APPLET BASICS

- **Two Types of Applets**
- It is important to state at the outset that there are two varieties of applets.
- The first are those based directly on the **Applet** class . These applets use the Abstract Window Toolkit (AWT) to provide the graphic user interface (or use no GUI at all).
- This style of applet has been available since Java was first created.
- The second type of applets are those based on the Swing class **JApplet**.
- Swing applets use the Swing classes to provide the GUI.
- Swing offers a richer and often easier-to-use user interface than does the AWT. Thus, Swing-based applets are now the most popular.
- However, traditional AWT-based applets are still used, especially when only a very simple user interface is required.
- Thus, both AWT- and Swing-based applets are valid.

Applet Basics

All applets are subclasses (either directly or indirectly) of **Applet**. Applets are not stand-alone programs. Instead, they run within either a web browser or an applet viewer.

appletviewer, provided by the JDK. But you can use any applet viewer or browser you like.

Execution of an applet does not begin at **main( )**. Actually, few applets even have **main( )** methods. Instead, execution of an applet is started and controlled with an entirely different mechanism, which will be explained shortly. Output to your applet's window is not performed by **System.out.println( )**. Rather, in non-Swing applets, output is handled with various AWT methods, such as **drawString( )**, which outputs a string to a specified X,Y location. Input is also handled differently than in a console application.

To use an applet, it is specified in an HTMLfile. One way to do this is by using the APPLET tag. The applet will be executed by a Java-enabled web browser when it encounters the APPLET tag within the HTML file. To view and test an applet

more conveniently, simply include a comment at the head of your Java source code file that contains the APPLET tag. This way, your code is documented with the necessary HTML statements needed by your applet, and you can test the compiled applet by starting the applet

viewer with your Java source code file specified as the target. Here is an example of such a comment:

```
/*
<applet code="MyApplet" width=200 height=60>
</applet>*/
```

The Applet Class

The **Applet** class defines some methods. **Applet** provides all necessary support for applet execution, such as starting and stopping. It also provides methods that load and display images, and methods that load and play audio clips. **Applet** extends the AWT class **Panel**. In turn, **Panel** extends **Container**, which extends **Component**. These classes provide support for Java's window-based, graphical interface. Thus, **Applet** provides all of the necessary support for window-based activities

Method Description

void destroy( ) Called by the browser just before an applet is terminated. Your applet will override this method if it needs to perform any cleanup prior to its destruction.

AccessibleContext -getAccessibleContext( )- Returns the accessibility context for the invoking object.

AppletContext getAppletContext( )- Returns the context associated with the applet.

String getAppletInfo( ) Returns a string that describes the applet.

AudioClip getAudioClip(URL url) Returns an AudioClip object that encapsulates the audio clip found at the location specified by url.

Method Description

AudioClip getAudioClip(URL url,String clipName)-Returns an AudioClip object that encapsulates the audio clip found at the location specified by url and having the name specified by clipName.

URL getCodeBase( ) -Returns the URL associated with the invoking applet.

URL getDocumentBase( ) -Returns the URL of the HTML document that invokes the applet.

Image getImage(URL url) -Returns an Image object that encapsulates the image found at the location specified by url.

Image getImage(URL url,String imageName)-Returns an Image object that encapsulates the image found at the location specified by url and having the name specified by imageName.

Locale getLocale( )- Returns a Locale object that is used by various localesensitive classes and methods.

String getParameter(String paramName)- Returns the parameter associated with paramName .null is returned if the specified parameter is not found.

String[ ] [ ] getParameterInfo( )- Returns a String table that describes the parameters recognized by the applet. Each entry in the table must consist of three strings that contain the name of the parameter, a description of its type and/or range, and an explanation of its purpose.

void init( ) Called when an applet begins execution. It is the first method called for any applet.

boolean isActive( ) Returns true if the applet has been started. It returns false if the applet has been stopped.

static final AudioClip newAudioClip(URL url)- Returns an AudioClip object that encapsulates the audio clip found at the location specified by url. This method is similar to getAudioClip( ) except that it is static and can be executed without the need for an Applet object.

void play(URL url) If an audio clip is found at the location specified by

url, the clip is played.

void play(URL url, String clipName) If an audio clip is found at the location specified by url with the name specified by clipName, the clip is played.

void resize(Dimension dim) Resizes the applet according to the dimensions specified by dim. Dimension is a class stored inside java.awt. It contains two integer fields: width and height.

void resize(int width, int height) Resizes the applet according to the dimensions specified by width and height.

final void setStub(AppletStub stubObj) Makes stubObj the stub for the applet. This method is used by the run-time system and is not usually called by your applet. A stub is a small piece of code that provides the linkage between your applet and the browser.

void showStatus(String str) Displays str in the status window of the browser or applet viewer. If the browser does not support a status window, then no action takes place.

void start( ) Called by the browser when an applet should start (or resume) execution. It is automatically called after init( ) when an applet first begins.

void stop( ) Called by the browser to suspend execution of the applet. Once stopped, an applet is restarted when the browser calls start( ).

Applet Architecture

An applet is a window-based program. As such, its architecture is different from the console-based Programs. Applets are event driven. An applet resembles a set of interrupt service routines. The user initiates interaction with an applet—not the other way around.

An Applet Skeleton

All but the most trivial applets override a set of methods that provides the basic mechanism by which the browser or applet viewer interfaces to the applet and controls its execution. Four of these methods, **init( )**, **start( )**, **stop( )**, and **destroy( )**, apply to all applets and are defined by **Applet**. Default implementations for all of these methods are provided. Applets do not need to override those methods they do not use. However, only very simple applets will not need to define all of them.

AWT-based applets will also override the **paint( )** method, which is defined by the AWT **Component** class. This method is called when the applet's output must be redisplayed. (Swing-based applets use a different mechanism to accomplish this task.)

These five methods can be assembled into the skeleton shown here:

```
// An Applet skeleton.
import java.awt.*;
import java.applet.*;
/*
<applet code="AppletSkel" width=300 height=100>
</applet>
*/
public class AppletSkel extends Applet {
// Called first.
public void init() {
// initialization
}
/* Called second, after init(). Also called whenever
the applet is restarted. */
public void start() {
// start or resume execution
}
```

```
// Called when the applet is stopped.
public void stop() {
// suspends execution
}

/* Called when applet is terminated. This is the last
method executed. */
public void destroy() {
// perform shutdown activities
}
// Called when an applet's window must be restored.
public void paint(Graphics g) {
// redisplay contents of window
}
}
```

Applet Initialization and Termination

It is important to understand the order in which the various methods shown in the skeleton

are called. When an applet begins, the following methods are called, in this sequence:

1. **init( )**
2. **start( )**
3. **paint( )**

When an applet is terminated, the following sequence of method calls takes place:

1. **stop( )**
2. **destroy( )**

Let’s look more closely at these methods.

init( )

The **init( )** method is the first method to be called. This is where you should initialize variables.

This method is called only once during the run time of your applet.

start( )

The **start( )** method is called after **init( )**. It is also called to restart an applet after it has been

stopped. Whereas **init( )** is called once—the first time an applet is loaded—**start( )** is called each time an applet's HTML document is displayed onscreen. So, if a user leaves a web page and comes back, the applet resumes execution at **start( )**.

paint( )

The **paint( )** method is called each time your applet's output must be redrawn. This situation

can occur for several reasons. For example, the window in which the applet is running may

be overwritten by another window and then uncovered. Or the applet window may be

minimized and then restored. **paint( )** is also called when the applet begins execution.

Whatever the cause, whenever the applet must redraw its output, **paint( )** is called. The **paint( )**

method has one parameter of type **Graphics**. This parameter will contain the graphics context,

which describes the graphics environment in which the applet is running. This context is used

whenever output to the applet is required.

stop( )

The **stop( )** method is called when a web browser leaves the HTML document containing the

applet—when it goes to another page, for example. When **stop( )** is called, the applet is probably running. You should use **stop( )** to suspend threads that don't need to run when the

applet is not visible. You can restart them when **start( )** is called if the user returns to the page.

destroy( )

The **destroy( )** method is called when the environment determines that your applet needs to

be removed completely from memory. At this point, you should free up any resources the

applet may be using. The **stop( )** method is always called before **destroy( )**.

Overriding update( )

In some situations, your applet may need to override another method defined by the AWT,called **update( )**. This method is called when your applet has requested that a portion of its window be redrawn. The default version of **update( )** simply calls **paint( )**. However, you can override the **update( )** method so that it performs more subtle repainting. In general, overriding **update( )** is a specialized technique that is not applicable to all applets.

- **Simple Applet Display Methods**

As we've mentioned, applets are displayed in a window, and AWT-based applets use the AWT to perform input and output.

- To output a string to an applet, use **drawString( )**, which is a member of the **Graphics** class. Typically, it is called from within either **update( )** or **paint( )**.

- It has the following general form:
- **void drawString(String *message*, int *x*, int *y*)**

Here, *message* is the string to be output beginning at *x,y.* In a Java window, the upper-left corner is location 0,0. The **drawString( )** method will not recognize newline characters. If you want to start a line of text on another line, you must do so manually, specifying the precise X,Y location where you want the line to begin.

To set the background color of an applet's window, use **setBackground( )**. To set the foreground color (the color in which text is shown, for example), use **setForeground( )**.These methods are defined by **Component**, and they have the following general forms:

- void setBackground(Color *newColor*)
- void setForeground(Color *newColor*)
- Here, *newColor* specifies the new color. The class **Color** defines the constants shown here that can be used to specify colors:
- Color.black Color.magenta
- Color.blue Color.orange
- Color.cyan Color.pink
- Color.darkGray Color.red

- Color.gray Color.white
- Color.green Color.yellow
- Color.lightGray
- Uppercase versions of the constants are also defined.
- The following example sets the background color to green and the text color to red:
- setBackground(Color.green);
- setForeground(Color.red);
- A good place to set the foreground and background colors is in the **init( )** method. Ofcourse, you can change these colors as often as necessary during the execution of your applet.

- You can obtain the current settings for the background and foreground colors by calling
- **getBackground( )** and **getForeground( )**, respectively. They are also defined by **Component** and are shown here:
- Color getBackground( )
- Color getForeground( )

- Here is a very simple applet that sets the background color to cyan, the foreground color to red, and displays a message that illustrates the order in which the **init( )**, **start( )**, and **paint( )** methods are called when an applet starts up:

```
/* A simple applet that sets the foreground and
   background colors and outputs a string. */

import java.awt.*;

import java.applet.*;

/*<applet code="Sample" width=300
   height=50>

</applet>*/

public class Sample extends Applet{

String msg;
```

```
// set the foreground and background colors.

public void init() {

setBackground(Color.cyan);

setForeground(Color.red);

msg = "Inside init( ) --";

}

// Initialize the string to be displayed.

public void start() {

msg += " Inside start( ) --";
```

```
}
// Display msg in applet window.
public void paint(Graphics g) {
msg += " Inside paint( ).";
g.drawString(msg, 10, 30);
}
}
```