10.4. Applets in Java

IT1406 – Introduction to Programming

Level I - Semester 1

Applets in Java

- The **Applet** class is contained in the **java.applet** package.

- **Applet** contains several methods that give you detailed control over the execution of your applet. In addition, **java.applet** also defines three interfaces: **AppletContext**, **AudioClip**, and **AppletStub**.

Two types of applets

- It is important to state at the outset that there are two varieties of applets based on **Applet**.

- These applets use the Abstract Window Toolkit (AWT) to provide the graphical 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**, which inherits **Applet**. 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

- AWT-based applets are subclasses of **Applet**. Applets are not stand-alone programs. Instead, they run within either a web browser or an applet viewer.
- 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. Output to your applet's window is not performed by **System.out.println( )**.

- Rather, in an AWT-based applet, 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.

- Before an applet can be used, a deployment strategy must be chosen. There are two basic approaches. The first is to use the Java Network Launch Protocol (JNLP). This approach offers the most flexibility, especially as it relates to rich Internet applications. For real-world applets that you create, JNLP will often be the best choice.

- The second basic approach to deploying an applet is to specify the applet directly in an HTML file, without the use of JNLP. This is the original way that applets were launched when Java was created, and it is still used today—especially for simple applets.

- The use of the APPLET tag offers a secondary advantage when developing applets because it enables you to easily view and test the applet. To do so, 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>
*/
```

- This comment contains an APPLET tag that will run an applet called **MyApplet** in a window that is 200 pixels wide and 60 pixels high.

The applet class

- **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( )	Overrides of this method should return a string that describes the applet. The default implementation returns **null**.
AudioClip getAudioClip(URL *url*)	Returns an **AudioClip** object that encapsulates the audio clip found at the location specified by *url*.
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 locale-sensitive 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( )	Overrides of this method should return 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. The default implementation returns **null**.
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.
boolean isValidateRoot( )	Returns **true**, which indicates that an applet is a validate root.
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

- As a general rule, an applet is a GUI-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. Here is how the process works. An applet waits until an event occurs. The runtime system notifies the applet about an event by calling an event handler that has been provided by the applet.
- Once this happens, the applet must take appropriate action and then quickly return. This is a crucial point. For the most part, your applet should not enter a "mode" of operation in which it maintains control for an extended period.

- Instead, it must perform specific actions in response to events and then return control to the run-time system. In those situations in which your applet needs to perform a repetitive task on its own (for example, displaying a scrolling message across its window), you must start an additional thread of execution.
- Second, the user initiates interaction with an applet—not the other way around. As you know, in a console-based program, when the program needs input, it will prompt the user and then call some input method, such as **readLine( )**.
- This is not the way it works in an applet. Instead, the user interacts with the applet as he or she wants, when he or she wants. These interactions are sent to the applet as events to which the applet must respond.

- For example, when the user clicks the mouse inside the applet's window, a mouse-clicked event is generated. If the user presses a key while the applet's window has input focus, a keypress event is generated. Applets can contain various controls, such as push buttons and check boxes. When the user interacts with one of these controls, an event is generated.

- While the architecture of an applet is not as easy to understand as that of a console based program, Java makes it as simple as possible. If you have written programs for Windows (or other GUI-based operating systems), you know how intimidating that environment can be. Fortunately, Java provides a much cleaner approach that is more quickly mastered.

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 often 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
  }
}
```

- Although this skeleton does not do anything, it can be compiled and run. When run, it generates the following empty window when viewed with **appletviewer**.

The HTML applet tag

- Oracle recommends that the APPLET tag be used to manually start an applet when JNLP is not used.

- An applet viewer will execute each APPLET tag that it finds in a separate window, while web browsers will allow many applets on a single page. So far, we have been using only a simplified form of the APPLET tag.

- The syntax for a fuller form of the APPLET tag is shown here. Bracketed items are optional.

- **CODEBASE**
  - CODEBASE is an optional attribute that specifies the base URL of the applet code, which is the directory that will be searched for the applet's executable class file (specified by the CODE tag).
  - The HTML document's URL directory is used as the CODEBASE if this attribute is not specified.

- **CODE**
  - CODE is a required attribute that gives the name of the file containing your applet's compiled **.class** file.
  - This file is relative to the code base URL of the applet, which is the directory that the HTML file was in or the directory indicated by CODEBASE if set.

- **ALT**
  - The ALT tag is an optional attribute used to specify a short text message that should be displayed if the browser recognizes the APPLET tag but can't currently run Java applets.
  - This is distinct from the alternate HTML you provide for browsers that don't support applets.

- **NAME**
  - NAME is an optional attribute used to specify a name for the applet instance.
  - Applets must be named in order for other applets on the same page to find them by name and communicate with them.
  - To obtain an applet by name, use **getApplet( )**, which is defined by the **AppletContext** interface.

- **WIDTH and HEIGHT**
  - WIDTH and HEIGHT are required attributes that give the size (in pixels) of the applet display area.

- **ALIGN**
  - ALIGN is an optional attribute that specifies the alignment of the applet.
  - This attribute is treated the same as the HTML IMG tag with these possible values: LEFT, RIGHT, TOP, BOTTOM, MIDDLE, BASELINE, TEXTTOP, ABSMIDDLE, and ABSBOTTOM.

- **VSPACE and HSPACE**
  - These attributes are optional. VSPACE specifies the space, in pixels, above and below the applet.
  - HSPACE specifies the space, in pixels, on each side of the applet. They're treated the same as the IMG tag's VSPACE and HSPACE attributes.

- **PARAM NAME and VALUE**
  - The PARAM tag allows you to specify applet-specific arguments. Applets access their attributes with the **getParameter( )** method.

- Other valid APPLET attributes include ARCHIVE, which lets you specify one or more archive files, and OBJECT, which specifies a saved version of the applet. In general, an APPLET tag should include only a CODE or an OBJECT attribute, but not both.