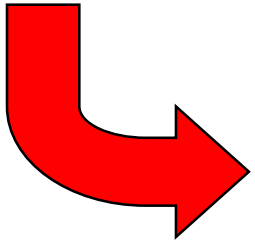
Drawing

some event occurs, such as calling **repaint**() or making the window visible



These methods can be overridden.

update() is called on the window's graphics, which then calls **paint**()



display by drawing inside paint

Double Buffer to Avoid Flicker

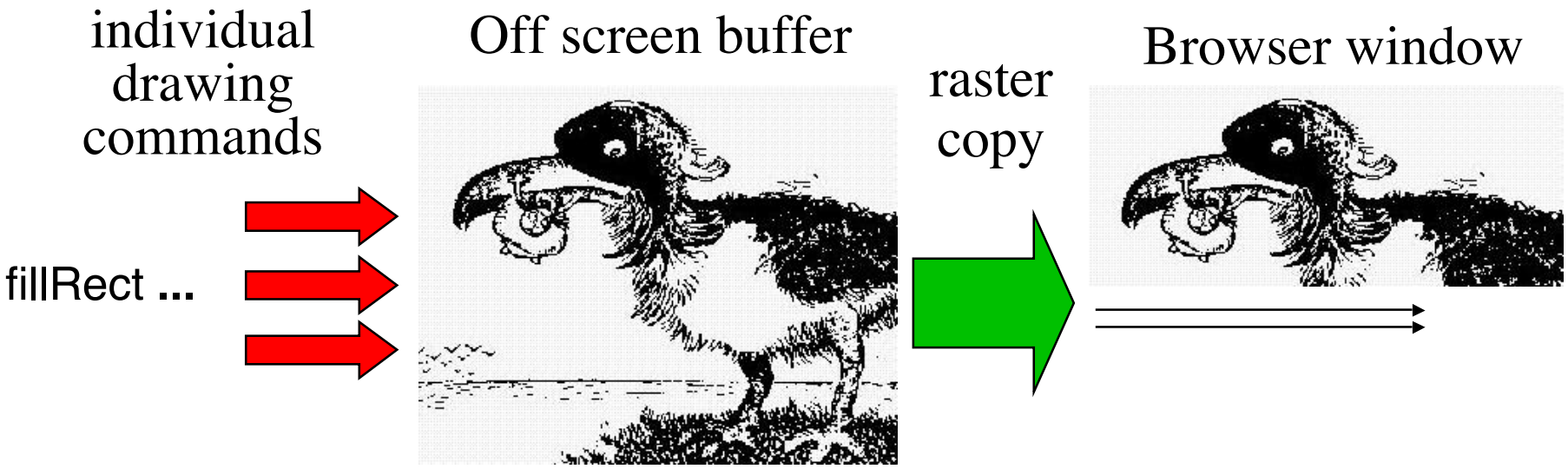


image
graphics
repaint()

```
public void update(Graphics g) {
    paint(g);
}

public void paint(Graphics g)    {
    g.drawImage(image, 0, 0, null);
}
```

Double Buffer Creation Code

User declarations inside the JFrame extension:

```
/**
 * Image providing off-screen buffer.
 */

Image buffer;              // Image to be drawn on screen by paint method

Graphics graphics = null; // The Graphics of that buffer.
```

Buffer initialization code:

```
 buffer = mainPanel.createImage(mainPanel.getWidth(),
                                mainPanel.getHeight());

graphics = buffer.getGraphics();
```

Double Buffer Creation Code

User declarations inside the JFrame extension:

```
/**
 * Image providing off-screen buffer.
 */

Image buffer;              // Image to be drawn on screen by paint method

Graphics graphics = null; // The Graphics of that buffer.
```

Buffer initialization code:

```
buffer = mainPanel.createImage(mainPanel.getWidth(),
                               mainPanel.getHeight());

graphics = buffer.getGraphics();
```

Painting Code

(example not over-riding mainPanel's paint method)

```
mainPanel.getGraphics().drawImage(buffer, 0, 0, null);

mainPanel.paintAll(buffer.getGraphics());
```