Applets and Basic Graphics

Originals of slides and source code for examples: http://courses.coreservlets.com/Course-Materials/java.html
Also see Java 8 tutorial: http://www.coreservlets.com/java-8-tutorial/ and many other Java EE tutorials: http://www.coreservlets.com/
Customized Java training courses (onsite or at public venues): http://courses.coreservlets.com/java-training.html

Developed and taught by well-known author and developer. At public venues or onsite at your location.

Customized Java EE Training: http://courses.coreservlets.com/
Java 7, Java 8, JSF 2, PrimeFaces, Android, JSP, Ajax, jQuery, Spring MVC, RESTful Web Services, GWT, Hadoop
Developed and taught by well-known author and developer. At public venues or onsite at your location.

For customized Java-related training at your organization, email hall@coreservlets.com
Marty is also available for consulting and development support

Taught by lead author of Core Servlets & JSP, co-author of Core JSF (4th Ed), & this tutorial. Available at public venues, or customized versions can be held on-site at your organization.

• Courses developed and taught by Marty Hall
  – JSF 2.2, PrimeFaces, servlets/JSP, Ajax, JavaScript, jQuery, Android, Java 7 or 8 programming, GWT, custom mix of topics
  – Courses available in any state or country. Maryland/DC area companies can also choose afternoon/evening courses.
• Courses developed and taught by coreservlets.com experts (edited by Marty)
  – Spring MVC, Core Spring, Hibernate/JPA, Hadoop, HTML5, RESTful Web Services
Contact hall@coreservlets.com for details

© 2015 Marty Hall

© 2015 Marty Hall
Topics in This Section

• Applet restrictions
• Basic applet and HTML template
• The value of @Override
• The applet lifecycle
• Customizing applets through HTML parameters
• Methods available for graphical operations
• Loading and drawing images
• Using try/catch blocks (part 1)
• Controlling image loading

Motivation
Why Look at Applets?

- **Applets are now used rarely in real life**
  - Very common in early days, now mostly relegated to relatively rare intranet applications.

- **Value**
  - Natural excuse to cover other topics
    - Life-cycle methods
    - Inheritance in action
    - Import statements
    - Try/catch blocks
    - @Override
    - Background threads
  - Chance to introduce simple graphics
    - To make exercises more interesting
  - Applets still used *some* in real life
    - Not so common, but not *entirely* gone

Security Restrictions
Applets Cannot…

• **Read from the local (client) disk**
  – Applets cannot read arbitrary files
  – They can, however, instruct the browser to display pages that are generally accessible on the Web, which might include some local files

• **Write to the local (client) disk**
  – The browser may choose to cache certain files, including some loaded by applets, but this choice is not under direct control of the applet

• **Open network connections other than to the server from which the applet was loaded**
  – This restriction prevents applets from browsing behind network firewalls

Applets Cannot…

• **Link to client-side C code or call programs installed on the browser machine**
  – Ordinary Java applications can invoke locally installed programs (Runtime.exec or ProcessBuilder.start) as well as link to local C/C++ modules (“native” methods)
  – These actions are prohibited in applets because there is no way to determine whether the operations these local programs perform are safe

• **Discover private information about the user**
  – Applets should not be able to discover the username of the person running them or specific system information such as current users, directory names or listings, system software, and so forth
  – However, applets can determine the name of the host they are on; this information is already reported to the HTTP server that delivered the applet
Applet Basics

Customized Java EE Training: http://courses.coreservlets.com/
Java 7, Java 8, JSF 2, PrimeFaces, Android, JSP, Ajax, jQuery, Spring MVC, RESTful Web Services, GWT, Hadoop
Developed and taught by well-known author and developer. At public venues or onsite at your location.

Applet Java Template

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

public class AppletTemplate extends Applet {

    // Variable declarations

    @Override // More on @Override later
    public void init() {
        // Variable initializations, image loading, etc.
    }

    @Override // More on @Override later
    public void paint(Graphics g) {
        // Drawing operations
    }
}
```
Browser Caching Problems

- **Browsers normally cache applets**
  - So, when you recompile applet and reload HTML page, you still see old applet. Very annoying problem!

- **Browser solution**
  - Open Java console (see later slide) and hit “x”

- **Eclipse solution**
  - R-click, Run As → Java Applet (calls appletviewer)
    - Use this on your exercises!
  - Problem: doesn’t use your HTML file, so you can’t test if the WIDTH and HEIGHT are correct. Still, convenient.

- **Appletviewer solution**
  - During development, can call “appletviewer file.html” or “appletviewer http://.../file.html” from command prompt.

---

Applet HTML Template

```html
<!DOCTYPE HTML>
<html>
<head>
  <title>A Template for Loading Applets</title>
</head>
<body>
  <h1>A Template for Loading Applets</h1>
  <p>
    <applet code="AppletTemplate.class" width="120" height="60">
      <b>Error! You must use a Java-enabled browser.</b>
    </applet>
  </p>
</body>
</html>
```

The HTML 5 spec says to use “object” instead of “applet”. However, “object” is more complex, and it fails on multiple IE versions. So, “applet” is still widely used, even with the HTML 5 DOCTYPE.

Besides, for practice you normally use the Eclipse trick (click in Java code, Run As → Java Applet) and don’t make an HTML file at all.
Applet Example 1: Drawing Diagonal Lines

import java.applet.Applet;
import java.awt.*;

/** Draws a line from top-left halfway to bottom-right. */
public class DrawLine extends Applet {
    private int width;   // Instance var declarations here
    private int height;

    @Override
    public void init() {           // Initializations here
        setBackground(Color.YELLOW);
        setForeground(Color.RED);
        width = getWidth();
        height = getHeight();
    }

    @Override
    public void paint(Graphics g) {  // Drawing here
        g.drawLine(0, 0, width/2, height/2);
    }
}

Applet Example 1: HTML

<!DOCTYPE HTML>
<html>
<head>
    <title>Drawing Diagonal Lines</title>
</head>
<body bgcolor="black" text="white">
<h1>Drawing Diagonal Lines</h1>
<p>
    <applet code="DrawLine.class" width="300" height="200">
        <b>Sorry, this example requires Java.</b>
    </applet>
</p>
<p>
    <applet code="DrawLine.class" width="500" height="400">
        <b>Sorry, this example requires Java.</b>
    </applet>
</p>
</body></html>
Applet Example 1: Result

import java.applet.Applet;
import java.awt.*;
/** An applet that draws an image. */
public class JavaJump extends Applet {
    private Image jumpingJava; // Instance var declarations here
    public void init() {       // Initializations here
        setBackground(Color.WHITE);
        setFont(new Font("SansSerif", Font.BOLD, 18));
        jumpingJava = getImage(getCodeBase(),
            "images/Jumping-Java.gif");
        add(new Label("Great Jumping Java!"));
        System.out.println("Yow! I'm jiving with Java.");
    }

    public void paint(Graphics g) {  // Drawing here
        g.drawImage(jumpingJava, 0, 50, this);
    }
}

Hey! You forgot @Override and it still works. Why?
And if it works without @Override, why is @Override useful?
Good question: see upcoming slides

Applet Example 2

import java.applet.Applet;
import java.awt.*;
/** An applet that draws an image. */
public class JavaJump extends Applet {
    private Image jumpingJava; // Instance var declarations here
    public void init() {       // Initializations here
        setBackground(Color.WHITE);
        setFont(new Font("SansSerif", Font.BOLD, 18));
        jumpingJava = getImage(getCodeBase(),
            "images/Jumping-Java.gif");
        add(new Label("Great Jumping Java!"));
        System.out.println("Yow! I'm jiving with Java.");
    }

    public void paint(Graphics g) {  // Drawing here
        g.drawImage(jumpingJava, 0, 50, this);
    }
}
<!DOCTYPE HTML>
<html>
<head>
<title>Jumping Java</title>
</head>
<body bgcolor="black" text="white">
<h1>Jumping Java</h1>
<p><applet code="JavaJump.class" width="250" height="335">
<b>Sorry, this example requires Java.</b>
</applet></p>
</body></html>
Problem: Odd Runtime Behavior

```java
public class MyApplet extends Applet {
    public void Paint(Graphics g) {
        drawSomethingCool();
    }
}
```

- **What happens when you run this?**
  - No compilation error
  - No error message
  - Nothing cool is drawn

Solution: Use `@Override`

```java
public class MyApplet extends Applet {
    @Override
    public void Paint(Graphics g) {
        drawSomethingCool();
    }
}
```

- **What happens when you run this?**
  - Won’t compile, since there is no “Paint” method in the parent class that takes Graphics as an argument. Once you change it to “paint”, it compiles normally.
  - You can override methods with or without `@Override`, but `@Override` expresses design intent and also catches errors at compile time instead of run time. (@FunctionalInterface, covered later, is similar)
Applet Life Cycle and Methods

The Applet Life Cycle

• **public void init()**
  – Called when applet is first loaded into the browser.
  – *Not* called each time the applet is executed

• **public void paint(Graphics g)**
  – Called by the browser after init and start
  – Reinvoked whenever the browser redraws the screen (typically when part of the screen has been obscured and then reexposed)
  – Use this for your drawing code
The Applet Life Cycle (Continued)

- **public void start()**
  - Called immediately after init initially
  - Reinvoked each time user returns to page after having left
  - Used to start animation threads

- **public void stop()**
  - Called when the user leaves the page
  - Used to stop animation threads

- **public void destroy()**
  - Called when applet is killed by the browser

Useful Applet Methods

- **getCodeBase, getDocumentBase**
  - The URL of the:
    - Applet file - `getCodeBase`
    - HTML file - `getDocumentBase`

- **getParameter**
  - Retrieves the value from the associated HTML `PARAM` element

- **getWidth, getHeight**
  - Returns the width/height of the applet

- **getGraphics**
  - Retrieves the current `Graphics` object for the applet
  - The `Graphics` object does not persist across paint invocations
Useful Applet Methods (Continued)

• **showDocument (AppletContext method)**
  
  ```java
  getAppletContext().showDocument(...)
  ```

  – Asks the browser to retrieve and display a Web page
  – Can direct page to a named FRAME cell

• **showStatus**
  
  – Displays a string in the status line at the bottom of the browser

• **getCursor, setCursor**
  
  – Defines the Cursor for the mouse, for example, CROSSHAIR_CURSOR, HAND_CURSOR, WAIT_CURSOR

Useful Applet Methods (Continued)

• **getAudioClip, play**
  
  – Retrieves an audio file from a remote location and plays it
  – Java supports MIDI, .aiff, .wav, etc.

• **getBackground, setBackground**
  
  – Gets/sets the background color of the applet
  – SystemColor class provides access to desktop colors

• **getForeground, setForeground**
  
  – Gets/sets foreground color of applet (default color of drawing operations)
Using Appletviewer

- **Real-life applets**
  - Deployed to a Web site (usually intranet) and then run in a browser. You must digitally sign the class files. Rarely done and not important to learn.

- **Applets for practice**
  - Run in appletviewer. For learning other concepts and for doing simple graphics. Useful to learn.

- **Running applets for practice**
  - Run in Eclipse (uses appletviewer behind the scenes)
    - R-click in Java code, Run As → Java applet
      - No HTML file is needed
  - Run in appletviewer
    - Go to directory containing HTML, appletviewer file.html

---

**HTML applet Element (Not Needed if Running in Eclipse)**

```html
<applet code="..." width="www" height="hhh" ...
...
</applet>
```

- **Required Attributes**
  - code
    - Designates the filename of the Java class file to load
    - Filename interpreted with respect to directory of current HTML page (default) unless codebase is supplied
  - width and height
    - Specifies area the applet will occupy
    - Values can be given in pixels or as a percentage of the browser window (width only). Percentages fail in appletviewer.
HTML applet Element
(Only if Running in Browser)

• Other Attributes
  – align, hspace, and vspace
    • Controls position and border spacing. Exactly the same as with the IMG element
  – archive
    • Designates JAR file (zip file with .jar extension) containing all classes and images used by applet
    • Save considerable time when downloading multiple class files
  – name
    • Names the applet for interapplet and JavaScript communication
  – mayscript (nonstandard)
    • Permits JavaScript to control the applet

Applet Parameters:
Letting the HTML Author Supply Data
Reading Applet Parameters

• **Motivation**
  – Lets the HTML author pass data to the program

• **HTML**
  – Use embedded param tag inside applet

• **Java**
  – Call `getParameter` and pass name that matches param
  – The name is case sensitive
  – `getParameter` returns null if there is no matching entry in the param tag
  – Always plan for missing data

Setting Applet Parameters: HTML

```html
<h1>Customizable HelloWWW Applet</h1>
<p>
<applet code="HelloWWW2.class" width="400" height="40">
  <param name="BACKGROUND" value="LIGHT">
  <b>Error! You must use a Java-enabled browser.</b>
</applet>
</p>
<p>
<applet code="HelloWWW2.class" width="400" height="40">
  <param name="BACKGROUND" value="DARK">
  <b>Error! You must use a Java-enabled browser.</b>
</applet>
</p>
<p>
<applet code="HelloWWW2.class" width="400" height="40">
  <b>Error! You must use a Java-enabled browser.</b>
</applet>
</p>
```
public void init() {
    Color background = Color.GRAY;
    Color foreground = Color.DARK_GRAY;
    String backgroundType = getParameter("BACKGROUND");
    if (backgroundType != null) {
        if (backgroundType.equalsIgnoreCase("LIGHT")) {
            background = Color.WHITE;
            foreground = Color.BLACK;
        } else if (backgroundType.equalsIgnoreCase("DARK")) {
            background = Color.BLACK;
            foreground = Color.WHITE;
        }
    }
    ...
}
Drawing in Applets

Basic Approach

• **Put code in paint**
  - Put drawing code in the paint method. Use `@Override`.
    - Later, when we learn about handling mouse events, we can also draw from an event handler.

• **Use a Graphics (i.e., pen) object to draw**
  - There is no drawLine method of Applet. You use the drawLine method of the Graphics object passed to paint.
  - In paint, you are given Graphics object. In other methods of Applet subclass, call `getGraphics()` to obtain it.

• **Understand the coordinate system**
  - The top-left corner of the applet is (0,0)
  - x is to the right
  - y is down
Quick example

```java
@Override
public void paint(Graphics g) {
    g.drawLine(0, 0, 100, 100);
}
```

Draws a line from top-left corner of applet, diagonally down and to the right.

Useful Graphics Methods

- `drawString(string, left, bottom)`
  - Draws a string in the current font and color with the `bottom left` corner of the string at the specified location
  - One of the few methods where the y coordinate refers to the bottom of shape, not the top. But y values are still with respect to the `top left` corner of the applet window

- `drawRect(left, top, width, height)`
  - Draws the outline of a rectangle (1-pixel border) in the current color

- `fillRect(left, top, width, height)`
  - Draws a solid rectangle in the current color

- `drawLine(x1, y1, x2, y2)`
  - Draws a 1-pixel-thick line from `(x1, y1)` to `(x2, y2)`
Useful Graphics Methods (Continued)

- **drawOval, fillOval**
  - Draws an outlined and solid oval, where the arguments describe a rectangle that bounds the oval

- **drawPolygon, fillPolygon**
  - Draws an outlined and solid polygon whose points are defined by arrays or a `Polygon` (a class that stores a series of points)
  - By default, polygon is closed; to make an open polygon use the drawPolyline method

- **drawImage**
  - Draws an image
  - Images can be in JPEG or GIF (including animated GIF) format

Drawing Color

- **setColor, getColor**
  - Specifies the foreground color prior to drawing operation
  - By default, the graphics object receives the foreground color of the window
    - As set via setForeground from the init() method
  - AWT has 16 predefined colors (Color.RED, Color.BLUE, etc.) or create your own color: `new Color(r, g, b)`
  - Changing the color of the Graphics object affects only the drawing that explicitly uses that Graphics object
    - To make permanent changes, call the applet’s setForeground method.
Graphics Font

• setFont, getFont
  – Specifies the font to be used for drawing text
  – Determine the size of a character through FontMetrics or LineMetrics
  – Setting the font for the Graphics object does not persist to subsequent invocations of paint
  – Set the font of the window (i.e., call the applet’s setFont method) for permanent changes to the font
  – Standard Java font names: Serif (e.g., Times New Roman), SansSerif (e.g., Arial), Monospaced (e.g., Courier New), Dialog, and DialogInput

Graphic Drawing Modes

• setXORMode
  – Specifies a color to XOR with the color of underlying pixel before drawing the new pixel
  – Drawing something twice in a row will restore the original condition

• setPaintMode
  – Set drawing mode back to normal (versus XOR)
  – Subsequent drawing will use the normal foreground color
  – Remember that the Graphics object is reset to the default each time. So, no need to call g.setPaintMode() in paint unless you do non-XOR drawing after your XOR drawing
import java.applet.Applet;
import java.awt.*;

public class DrawLine extends Applet {
    private int width;  // Instance var declarations here
    private int height;

    @Override
    public void init() {
        // Initializations here
        setBackground(Color.YELLOW);
        setForeground(Color.RED);
        width = getWidth();
        height = getHeight();
    }

    @Override
    public void paint(Graphics g) {
        g.drawLine(0, 0, width/2, height/2);
    }
}

Draws a line from top-left halfway to bottom-right.

<!DOCTYPE HTML>
<html>
<head>
    <title>Drawing Diagonal Lines</title>
</head>
<body bgcolor="black" text="white">
<h1>Drawing Diagonal Lines</h1>
<p>
    <applet code="DrawLine.class" width="300" height="200">
        <b>Sorry, this example requires Java.</b>
    </applet>
</p>
<p>
    <applet code="DrawLine.class" width="500" height="400">
        <b>Sorry, this example requires Java.</b>
    </applet>
</p>
</body></html>
Result

Drawing Diagonal Lines

Loading and Drawing Images

Customized Java EE Training: http://courses.coreservlets.com/
Java 7, Java 8, JSF 2, PrimeFaces, Android, JSP, Ajax, jQuery, Spring MVC, RESTful Web Services, GWT, Hadoop
Developed and taught by well-known author and developer. At public venues or onsite at your location.
Overview

- **Make an Image instance variable**
  - private Image myImage;
- **Load the image from init**
  - myImage = getImage(getCodeBase(), "file");
  - myImage = getImage(url);
- **Draw the image in paint**
  - g.drawImage(myImage, x, y, window);
  - g.drawImage(myImage, x, y, width, height, window)

  • In applet, use “this” for the window

---

Loading Applet Image from Relative URL

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

public class JavaMan1 extends Applet {
    private Image javaMan;

    @Override
    public void init() {
        javaMan = getImage(getCodeBase(), "images/Java-Man.gif");
    }

    @Override
    public void paint(Graphics g) {
        g.drawImage(javaMan, 0, 0, this);
    }
}
```
@Override
public void init() {
    try {
        URL imageFile =
            new URL("http://www.corewebprogramming.com" +
                "/images/Java-Man.gif");
        javaMan = getImage(imageFile);
    } catch (MalformedURLException mue) {
        showStatus("Bogus image URL.");
        System.out.println("Bogus URL");
    }
}
Basic Try/Catch Blocks
(More Features in Section on File IO)

Exceptions and try/catch Blocks

- **Handling exceptions**
  - If your code potentially has an error (“throws an exception”), you must either use a try/catch block or throw the exception, and let the calling code handle it.
    - There are a few error types (e.g., NullPointerException, ArrayIndexOutOfBoundsException) which you are not required to handle.

- **Basic form**
  ```java
  try {
      statement1;
      statement2;
      ...
  } catch(SomeExceptionClass someVar) {
      handleTheException(someVar);
  }
  ```
### More Info

<table>
<thead>
<tr>
<th>One Catch</th>
<th>Multiple Catches</th>
<th>Throwing</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>try {</code></td>
<td><code>try {</code></td>
<td><code>public void foo()</code></td>
</tr>
<tr>
<td><code>    statement1;</code></td>
<td><code>    statement1;</code></td>
<td><code>    throws Eclass1 {</code></td>
</tr>
<tr>
<td><code>    statement2;</code></td>
<td><code>    statement2;</code></td>
<td><code>    ...</code></td>
</tr>
<tr>
<td><code>...</code></td>
<td><code>...</code></td>
<td><code>)</code></td>
</tr>
<tr>
<td><code>} catch(Eclass var) {</code></td>
<td><code>} catch(Eclass1 var1) {</code></td>
<td><code>Instead of catching exception here, throw it to the calling code, which must catch the exception or throw it further.</code></td>
</tr>
<tr>
<td><code>    doBlah(var);</code></td>
<td><code>    ...</code></td>
<td><code>}</code></td>
</tr>
<tr>
<td><code>}</code></td>
<td><code>} catch(Eclass2 var2) {</code></td>
<td><code>}</code></td>
</tr>
<tr>
<td><code>Run all statements in try block. If everything finishes normally, you are done. If an exception of type Eclass occurs, jump down to catch block.</code></td>
<td><code>    ...</code></td>
<td><code>}</code></td>
</tr>
<tr>
<td><code>}</code></td>
<td><code>} catch(Eclass3 var3) {</code></td>
<td><code>First matching catch fires, so exceptions should be ordered from most specific to most general </code></td>
</tr>
<tr>
<td><code>...</code></td>
<td><code>    ...</code></td>
<td><code>• Due to inheritance, more than one match is possible</code></td>
</tr>
</tbody>
</table>

Covered in later lecture on File IO: finally blocks [`.catch()`.]. multicatch [`.catch(E1|E2 e)`], and try with resources [`.try(BufferedReader reader ...)`].

### Preview of Topics Covered in File IO Lecture

- **Covered here:** basics
  `try {` `statement1;` `statement2;` `...` `} catch(Eclass1 var1) {` `...` `} catch(Eclass2 var2) {` `...` `} catch(Eclass3 var3) {` `...` `}

- **New:** `.finally`
  `try {...` `
  `catch(...) {...` `
  `finally {
  `...` `
  `}

- **New:** `.multicatch`
  `try {...` `
  `catch(Eclass1 | Eclass e) {
  `...` `
  `}...`

- **New:** `.try with resources`
  `try (SomeAutoCloseable var = ...) {...` `
  `catch(...) {...` `
  `}...`
Advanced Topics

(These topics are specific to GUI programs. Skip this section if you are planning on only doing server-side apps and are using this lecture only to learn general topics like try/catch blocks, @Override, lifecycle methods, import statements, and so forth.)

Customized Java EE Training: http://courses.coreservlets.com/
Java 7, Java 8, JSF 2, PrimeFaces, Android, JSP, Ajax, jQuery, Spring MVC, RESTful Web Services, GWT, Hadoop
Developed and taught by well-known author and developer. At public venues or onsite at your location.

Controlling Image Loading

• **Use `prepareImage` to start loading image**

  ```
  prepareImage(image, window)
  prepareImage(image, width, height, window)
  ```

  - Starts loading image immediately (on separate thread), instead of when needed by `drawImage`
  - Particularly useful if the images will not be drawn until the user initiates some action such as clicking on a button or choosing a menu option
  - Since the applet thread immediately continues execution after the call to `prepareImage`, the image *may* not be completely loaded before `paint` is reached
Controlling Image Loading, Case I: No prepareImage

- Image is not loaded over network until after Display Image is pressed. 30.4 seconds.

Controlling Image Loading, Case 2: With prepareImage

- Image loaded over network immediately. 0.05 seconds after pressing button.
Controlling Image Loading: MediaTracker

- Registering images with a MediaTracker to control image loading

```java
MediaTracker tracker = new MediaTracker(this);
tracker.addImage(image1, 0);
tracker.addImage(image2, 1);
try {
    tracker.waitForAll();
} catch (InterruptedException ie) {}
if (tracker.isErrorAny()) {
    System.out.println("Error while loading image");
}
```

- Applet thread will block until all images are loaded
- Each image is loaded in parallel on a separate thread

Useful MediaTracker Methods

- `addImage`
  - Register a normal or scaled image with a given ID

- `checkAll, checkID`
  - Checks whether all or a particular registered image is done loading

- `isErrorAny, isErrorID`
  - Indicates if any or a particular image encountered an error while loading

- `waitForAll, waitForID`
  - Start loading all images or a particular image
  - Method does not return (blocks) until image is loaded

- See `TrackerUtil` in book for simplified usage of `MediaTracker`
**Loading Images, Case I: No MediaTracker**

- Image size is wrong, since the image won’t be done loading, and –1 will be returned

```java
@Override
public void init() {
    image = getImage(getDocumentBase(), imageName);
    imageWidth = image.getWidth(this);
    imageHeight = image.getHeight(this);
}
```

```java
@Override
public void paint(Graphics g) {
    g.drawImage(image, 0, 0, this);
    g.drawRect(0, 0, imageWidth, imageHeight);
}
```

**Loading Images, Case 2: With MediaTracker**

- Image is loaded before determining size

```java
@Override
public void init() {
    image = getImage(getDocumentBase(), imageName);
    MediaTracker tracker = new MediaTracker(this);
    tracker.addImage(image, 0);
    try {
        tracker.waitForAll();
    } catch(InterruptedException ie) {} 
    ...
    imageWidth = image.getWidth(this);
    imageHeight = image.getHeight(this);
}
```

```java
@Override
public void paint(Graphics g) {
    g.drawImage(image, 0, 0, this);
    g.drawRect(0, 0, imageWidth, imageHeight);
}
```
Wrap-Up
Summary: General Topics

- Use try/catch blocks to handle exceptions
  ```java
  try {
      ...
  } catch(Eclass1 var1) { ...
  } catch(Eclass2 var2) { ...
  } catch(Eclass3 var3) { ...
  }
  ```
  – More features of try/catch blocks in upcoming lecture (File IO)

- Always use @Override when replacing methods from parent class
  – Catches typos at compile time
    - If does not match, code will not compile
  – Expresses design intent
    - I.e., tells fellow developers that this is method from parent

Summary: Applet-Specific Topics

- Applet operations are restricted
  – Applet cannot read/write local files, call local programs, or connect to any host other than the one from which it was loaded

- The init method
  – Called only when applet loaded, not each time executed
  – This is where you use getParameter to read param data

- The paint method
  – Called each time applet is displayed
  – Coordinates in drawing operations are wrt top-left corner

- Drawing images
  – getImage(getCodeBase(), "imageFile") to “load” image
  – g.drawImage(image, x, y, this) to draw image
Questions?

More info:
http://courses.coreservlets.com/Course-Materials/java.html – General Java programming tutorial
http://www.coreservlets.com/java-8-tutorial – Java 8 tutorial
http://courses.coreservlets.com/java-training.html – Customized Java training courses at public venues or onsite at your organization
http://coreservlets.com/ – JSF 2, PrimeFaces, Java 7 or 8, Ajax, Query, Hadoop, RESTful Web Services, Android, HTML5, Spring, Hibernate, Services, JSP, GWT, and other Java EE training

Many additional free tutorials at coreservlets.com (JSF, Android, Ajax, Hadoop, and lots more)

Customized Java EE Training: http://courses.coreservlets.com/
Java 7, Java 8, JSF 2, PrimeFaces, Android, JSP, Ajax, jQuery, Spring MVC, RESTful Web Services, GWT, Hadoop
Developed and taught by well-known author and developer. At public venues or onsite at your location.