

## Start a Web Browser and Request the Servlet

Start a web browser and enter the URL shown here:

```
http://localhost:8080/examples/servlets/servlet/HelloServlet
```

Alternatively, you may enter the URL shown here:

```
http://127.0.0.1:8080/examples/servlets/servlet/HelloServlet
```

This can be done because 127.0.0.1 is defined as the IP address of the local machine.

You will observe the output of the servlet in the browser display area. It will contain the string **Hello!** in bold type.

## The Servlet API

Two packages contain the classes and interfaces that are required to build the servlets described in this chapter. These are **javax.servlet** and **javax.servlet.http**. They constitute the core of the Servlet API. Keep in mind that these packages are not part of the Java core packages. Therefore, they are not included with Java SE. Instead, they are provided by Tomcat. They are also provided by Java EE.

The Servlet API has been in a process of ongoing development and enhancement. The current servlet specification is version 3.1. However, because changes happen fast in the world of Java, you will want to check for any additions or alterations. This chapter discusses the core of the Servlet API, which will be available to most readers and works with all modern versions of the servlet specification.

## The javax.servlet Package

The **javax.servlet** package contains a number of interfaces and classes that establish the framework in which servlets operate. The following table summarizes several key interfaces that are provided in this package. The most significant of these is **Servlet**. All servlets must implement this interface or extend a class that implements the interface. The **ServletRequest** and **ServletResponse** interfaces are also very important.

Interface	Description
Servlet	Declares life cycle methods for a servlet.
ServletConfig	Allows servlets to get initialization parameters.
ServletContext	Enables servlets to log events and access information about their environment.
ServletRequest	Used to read data from a client request.
ServletResponse	Used to write data to a client response.

The following table summarizes the core classes that are provided in the **javax.servlet** package:

Class	Description
GenericServlet	Implements the <b>Servlet</b> and <b>ServletConfig</b> interfaces.
ServletInputStream	Encapsulates an input stream for reading requests from a client.
ServletOutputStream	Encapsulates an output stream for writing responses to a client.
ServletException	Indicates a servlet error occurred.
UnavailableException	Indicates a servlet is unavailable.

Let us examine these interfaces and classes in more detail.

## The Servlet Interface

All servlets must implement the **Servlet** interface. It declares the **init()**, **service()**, and **destroy()** methods that are called by the server during the life cycle of a servlet. A method is also provided that allows a servlet to obtain any initialization parameters. The methods defined by **Servlet** are shown in Table 38-1.

The **init()**, **service()**, and **destroy()** methods are the life cycle methods of the servlet. These are invoked by the server. The **getServletConfig()** method is called by the servlet to obtain initialization parameters. A servlet developer overrides the **getServletInfo()** method to provide a string with useful information (for example, the version number). This method is also invoked by the server.

Method	Description
<code>void destroy()</code>	Called when the servlet is unloaded.
<code>ServletConfig getServletConfig()</code>	Returns a <b>ServletConfig</b> object that contains any initialization parameters.
<code>String getServletInfo()</code>	Returns a string describing the servlet.
<code>void init(ServletConfig <i>sc</i>)</code> throws <code>ServletException</code>	Called when the servlet is initialized. Initialization parameters for the servlet can be obtained from <i>sc</i> . A <b>ServletException</b> should be thrown if the servlet cannot be initialized.
<code>void service(ServletRequest <i>req</i>,               ServletResponse <i>res</i>)</code> throws <code>ServletException</code> , <code>IOException</code>	Called to process a request from a client. The request from the client can be read from <i>req</i> . The response to the client can be written to <i>res</i> . An exception is generated if a servlet or IO problem occurs.

**Table 38-1** The Methods Defined by **Servlet**

## The ServletConfig Interface

The **ServletConfig** interface allows a servlet to obtain configuration data when it is loaded. The methods declared by this interface are summarized here:

Method	Description
<code>ServletContext getServletContext( )</code>	Returns the context for this servlet.
<code>String getInitParameter(String <i>param</i>)</code>	Returns the value of the initialization parameter named <i>param</i> .
<code>Enumeration&lt;String&gt; getInitParameterNames( )</code>	Returns an enumeration of all initialization parameter names.
<code>String getServletName( )</code>	Returns the name of the invoking servlet.

## The ServletContext Interface

The **ServletContext** interface enables servlets to obtain information about their environment. Several of its methods are summarized in Table 38-2.

## The ServletRequest Interface

The **ServletRequest** interface enables a servlet to obtain information about a client request. Several of its methods are summarized in Table 38-3.

## The ServletResponse Interface

The **ServletResponse** interface enables a servlet to formulate a response for a client. Several of its methods are summarized in Table 38-4.

Method	Description
<code>Object getAttribute(String <i>attr</i>)</code>	Returns the value of the server attribute named <i>attr</i> .
<code>String getMimeType(String <i>file</i>)</code>	Returns the MIME type of <i>file</i> .
<code>String getRealPath(String <i>vpath</i>)</code>	Returns the real (i.e., absolute) path that corresponds to the relative path <i>vpath</i> .
<code>String getServerInfo( )</code>	Returns information about the server.
<code>void log(String <i>s</i>)</code>	Writes <i>s</i> to the servlet log.
<code>void log(String <i>s</i>, Throwable <i>e</i>)</code>	Writes <i>s</i> and the stack trace for <i>e</i> to the servlet log.
<code>void setAttribute(String <i>attr</i>, Object <i>val</i>)</code>	Sets the attribute specified by <i>attr</i> to the value passed in <i>val</i> .

**Table 38-2** Various Methods Defined by **ServletContext**

Method	Description
Object getAttribute(String <i>attr</i> )	Returns the value of the attribute named <i>attr</i> .
String getCharacterEncoding( )	Returns the character encoding of the request.
int getContentLength( )	Returns the size of the request. The value -1 is returned if the size is unavailable.
String getContentType( )	Returns the type of the request. A <b>null</b> value is returned if the type cannot be determined.
ServletInputStream getInputStream( ) throws IOException	Returns a <b>ServletInputStream</b> that can be used to read binary data from the request. An <b>IllegalStateException</b> is thrown if <b>getReader( )</b> has been previously invoked on this object.
String getParameter(String <i>pname</i> )	Returns the value of the parameter named <i>pname</i> .
Enumeration<String> getParameterNames( )	Returns an enumeration of the parameter names for this request.
String[ ] getParameterValues(String <i>name</i> )	Returns an array containing values associated with the parameter specified by <i>name</i> .
String getProtocol( )	Returns a description of the protocol.
BufferedReader getReader( ) throws IOException	Returns a buffered reader that can be used to read text from the request. An <b>IllegalStateException</b> is thrown if <b>getInputStream( )</b> has been previously invoked on this object.
String getRemoteAddr( )	Returns the string equivalent of the client IP address.
String getRemoteHost( )	Returns the string equivalent of the client host name.
String getScheme( )	Returns the transmission scheme of the URL used for the request (for example, "http", "ftp").
String getServerName( )	Returns the name of the server.
int getServerPort( )	Returns the port number.

**Table 38-3** Various Methods Defined by **ServletRequest**

Method	Description
String getCharacterEncoding( )	Returns the character encoding for the response.
ServletOutputStream getOutputStream( ) throws IOException	Returns a <b>ServletOutputStream</b> that can be used to write binary data to the response. An <b>IllegalStateException</b> is thrown if <b>getWriter( )</b> has been previously invoked on this object.
PrintWriter getWriter( ) throws IOException	Returns a <b>PrintWriter</b> that can be used to write character data to the response. An <b>IllegalStateException</b> is thrown if <b>getOutputStream( )</b> has been previously invoked on this object.
void setContentLength(int <i>size</i> )	Sets the content length for the response to <i>size</i> .
void setContentType(String <i>type</i> )	Sets the content type for the response to <i>type</i> .

**Table 38-4** Various Methods Defined by **ServletResponse**

## The GenericServlet Class

The **GenericServlet** class provides implementations of the basic life cycle methods for a servlet. **GenericServlet** implements the **Servlet** and **ServletConfig** interfaces. In addition, a method to append a string to the server log file is available. The signatures of this method are shown here:

```
void log(String s)
void log(String s, Throwable e)
```

Here, *s* is the string to be appended to the log, and *e* is an exception that occurred.

## The ServletInputStream Class

The **ServletInputStream** class extends **InputStream**. It is implemented by the servlet container and provides an input stream that a servlet developer can use to read the data from a client request. In addition to the input methods inherited from **InputStream**, a method is provided to read bytes from the stream. It is shown here:

```
int readLine(byte[] buffer, int offset, int size) throws IOException
```

Here, *buffer* is the array into which *size* bytes are placed starting at *offset*. The method returns the actual number of bytes read or  $-1$  if an end-of-stream condition is encountered.

## The ServletOutputStream Class

The **ServletOutputStream** class extends **OutputStream**. It is implemented by the servlet container and provides an output stream that a servlet developer can use to write data to a client response. In addition to the output methods provided by **OutputStream**, it also defines the **print()** and **println()** methods, which output data to the stream.

## The Servlet Exception Classes

**javax.servlet** defines two exceptions. The first is **ServletException**, which indicates that a servlet problem has occurred. The second is **UnavailableException**, which extends **ServletException**. It indicates that a servlet is unavailable.

## Reading Servlet Parameters

The **ServletRequest** interface includes methods that allow you to read the names and values of parameters that are included in a client request. We will develop a servlet that illustrates their use. The example contains two files. A web page is defined in **PostParameters.html**, and a servlet is defined in **PostParametersServlet.java**.

The HTML source code for **PostParameters.html** is shown in the following listing. It defines a table that contains two labels and two text fields. One of the labels is Employee and the other is Phone. There is also a submit button. Notice that the action parameter of the form tag specifies a URL. The URL identifies the servlet to process the HTTP POST request.

```
<html>
<body>
```

```

<center>
<form name="Form1"
    method="post"
    action="http://localhost:8080/examples/servlets/
        servlet/PostParametersServlet">
<table>
<tr>
    <td><B>Employee</td>
    <td><input type="text" name="e" size="25" value=""></td>
</tr>
<tr>
    <td><B>Phone</td>
    <td><input type="text" name="p" size="25" value=""></td>
</tr>
</table>
<input type="submit" value="Submit">
</body>
</html>

```

The source code for **PostParametersServlet.java** is shown in the following listing. The **service( )** method is overridden to process client requests. The **getParameterNames( )** method returns an enumeration of the parameter names. These are processed in a loop. You can see that the parameter name and value are output to the client. The parameter value is obtained via the **getParameter( )** method.

```

import java.io.*;
import java.util.*;
import javax.servlet.*;

public class PostParametersServlet
    extends GenericServlet {

    public void service(ServletRequest request,
        ServletResponse response)
        throws ServletException, IOException {

        // Get print writer.
        PrintWriter pw = response.getWriter();

        // Get enumeration of parameter names.
        Enumeration e = request.getParameterNames();

        // Display parameter names and values.
        while(e.hasMoreElements()) {
            String pname = (String)e.nextElement();
            pw.print(pname + " = ");
            String pvalue = request.getParameter(pname);
            pw.println(pvalue);
        }
        pw.close();
    }
}

```

Compile the servlet. Next, copy it to the appropriate directory, and update the **web.xml** file, as previously described. Then, perform these steps to test this example:

1. Start Tomcat (if it is not already running).
2. Display the web page in a browser.
3. Enter an employee name and phone number in the text fields.
4. Submit the web page.

After following these steps, the browser will display a response that is dynamically generated by the servlet.

## The javax.servlet.http Package

The preceding examples have used the classes and interfaces defined in **javax.servlet**, such as **ServletRequest**, **ServletResponse**, and **GenericServlet**, to illustrate the basic functionality of servlets. However, when working with HTTP, you will normally use the interfaces and classes in **javax.servlet.http**. As you will see, its functionality makes it easy to build servlets that work with HTTP requests and responses.

The following table summarizes the interfaces used in this chapter:

Interface	Description
HttpServletRequest	Enables servlets to read data from an HTTP request.
HttpServletResponse	Enables servlets to write data to an HTTP response.
HttpSession	Allows session data to be read and written.

The following table summarizes the classes used in this chapter. The most important of these is **HttpServlet**. Servlet developers typically extend this class in order to process HTTP requests.

Class	Description
Cookie	Allows state information to be stored on a client machine.
HttpServlet	Provides methods to handle HTTP requests and responses.

## The HttpServletRequest Interface

The **HttpServletRequest** interface enables a servlet to obtain information about a client request. Several of its methods are shown in Table 38-5.

## The HttpServletResponse Interface

The **HttpServletResponse** interface enables a servlet to formulate an HTTP response to a client. Several constants are defined. These correspond to the different status codes that can be assigned to an HTTP response. For example, **SC\_OK** indicates that the HTTP

Method	Description
<code>String getAuthType( )</code>	Returns authentication scheme.
<code>Cookie[ ] getCookies( )</code>	Returns an array of the cookies in this request.
<code>long getDateHeader(String field)</code>	Returns the value of the date header field named <i>field</i> .
<code>String getHeader(String field)</code>	Returns the value of the header field named <i>field</i> .
<code>Enumeration&lt;String&gt; getHeaderNames( )</code>	Returns an enumeration of the header names.
<code>int getIntHeader(String field)</code>	Returns the <b>int</b> equivalent of the header field named <i>field</i> .
<code>String getMethod( )</code>	Returns the HTTP method for this request.
<code>String getPathInfo( )</code>	Returns any path information that is located after the servlet path and before a query string of the URL.
<code>String getPathTranslated( )</code>	Returns any path information that is located after the servlet path and before a query string of the URL after translating it to a real path.
<code>String getQueryString( )</code>	Returns any query string in the URL.
<code>String getRemoteUser( )</code>	Returns the name of the user who issued this request.
<code>String getRequestedSessionId( )</code>	Returns the ID of the session.
<code>String getRequestURI( )</code>	Returns the URL.
<code>StringBuffer getRequestURL( )</code>	Returns the URL.
<code>String getServletPath( )</code>	Returns that part of the URL that identifies the servlet.
<code>HttpSession getSession( )</code>	Returns the session for this request. If a session does not exist, one is created and then returned.
<code>HttpSession getSession(boolean new)</code>	If <i>new</i> is <b>true</b> and no session exists, creates and returns a session for this request. Otherwise, returns the existing session for this request.
<code>boolean isRequestedSessionIdFromCookie( )</code>	Returns <b>true</b> if a cookie contains the session ID. Otherwise, returns <b>false</b> .
<code>boolean isRequestedSessionIdFromURL( )</code>	Returns <b>true</b> if the URL contains the session ID. Otherwise, returns <b>false</b> .
<code>boolean isRequestedSessionIdValid( )</code>	Returns <b>true</b> if the requested session ID is valid in the current session context.

**Table 38-5** Various Methods Defined by **HttpServletRequest**

request succeeded, and **SC\_NOT\_FOUND** indicates that the requested resource is not available. Several methods of this interface are summarized in Table 38-6.

## The HttpSession Interface

The **HttpSession** interface enables a servlet to read and write the state information that is associated with an HTTP session. Several of its methods are summarized in Table 38-7. All of these methods throw an **IllegalStateException** if the session has already been invalidated.



Method	Description
<code>void addCookie(Cookie cookie)</code>	Adds <i>cookie</i> to the HTTP response.
<code>boolean containsHeader(String field)</code>	Returns <b>true</b> if the HTTP response header contains a field named <i>field</i> .
<code>String encodeURL(String url)</code>	Determines if the session ID must be encoded in the URL identified as <i>url</i> . If so, returns the modified version of <i>url</i> . Otherwise, returns <i>url</i> . All URLs generated by a servlet should be processed by this method.
<code>String encodeRedirectURL(String url)</code>	Determines if the session ID must be encoded in the URL identified as <i>url</i> . If so, returns the modified version of <i>url</i> . Otherwise, returns <i>url</i> . All URLs passed to <b>sendRedirect()</b> should be processed by this method.
<code>void sendError(int c)</code> throws <code>IOException</code>	Sends the error code <i>c</i> to the client.
<code>void sendError(int c, String s)</code> throws <code>IOException</code>	Sends the error code <i>c</i> and message <i>s</i> to the client.
<code>void sendRedirect(String url)</code> throws <code>IOException</code>	Redirects the client to <i>url</i> .
<code>void setDateHeader(String field, long msec)</code>	Adds <i>field</i> to the header with date value equal to <i>msec</i> (milliseconds since midnight, January 1, 1970, GMT).
<code>void setHeader(String field, String value)</code>	Adds <i>field</i> to the header with value equal to <i>value</i> .
<code>void setIntHeader(String field, int value)</code>	Adds <i>field</i> to the header with value equal to <i>value</i> .
<code>void setStatus(int code)</code>	Sets the status code for this response to <i>code</i> .

Table 38-6 Various Methods Defined by `HttpServletResponse`

## The Cookie Class

The **Cookie** class encapsulates a cookie. A *cookie* is stored on a client and contains state information. Cookies are valuable for tracking user activities. For example, assume that a user visits an online store. A cookie can save the user's name, address, and other information. The user does not need to enter this data each time he or she visits the store.

A servlet can write a cookie to a user's machine via the **addCookie()** method of the **HttpServletResponse** interface. The data for that cookie is then included in the header of the HTTP response that is sent to the browser.

The names and values of cookies are stored on the user's machine. Some of the information that can be saved for each cookie includes the following:

- The name of the cookie
- The value of the cookie
- The expiration date of the cookie
- The domain and path of the cookie

Method	Description
Object <code>getAttribute(String attr)</code>	Returns the value associated with the name passed in <i>attr</i> . Returns <b>null</b> if <i>attr</i> is not found.
Enumeration<String> <code>getAttributeNames( )</code>	Returns an enumeration of the attribute names associated with the session.
long <code>getCreationTime( )</code>	Returns the creation time (in milliseconds since midnight, January 1, 1970, GMT) of the invoking session.
String <code>getId( )</code>	Returns the session ID.
long <code>getLastAccessedTime( )</code>	Returns the time (in milliseconds since midnight, January 1, 1970, GMT) when the client last made a request on the invoking session.
void <code>invalidate( )</code>	Invalidates this session and removes it from the context.
boolean <code>isNew( )</code>	Returns <b>true</b> if the server created the session and it has not yet been accessed by the client.
void <code>removeAttribute(String attr)</code>	Removes the attribute specified by <i>attr</i> from the session.
void <code>setAttribute(String attr, Object val)</code>	Associates the value passed in <i>val</i> with the attribute name passed in <i>attr</i> .

**Table 38-7** Various Methods Defined by **HttpSession**

The expiration date determines when this cookie is deleted from the user's machine. If an expiration date is not explicitly assigned to a cookie, it is deleted when the current browser session ends.

The domain and path of the cookie determine when it is included in the header of an HTTP request. If the user enters a URL whose domain and path match these values, the cookie is then supplied to the web server. Otherwise, it is not.

There is one constructor for **Cookie**. It has the signature shown here:

```
Cookie(String name, String value)
```

Here, the name and value of the cookie are supplied as arguments to the constructor. The methods of the **Cookie** class are summarized in Table 38-8.

## The HttpServlet Class

The **HttpServlet** class extends **GenericServlet**. It is commonly used when developing servlets that receive and process HTTP requests. The methods defined by the **HttpServlet** class are summarized in Table 38-9.

Method	Description
<code>Object clone( )</code>	Returns a copy of this object.
<code>String getComment( )</code>	Returns the comment.
<code>String getDomain( )</code>	Returns the domain.
<code>int getMaxAge( )</code>	Returns the maximum age (in seconds).
<code>String getName( )</code>	Returns the name.
<code>String getPath( )</code>	Returns the path.
<code>boolean getSecure( )</code>	Returns <b>true</b> if the cookie is secure. Otherwise, returns <b>false</b> .
<code>String getValue( )</code>	Returns the value.
<code>int getVersion( )</code>	Returns the version.
<code>boolean isHttpOnly( )</code>	Returns <b>true</b> if the cookie has the <b>HttpOnly</b> attribute.
<code>void setComment(String <i>c</i>)</code>	Sets the comment to <i>c</i> .
<code>void setDomain(String <i>d</i>)</code>	Sets the domain to <i>d</i> .
<code>void setHttpOnly(boolean <i>httpOnly</i>)</code>	If <i>httpOnly</i> is <b>true</b> , then the <b>HttpOnly</b> attribute is added to the cookie. If <i>httpOnly</i> is <b>false</b> , the <b>HttpOnly</b> attribute is removed.
<code>void setMaxAge(int <i>secs</i>)</code>	Sets the maximum age of the cookie to <i>secs</i> . This is the number of seconds after which the cookie is deleted.
<code>void setPath(String <i>p</i>)</code>	Sets the path to <i>p</i> .
<code>void setSecure(boolean <i>secure</i>)</code>	Sets the security flag to <i>secure</i> .
<code>void setValue(String <i>v</i>)</code>	Sets the value to <i>v</i> .
<code>void setVersion(int <i>v</i>)</code>	Sets the version to <i>v</i> .

**Table 38-8** The Methods Defined by **Cookie**

Method	Description
<code>void doDelete(HttpServletRequest <i>req</i>,                   HttpServletResponse <i>res</i>)       throws IOException, ServletException</code>	Handles an HTTP DELETE request.
<code>void doGet(HttpServletRequest <i>req</i>,             HttpServletResponse <i>res</i>)       throws IOException, ServletException</code>	Handles an HTTP GET request.
<code>void doHead(HttpServletRequest <i>req</i>,             HttpServletResponse <i>res</i>)       throws IOException,                       ServletException</code>	Handles an HTTP HEAD request.
<code>void doOptions(HttpServletRequest <i>req</i>,                 HttpServletResponse <i>res</i>)       throws IOException, ServletException</code>	Handles an HTTP OPTIONS request.

**Table 38-9** The Methods Defined by **HttpServlet**

Method	Description
void doPost(HttpServletRequest req, HttpServletResponse res) throws IOException, ServletException	Handles an HTTP POST request.
void doPut(HttpServletRequest req, HttpServletResponse res) throws IOException, ServletException	Handles an HTTP PUT request.
void doTrace(HttpServletRequest req, HttpServletResponse res) throws IOException, ServletException	Handles an HTTP TRACE request.
long getLastModified(HttpServletRequest req)	Returns the time (in milliseconds since midnight, January 1, 1970, GMT) when the requested resource was last modified.
void service(HttpServletRequest req, HttpServletResponse res) throws IOException, ServletException	Called by the server when an HTTP request arrives for this servlet. The arguments provide access to the HTTP request and response, respectively.

**Table 38-9** The Methods Defined by **HttpServlet** (continued)

## Handling HTTP Requests and Responses

The **HttpServlet** class provides specialized methods that handle the various types of HTTP requests. A servlet developer typically overrides one of these methods. These methods are **doDelete()**, **doGet()**, **doHead()**, **doOptions()**, **doPost()**, **doPut()**, and **doTrace()**. A complete description of the different types of HTTP requests is beyond the scope of this book. However, the GET and POST requests are commonly used when handling form input. Therefore, this section presents examples of these cases.

### Handling HTTP GET Requests

Here we will develop a servlet that handles an HTTP GET request. The servlet is invoked when a form on a web page is submitted. The example contains two files. A web page is defined in **ColorGet.html**, and a servlet is defined in **ColorGetServlet.java**. The HTML source code for **ColorGet.html** is shown in the following listing. It defines a form that contains a select element and a submit button. Notice that the action parameter of the form tag specifies a URL. The URL identifies a servlet to process the HTTP GET request.

```
<html>
<body>
<center>
<form name="Form1"
      action="http://localhost:8080/examples/servlets/servlet/ColorGetServlet">
<B>Color:</B>
<select name="color" size="1">
<option value="Red">Red</option>
<option value="Green">Green</option>
```

```

<option value="Blue">Blue</option>
</select>
<br><br>
<input type="submit" value="Submit">
</form>
</body>
</html>

```

The source code for **ColorGetServlet.java** is shown in the following listing. The **doGet()** method is overridden to process any HTTP GET requests that are sent to this servlet. It uses the **getParameter()** method of **HttpServletRequest** to obtain the selection that was made by the user. A response is then formulated.

```

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class ColorGetServlet extends HttpServlet {

    public void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {

        String color = request.getParameter("color");
        response.setContentType("text/html");
        PrintWriter pw = response.getWriter();
        pw.println("<B>The selected color is: ");
        pw.println(color);
        pw.close();
    }
}

```

Compile the servlet. Next, copy it to the appropriate directory, and update the **web.xml** file, as previously described. Then, perform these steps to test this example:

1. Start Tomcat, if it is not already running.
2. Display the web page in a browser.
3. Select a color.
4. Submit the web page.

After completing these steps, the browser will display the response that is dynamically generated by the servlet.

One other point: Parameters for an HTTP GET request are included as part of the URL that is sent to the web server. Assume that the user selects the red option and submits the form. The URL sent from the browser to the server is

```
http://localhost:8080/examples/servlets/servlet/ColorGetServlet?color=Red
```

The characters to the right of the question mark are known as the *query string*.

## Handling HTTP POST Requests

Here we will develop a servlet that handles an HTTP POST request. The servlet is invoked when a form on a web page is submitted. The example contains two files. A web page is defined in **ColorPost.html**, and a servlet is defined in **ColorPostServlet.java**.

The HTML source code for **ColorPost.html** is shown in the following listing. It is identical to **ColorGet.html** except that the method parameter for the form tag explicitly specifies that the POST method should be used, and the action parameter for the form tag specifies a different servlet.

```
<html>
<body>
<center>
<form name="Form1"
      method="post"
      action="http://localhost:8080/examples/servlets/servlet/ColorPostServlet">
<B>Color:</B>
<select name="color" size="1">
<option value="Red">Red</option>
<option value="Green">Green</option>
<option value="Blue">Blue</option>
</select>
<br><br>
<input type="submit" value="Submit">
</form>
</body>
</html>
```

The source code for **ColorPostServlet.java** is shown in the following listing. The **doPost()** method is overridden to process any HTTP POST requests that are sent to this servlet. It uses the **getParameter()** method of **HttpServletRequest** to obtain the selection that was made by the user. A response is then formulated.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class ColorPostServlet extends HttpServlet {

    public void doPost(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {

        String color = request.getParameter("color");
        response.setContentType("text/html");
        PrintWriter pw = response.getWriter();
        pw.println("<B>The selected color is: ");
        pw.println(color);
        pw.close();
    }
}
```

Compile the servlet and perform the same steps as described in the previous section to test it.

---

**NOTE** Parameters for an HTTP POST request are not included as part of the URL that is sent to the web server. In this example, the URL sent from the browser to the server is `http://localhost:8080/examples/servlets/servlet/ColorPostServlet`. The parameter names and values are sent in the body of the HTTP request.

## Using Cookies

Now, let's develop a servlet that illustrates how to use cookies. The servlet is invoked when a form on a web page is submitted. The example contains three files as summarized here:

File	Description
AddCookie.html	Allows a user to specify a value for the cookie named <b>MyCookie</b> .
AddCookieServlet.java	Processes the submission of <b>AddCookie.html</b> .
GetCookiesServlet.java	Displays cookie values.

The HTML source code for **AddCookie.html** is shown in the following listing. This page contains a text field in which a value can be entered. There is also a submit button on the page. When this button is pressed, the value in the text field is sent to **AddCookieServlet** via an HTTP POST request.

```
<html>
<body>
<center>
<form name="Form1"
    method="post"
    action="http://localhost:8080/examples/servlets/servlet/AddCookieServlet">
<B>Enter a value for MyCookie:</B>
<input type="text" name="data" size=25 value="">
<input type="submit" value="Submit">
</form>
</body>
</html>
```

The source code for **AddCookieServlet.java** is shown in the following listing. It gets the value of the parameter named "data". It then creates a **Cookie** object that has the name "MyCookie" and contains the value of the "data" parameter. The cookie is then added to the header of the HTTP response via the **addCookie()** method. A feedback message is then written to the browser.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class AddCookieServlet extends HttpServlet {

    public void doPost(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
```

```

// Get parameter from HTTP request.
String data = request.getParameter("data");

// Create cookie.
Cookie cookie = new Cookie("MyCookie", data);

// Add cookie to HTTP response.
response.addCookie(cookie);

// Write output to browser.
response.setContentType("text/html");
PrintWriter pw = response.getWriter();
pw.println("<B>MyCookie has been set to");
pw.println(data);
pw.close();
}
}

```

The source code for **GetCookiesServlet.java** is shown in the following listing. It invokes the **getCookies()** method to read any cookies that are included in the HTTP GET request. The names and values of these cookies are then written to the HTTP response. Observe that the **getName()** and **getValue()** methods are called to obtain this information.

```

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class GetCookiesServlet extends HttpServlet {

    public void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {

        // Get cookies from header of HTTP request.
        Cookie[] cookies = request.getCookies();

        // Display these cookies.
        response.setContentType("text/html");
        PrintWriter pw = response.getWriter();
        pw.println("<B>");
        for(int i = 0; i < cookies.length; i++) {
            String name = cookies[i].getName();
            String value = cookies[i].getValue();
            pw.println("name = " + name +
                "; value = " + value);
        }
        pw.close();
    }
}

```



Compile the servlets. Next, copy them to the appropriate directory, and update the **web.xml** file, as previously described. Then, perform these steps to test this example:

1. Start Tomcat, if it is not already running.
2. Display **AddCookie.html** in a browser.
3. Enter a value for **MyCookie**.
4. Submit the web page.

After completing these steps, you will observe that a feedback message is displayed by the browser.

Next, request the following URL via the browser:

```
http://localhost:8080/examples/servlets/servlet/GetCookiesServlet
```

Observe that the name and value of the cookie are displayed in the browser.

In this example, an expiration date is not explicitly assigned to the cookie via the **setMaxAge()** method of **Cookie**. Therefore, the cookie expires when the browser session ends. You can experiment by using **setMaxAge()** and observe that the cookie is then saved on the client machine.

## Session Tracking

HTTP is a stateless protocol. Each request is independent of the previous one. However, in some applications, it is necessary to save state information so that information can be collected from several interactions between a browser and a server. Sessions provide such a mechanism.

A session can be created via the **getSession()** method of **HttpServletRequest**. An **HttpSession** object is returned. This object can store a set of bindings that associate names with objects. The **setAttribute()**, **getAttribute()**, **getAttributeNames()**, and **removeAttribute()** methods of **HttpSession** manage these bindings. Session state is shared by all servlets that are associated with a client.

The following servlet illustrates how to use session state. The **getSession()** method gets the current session. A new session is created if one does not already exist. The **getAttribute()** method is called to obtain the object that is bound to the name "date". That object is a **Date** object that encapsulates the date and time when this page was last accessed. (Of course, there is no such binding when the page is first accessed.) A **Date** object encapsulating the current date and time is then created. The **setAttribute()** method is called to bind the name "date" to this object.

```
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class DateServlet extends HttpServlet {
```

```

public void doGet(HttpServletRequest request,
    HttpServletResponse response)
    throws ServletException, IOException {

    // Get the HttpSession object.
    HttpSession hs = request.getSession(true);

    // Get writer.
    response.setContentType("text/html");
    PrintWriter pw = response.getWriter();
    pw.print("<B>");

    // Display date/time of last access.
    Date date = (Date)hs.getAttribute("date");
    if(date != null) {
        pw.print("Last access: " + date + "<br>");
    }

    // Display current date/time.
    date = new Date();
    hs.setAttribute("date", date);
    pw.println("Current date: " + date);
}
}

```

When you first request this servlet, the browser displays one line with the current date and time information. On subsequent invocations, two lines are displayed. The first line shows the date and time when the servlet was last accessed. The second line shows the current date and time.

This page has been intentionally left blank

## APPENDIX

# Using Java's Documentation Comments

As explained in Part I, Java supports three types of comments. The first two are the `//` and the `/* */`. The third type is called a *documentation comment*. It begins with the character sequence `/**`. It ends with `*/`. Documentation comments allow you to embed information about your program into the program itself. You can then use the **javadoc** utility program (supplied with the JDK) to extract the information and put it into an HTML file.

Documentation comments make it convenient to document your programs. You have almost certainly seen documentation generated with **javadoc**, because that is the way the Java API library was documented.

## The javadoc Tags

The **javadoc** utility recognizes the following tags:

Tag	Meaning
@author	Identifies the author.
{@code}	Displays information as-is, without processing HTML styles, in code font.
@deprecated	Specifies that a program element is deprecated.
{@docRoot}	Specifies the path to the root directory of the current documentation.
@exception	Identifies an exception thrown by a method or constructor.
{@inheritDoc}	Inherits a comment from the immediate superclass.
{@link}	Inserts an in-line link to another topic.
{@linkplain}	Inserts an in-line link to another topic, but the link is displayed in a plain-text font.
{@literal}	Displays information as-is, without processing HTML styles.
@param	Documents a parameter.
@return	Documents a method's return value.
@see	Specifies a link to another topic.

Tag	Meaning
<code>@serial</code>	Documents a default serializable field.
<code>@serialData</code>	Documents the data written by the <code>writeObject( )</code> or <code>writeExternal( )</code> methods.
<code>@serialField</code>	Documents an <code>ObjectStreamField</code> component.
<code>@since</code>	States the release when a specific change was introduced.
<code>@throws</code>	Same as <code>@exception</code> .
<code>{@value}</code>	Displays the value of a constant, which must be a <code>static</code> field.
<code>@version</code>	Specifies the version of a class.

Document tags that begin with an “at” sign (`@`) are called *stand-alone* tags (also called *block* tags), and they must be used on their own line. Tags that begin with a brace, such as `{@code}`, are called *in-line* tags, and they can be used within a larger description. You may also use other, standard HTML tags in a documentation comment. However, some tags, such as headings, should not be used because they disrupt the look of the HTML file produced by **javadoc**.

As it relates to documenting source code, you can use documentation comments to document classes, interfaces, fields, constructors, and methods. In all cases, the documentation comment must immediately precede the item being documented. Some tags, such as `@see`, `@since`, and `@deprecated`, can be used to document any element. Other tags apply only to the relevant elements. Each tag is examined next.

---

**NOTE** Documentation comments can also be used for documenting a package and preparing an overview, but the procedures differ from those used to document source code. See the **javadoc** documentation for details on these uses.

## @author

The `@author` tag documents the author of a class or interface. It has the following syntax:

```
@author description
```

Here, *description* will usually be the name of the author. You will need to specify the `-author` option when executing **javadoc** in order for the `@author` field to be included in the HTML documentation.

## {@code}

The `{@code}` tag enables you to embed text, such as a snippet of code, into a comment. That text is then displayed as-is in code font, without any further processing, such as HTML rendering. It has the following syntax:

```
{@code code-snippet}
```

## @deprecated

The `@deprecated` tag specifies that a program element is deprecated. It is recommended that you include `@see` or `{@link}` tags to inform the programmer about available alternatives. The syntax is the following:

```
@deprecated description
```

Here, *description* is the message that describes the deprecation. The **@deprecated** tag can be used in documentation for fields, methods, constructors, classes, and interfaces.

## **{@docRoot}**

**{@docRoot}** specifies the path to the root directory of the current documentation.

## **@exception**

The **@exception** tag describes an exception to a method. It has the following syntax:

```
@exception exception-name explanation
```

Here, the fully qualified name of the exception is specified by *exception-name*, and *explanation* is a string that describes how the exception can occur. The **@exception** tag can only be used in documentation for a method or constructor.

## **{@inheritDoc}**

This tag inherits a comment from the immediate superclass.

## **{@link}**

The **{@link}** tag provides an in-line link to additional information. It has the following syntax:

```
{@link pkg.class#member text}
```

Here, *pkg.class#member* specifies the name of a class or method to which a link is added, and *text* is the string that is displayed.

## **{@linkplain}**

Inserts an in-line link to another topic. The link is displayed in plain-text font. Otherwise, it is similar to **{@link}**.

## **{@literal}**

The **{@literal}** tag enables you to embed text into a comment. That text is then displayed as-is, without any further processing, such as HTML rendering. It has the following syntax:

```
{@literal description}
```

Here, *description* is the text that is embedded.

## **@param**

The **@param** tag documents a parameter. It has the following syntax:

```
@param parameter-name explanation
```

Here, *parameter-name* specifies the name of a parameter. The meaning of that parameter is described by *explanation*. The **@param** tag can be used only in documentation for a method or constructor, or a generic class or interface.

## @return

The **@return** tag describes the return value of a method. It has the following syntax:

```
@return explanation
```

Here, *explanation* describes the type and meaning of the value returned by a method. The **@return** tag can be used only in documentation for a method.

## @see

The **@see** tag provides a reference to additional information. Two commonly used forms are shown here:

```
@see anchor
```

```
@see pkg.class#member text
```

In the first form, *anchor* is a link to an absolute or relative URL. In the second form, *pkg.class#member* specifies the name of the item, and *text* is the text displayed for that item. The text parameter is optional, and if not used, then the item specified by *pkg.class#member* is displayed. The member name, too, is optional. Thus, you can specify a reference to a package, class, or interface in addition to a reference to a specific method or field. The name can be fully qualified or partially qualified. However, the dot that precedes the member name (if it exists) must be replaced by a hash character.

## @serial

The **@serial** tag defines the comment for a default serializable field. It has the following syntax:

```
@serial description
```

Here, *description* is the comment for that field.

## @serialData

The **@serialData** tag documents the data written by the **writeObject()** and **writeExternal()** methods. It has the following syntax:

```
@serialData description
```

Here, *description* is the comment for that data.

## @serialField

For a class that implements **Serializable**, the **@serialField** tag provides comments for an **ObjectStreamField** component. It has the following syntax:

```
@serialField name type description
```

Here, *name* is the name of the field, *type* is its type, and *description* is the comment for that field.

## @since

The **@since** tag states that an element was introduced in a specific release. It has the following syntax:

```
@since release
```

Here, *release* is a string that designates the release or version in which this feature became available.

## @throws

The **@throws** tag has the same meaning as the **@exception** tag.

## {@value}

**{@value}** has two forms. The first displays the value of the constant that it precedes, which must be a **static** field. It has this form:

```
{@value}
```

The second form displays the value of a specified **static** field. It has this form:

```
{@value pkg.class#field}
```

Here, *pkg.class#field* specifies the name of the **static** field.

## @version

The **@version** tag specifies the version of a class or interface. It has the following syntax:

```
@version info
```

Here, *info* is a string that contains version information, typically a version number, such as 2.2. You will need to specify the **-version** option when executing **javadoc** in order for the **@version** field to be included in the HTML documentation.

# The General Form of a Documentation Comment

After the beginning **/\*\***, the first line or lines become the main description of your class, interface, field, constructor, or method. After that, you can include one or more of the various **@** tags. Each **@** tag must start at the beginning of a new line or follow one or more asterisks (\*) that are at the start of a line. Multiple tags of the same type should be grouped together. For example, if you have three **@see** tags, put them one after the other. In-line tags (those that begin with a brace) can be used within any description.

Here is an example of a documentation comment for a class:

```
/**
 * This class draws a bar chart.
 * @author Herbert Schildt
 * @version 3.2
 */
```

## What javadoc Outputs

The **javadoc** program takes as input your Java program's source file and outputs several HTML files that contain the program's documentation. Information about each class will be in its own HTML file. **javadoc** will also output an index and a hierarchy tree. Other HTML files can be generated.



## An Example that Uses Documentation Comments

Following is a sample program that uses documentation comments. Notice the way each comment immediately precedes the item that it describes. After being processed by **javadoc**, the documentation about the **SquareNum** class will be found in **SquareNum.html**.

```
import java.io.*;

/**
 * This class demonstrates documentation comments.
 * @author Herbert Schildt
 * @version 1.2
 */
public class SquareNum {
    /**
     * This method returns the square of num.
     * This is a multiline description. You can use
     * as many lines as you like.
     * @param num The value to be squared.
     * @return num squared.
     */
    public double square(double num) {
        return num * num;
    }

    /**
     * This method inputs a number from the user.
     * @return The value input as a double.
     * @exception IOException On input error.
     * @see IOException
     */
    public double getNumber() throws IOException {
        // create a BufferedReader using System.in
        InputStreamReader isr = new InputStreamReader(System.in);
        BufferedReader inData = new BufferedReader(isr);
        String str;

        str = inData.readLine();
        return (new Double(str)).doubleValue();
    }

    /**
     * This method demonstrates square().
     * @param args Unused.
     * @exception IOException On input error.
     * @see IOException
     */

    public static void main(String args[])
        throws IOException
    {
        SquareNum ob = new SquareNum();
        double val;
```

```
System.out.println("Enter value to be squared: ");
val = ob.getNumber();
val = ob.square(val);

System.out.println("Squared value is " + val);
}
}
```

This page has been intentionally left blank

# Index

- &
    - bitwise AND, 66, 67, 68–69
    - Boolean logical AND, 75–76
    - and bounded type declarations, 349
  - && (short-circuit AND), 75, 76–77
  - \*
    - and glob syntax, 715–716
    - multiplication operator, 27, 61–62
    - regular expression quantifier, 995
    - used in import statement, 194, 333
  - \*\* (glob syntax), 716
  - @
    - annotation syntax, 280
    - used with tags (javadoc), 1236, 1239
  - |
    - bitwise OR, 66, 67, 68–69
    - Boolean logical OR, 75–76
  - || (short-circuit OR), 75, 76–77
  - [ ], 33, 51, 52, 54, 58
    - character class specification, 995, 999
  - ^
    - bitwise exclusive OR (XOR), 66, 67, 68–69
    - Boolean logical exclusive OR (XOR), 75–76
  - : (used with a label), 105
  - ::
    - constructor reference, 33, 404, 408
    - method reference, 33, 396, 402
  - , (comma), 33, 95, 387
    - format flag, 615, 617
  - { }, 24, 25, 26, 30, 33, 45, 46, 53, 56, 81, 82, 89, 93, 217, 291, 388
    - used with javadoc tags, 1236
  - =, 27, 44, 74, 77
  - == (Boolean logical operator), 75
  - == (relational operator), 28, 74, 264, 269
    - versus equals( ), 422–423
  - !=, 75–76
  - !=, 74, 75
  - /, 61–62
  - /\* \*/, 24
  - /\*\* \*/, 33, 1235
  - //, 25
  - <, 28, 74
    - argument index syntax, 618–619
  - <>
    - diamond operator (type inference), 372–373
    - and generic type parameter, 340
  - <?>, 282, 283
  - <<, 66, 69–70
  - <=, 74
  - , 61–62
    - format flag, 615
  - > lambda expression arrow operator, 16, 61, 382
  - –, 30, 61, 64–65
  - %
    - used in format conversion specifier syntax, 607
    - modulus operator, 61, 63
  - ( format flag, 615, 617
  - ( ), 25, 33, 79, 114, 123
    - used in a lambda expression, 382, 383, 386
    - used to raise the precedence of operations, 33, 41, 79, 418
  - .
    - dot operator, 111, 117–118, 146, 170, 188, 194, 211
    - in import statement, 194
    - in multileveled package statement, 188
    - and nested interfaces, 200–201
    - regular expression wildcard character, 995, 998
    - separator, 33
  - ... (variable-length argument syntax), 156, 159
  - +
    - addition operator, 61–62
    - concatenation operator, 27, 152–153, 417–418
    - format flag, 615, 616
    - regular expression quantifier, 995, 997–999
    - unary plus, 61, 62
  - ++, 30, 61, 64–66
  - # format flag, 615, 617
  - ?
    - regular expression quantifier, 995, 998–999
    - wildcard argument specifier, 350, 353, 356, 370, 379
  - ?: (ternary if-then-else operator), 75, 77–78
  - >, 28, 74
  - >>, 66, 70–72
  - >>>, 66, 72–73
  - >=, 74
  - ; (semicolon), 26, 33, 90, 197
    - used in try-with-resources statement, 317, 650
  - ~ (bitwise unary NOT operator), 66, 67, 68–69
  - \_ (underscore), 32, 42, 43
- 
- ## A
- abs( ), 131–132, 478
  - Abstract method(s), 182–184
    - and lambda expressions, 382, 383, 384, 385
  - abstract type modifier, 182, 185, 199–200, 383

- Abstract Window Toolkit. *See* AWT
- (Abstract Window Toolkit)
- AbstractAction interface, 1091, 1092
- AbstractButton class, 1045, 1047, 1048, 1070, 1073, 1078, 1081
- AbstractCollection class, 511, 513, 520
- AbstractList class, 511, 562
- AbstractMap class, 537, 538, 539, 541
- AbstractQueue class, 511, 519
- AbstractSequentialList class, 511, 515
- AbstractSet class, 511, 516, 518, 521
- accept( ), 526, 636, 646, 648, 716, 742, 972, 985, 987
- Access control, 141–144
  - and default access, 190, 197
  - example program, 191–194
  - and inheritance, 142, 144, 163–164
  - and packages, 142, 187, 190–194
- Access modifiers, 25, 142, 190–191
- acos( ), 477
- acquire( ), 918–921
- Action (Swing), 1069, 1089–1094
- Action interface, 1089
- ActionEvent class, 772, 773, 836, 837, 847, 872, 1032, 1043, 1045, 1052, 1179
  - JavaFX, 1115, 1116, 1118
- ActionListener interface, 782, 783, 836, 839, 847, 872, 1032, 1045, 1051, 1074, 1089
- actionPerformed( ), 783, 836, 837, 839, 1032, 1033, 1045, 1051, 1052, 1074, 1078, 1089, 1091–1092
- adapt( ), 962
- Adapter classes, 791–793
- add( ), 502, 503, 504, 505, 516, 523, 588, 801, 834, 839, 844, 846, 858, 863, 871, 985, 1006, 1007, 1029, 1051, 1064, 1071, 1072, 1087, 1091, 1113, 1160, 1164, 1166, 1173, 1174, 1189
- addActionListener( ), 1032
- addAll( ), 502, 503, 504, 505, 551, 985, 1114, 1118, 1160, 1173, 1174, 1179
- addCookie( ), 1224, 1230
- addElement( ), 568
- addEventFilter( ), 1115
- addExact( ), 480
- addFirst( ), 509, 510, 515, 985
- addImage( ), 892, 893
- addItem( ), 1061
- addKeyListener( ), 770
- addLast( ), 509, 510, 515, 516
- addListener( ), 1139, 1160
- addMouseListener( ), 788, 793–794, 1084
- addMouseMotionListener( ), 770, 788
- Address, Internet, 728, 729–731
- addSeparator( ), 1072
- addSuppressed( ), 228
- addTab( ), 1053, 1054
- addTListener( ) 1202
- addTypeListener( ), 770, 771
- AdjustmentEvent class, 772, 773–774, 850
- AdjustmentListener interface, 782, 783, 850
- adjustmentValueChanged( ), 783
- Affine class, 1166
- Algorithms, collection, 499, 550–556, 561
- ALIGN, 760, 761
- allMatch( ), 990
- allocate( ), 691, 701, 702, 720–721, 723
- ALT, 760, 761
- anchor constraint field, 866, 867–868
- AND operator
  - bitwise (&), 66, 67, 68–69
  - Boolean logical (&), 75–76
  - and bounded type declarations (&), 349
  - short-circuit (&&), 75, 76–77
- AnnotatedElement interface, 286–287, 288, 298, 496
- Annotation interface, 280, 286, 496
- Annotation(s), 13, 14, 279–299, 496
  - built-in, 290–292
  - container, 297, 298
  - declaration example, 280
  - marker, 288–289
  - member, default value for, 287–288
  - obtaining all, 285–286
  - reflection to obtain, using, 281–286
  - repeated, 287, 297–299
  - restrictions on, 299
  - retention policies, 281
  - single-member, 289–290
  - type, 292–297
- annotationType( ), 280
- anyMatch( ), 990
- Apache Software Foundation, 1212
- API library, compact profiles of the, 336
- API packages, table of core Java, 991–993
- append( ), 435, 494, 671, 854
- Appendable interface, 494, 608, 665, 670, 679
- appendCodePoint( ), 438
- appendTo( ), 465
- Applet, 8, 16
- Applet, AWT-based, 318–321, 747–767
  - architecture, 751, 756
  - basics, 747–750
  - colors, setting and obtaining, 754–755
  - event-driven nature of the, 751, 769
  - executing an, 320–321, 747–748, 751, 760
  - and the Internet, 8–9, 16, 318–319, 320, 748–749
  - local, 748
  - and main( ), 26, 110, 320, 321, 748
  - outputting to console, 767
  - passing parameters to an, 761–764
  - request for repainting, 756–759
  - and security, 748–749
  - signed, 320, 748–749
  - skeleton, 751–754
  - and socket connections, 731
  - as source and listener for events, 788
  - string output to an, 319, 748, 754, 756
  - and uncaught exceptions, 762
  - viewer, 320–321, 747, 748, 751, 759, 760, 767, 798, 801
- Applet class, 319, 747–765, 781, 788, 792, 793, 801, 886, 887, 1033, 1035
  - methods, table of, 749–750
- applet package, 301, 319, 747
- Applet, Swing, 747, 752, 754, 1025, 1030, 1033–1035
- APPLET tag, HTML, 320, 321, 748
  - full syntax for, 760–761
- AppletContext interface, 747, 761, 765–766
  - methods, table of, 765–766

- AppletStub interface, 747, 767
  - appletviewer, 320, 747, 749, 752, 1034
    - status window, using, 759–760
  - Application class, 1107–1108, 1110
  - Application launcher (java), 24, 188, 189
    - and main(), 25
  - apply(), 636, 637, 973, 978
  - applyAsDouble(), 636, 981
  - ARCHIVE, 761
  - AreaAveragingScaleFilter class, 899
  - areFieldsSet, 588
  - Argument(s), 116, 120
    - command-line, 25, 154–155
    - index, 618–619
    - passing, 136–138
    - type. *See* Type argument(s)
    - variable-length. *See* Varargs
    - wildcard. *See* Wildcard arguments
  - Arithmetic operators, 61–66
  - ArithmeticException, 215, 216, 226, 479
  - Array class, 496
  - Array(s), 25, 51–58, 147, 185
    - boundary checks, 53
    - and collections, 556
    - constructor reference for, 408
    - converting collections into, 503, 513–514
    - copying with arraycopy(), 467, 469–470
    - declaration syntax, alternative, 58
    - dynamic, 496, 511–514, 520, 562
    - and the for-each loop, 97–101
    - and generics, 377–379
    - implemented as objects, 147
    - indexes, 52
    - initializing, 53, 56–57
    - length instance variable of, 147–149
    - multidimensional, 54–58
    - one-dimensional, 51–54
    - and spliterators, 559
    - and the stream API, 969
    - of strings, 154
    - and valueOf(), 429
    - and varargs, 156
  - ArrayBlockingQueue class, 943
  - arraycopy(), 467, 469–470
  - ArrayDeque class, 511, 520–521, 568
  - ArrayIndexOutOfBoundsException, 219, 226, 557
  - ArrayList class, 511–514, 530, 562, 563, 969
    - example using an, 524–525
    - example using a stream API stream, 969–973, 978–982, 983–984, 986–989
  - Arrays class, 556–561, 969
  - ArrayStoreException, 226, 557, 558
  - arrive(), 930–931
  - arriveAndAwaitAdvance(), 930, 931, 933, 936
  - arriveAndDeregister(), 931, 933
  - Arrow operator (→), 16, 61, 382
  - ASCII character set, 39, 40, 43, 424
    - and strings on the Internet, 415, 420
  - asin(), 477
  - asList(), 556
  - Assembly language, 4, 5
  - assert statement, 13, 328–331
  - Assertions, 328–331
  - AssertionError, 328, 329
  - Assignment operator
    - =, 27, 74, 77
    - arithmetic compound (*op*=), 61, 63–64
    - bitwise compound, 66, 73–74
    - Boolean logical, 75
  - atan(), 477
  - atan2(), 477
  - Atomic operations, 946–947
  - AtomicInteger class, 917, 946–947
  - AtomicLong class, 917, 946
  - AttributeView interface, 699
  - AudioClip interface, 747, 767
  - Autoboxing/unboxing, 14, 272, 274–279, 341–342
    - Boolean and Character values, 278
    - and the Collections Framework, 500, 514
    - definition of, 274
    - and error prevention, 278–279
    - and expressions, 276–277
    - and methods, 275–276
    - when to use, 279
  - AutoCloseable interface, 310, 316, 495, 619, 626, 648, 650, 651, 654, 665, 667, 668, 669, 670, 679, 683, 685, 691, 701, 714, 732, 743, 966
  - Automatic resource management (ARM), 214, 315–318, 495, 619, 734
  - available(), 651, 652–654, 685, 686
  - availableProcessors(), 958
  - await(), 923–925, 927, 944
  - awaitAdvance(), 936
  - awaitAdvanceInterruptibly(), 936
  - AWT (Abstract Window Toolkit), 301, 319, 747, 748, 797–798, 833, 1105
    - and applet architectural constraints, 756
    - classes, table of some, 798–800
    - color system, 815
    - controls. *See* Controls, AWT
    - creating stand-alone windows with, 809–810
    - and fonts, 819–825
    - layout managers. *See* Layout manager(s)
    - support for imaging, 885
    - support for text and graphics, 811
    - and Swing, 797, 1021–1022
  - AWTEvent class, 772, 798
- ## B
- 
- B, 4
  - Base64 class, 635
  - BaseStream interface, 966–968, 975
    - methods, table of, 966
  - BASIC, 4
  - Basic multilingual plane (BMP), 458
  - BasicFileAttributes class, 698–699, 712
    - methods, table of, 698
  - BasicFileAttributeView interface, 699
  - BCP 47, 595
  - BCPL, 4
  - BeanInfo interface, 1200, 1202–1203, 1204
  - Beans, Java. *See* Java Beans
  - Bell curve, 596
  - Bell Laboratories, 6
  - Berkeley UNIX, 727

Berners-Lee, Tim, 735  
*Beyond Photography, The Digital Darkroom* (Holzmann), 895  
 BiConsumer functional interface, 636, 985  
 BiFunction functional interface, 636, 973  
 Binary  
   literals, 42  
   numbers and integers, 66–67  
 BinaryOperator<T> predefined functional interface, 409, 636, 973  
 binarySearch( ), 551, 556–557  
 bitCount( ), 450, 452  
 BitSet class, 581–583  
   methods, table of, 581–582  
 Bitwise operators, 66–74  
 Block lambdas, 382, 388–389. *See also* Lambda expression(s)  
 BLOCKED, 260  
 Blocks of code. *See* Code blocks  
 Boolean, 35  
   literals, 43  
   logical operators, 75–77  
 Boolean class, 272, 273, 458–460  
   and autoboxing/unboxing, 278  
   methods, table of, 460  
 boolean data type, 35, 40–41, 43, 48  
   and relational operators, 40, 41, 74–75  
 booleanValue( ), 273, 460  
 Border interface, 1040  
 BorderFactory class, 1040  
 BorderLayout class, 798, 858–859, 1032  
   example with insets, 860–861  
 BorderPane class, 1107, 1178, 1187  
   methods for positioning nodes within a, 1178  
 boxed( ), 968  
 Boxing, 274  
 break statement, 84–86, 98–99, 102–106  
   and the for-each loop, 98–99  
   as form of goto, 104–106  
 Buffer class, 690–691  
   methods, table of, 690–691  
 Buffer, NIO, 690–691  
 BufferedInputStream class, 303, 659–661, 711  
 BufferedOutputStream class, 303, 659, 661–662, 711  
 BufferedReader class, 304, 305, 306–307, 676–677, 969  
 BufferedWriter class, 304, 678  
 Buffering, double, 889–892  
 bulkRegister( ), 936  
 Button class  
   AWT, 798, 836  
   JavaFX, 1115, 1130, 1133  
 ButtonBase class, 1115, 1133, 1135  
 ButtonGroup class, 1041, 1051  
 ButtonModel interface, 1024, 1045  
 Buttons, 773, 782  
   JavaFX, 1115–1116, 1130–1142  
   push. *See* Push buttons  
   radio. *See* Radio buttons  
   Swing, 1032–1033, 1045–1053, 1070  
   toggle. *See* Toggle button, JavaFX;  
   Toggle button, Swing  
 ButtonUI, 1024  
 Byte class, 273, 447, 454  
   methods defined by, table of, 448

byte data type, 35, 36–37, 41  
   and automatic type conversion, 48  
   and automatic type promotion, 50, 69–70, 72–73  
 ByteArrayInputStream class, 303, 656–657  
 ByteArrayOutputStream class, 303, 658–659  
 ByteBuffer class, 691, 700, 701, 704, 720  
   get( ) and put( ) methods, table of, 692  
 Bytecode, 9–10, 12, 13, 16, 23–24, 325, 336, 481  
 BYTES, 443, 447, 455  
 byteValue( ), 273, 442, 443, 444, 448, 449, 450, 452

## C

C  
   history of, 4–5  
   and Java, 3, 5, 7, 11  
*C Programming Language, The* (Kernighan and Ritchie), 4  
 C++  
   history of, 5–6  
   and Java, 3, 7, 11  
 C# and Java, 8  
 Calendar class, 586, 587, 588–591, 592, 596, 1013  
   constants, 590  
   methods defined by, table of a sampling of, 588–589  
 Call-by-reference, 136, 137  
 Call-by-value, 136–137, 138  
 call( ), 939, 962  
 Callable interface, 917, 939–942, 962  
 CallSite class, 496  
 cancel( ), 602, 603, 961  
 Canvas class  
   AWT, 798, 801, 886, 1209  
   JavaFX, 1119–1123  
 capacity( ), 433, 563, 690  
 capacityIncrement Vector data member, 563  
 Card layouts, 862–865  
 CardLayout class, 798, 862–863  
 CaretEvent class, 1043  
 Case sensitivity and Java, 23, 25, 32, 188  
 case statement, 84–87, 88–89  
 Casts, 48–50, 338, 341, 342, 344  
   and casting one instance of a generic class into another, 370  
   and erasure, 341, 373  
   using instanceof with, 322–324  
 catch clause(s), 213, 214, 216–219, 222, 224, 232  
   displaying exception description within, 218  
   and the more precise (final) rethrow feature, 231, 232  
   multi-catch feature of, 231–232  
   using multiple, 218–219  
   and nested try statements, 217, 220  
 cbrt( ), 478  
 ceil( ), 478  
 CGI (Common Gateway Interface), 10, 1211–1212  
 changed( ), 1139, 1161  
 ChangeListener interface, 1138–1139  
 Channel interface, 691–692  
 Channel(s), NIO, 690, 691–693. *See also* NIO and channel-based I/O

- char data type, 35, 39–40, 42, 61, 415
  - and automatic type conversion, 48
  - and automatic type promotion, 50
- Character class, 272, 273, 455–458
  - and autoboxing/unboxing, 278
  - methods, table of various, 457, 459
  - support for 32-bit Unicode, 458
- Character(s), 35, 39–40
  - basic multilingual plane (BMP), 458
  - changing case of, 429–430, 456, 457
  - classes (regular expressions), 995, 999
  - code point, 458
  - escape sequences, 43, 44
  - extraction from String objects, 419–420
  - formatting an individual, 609
  - literals, 43
  - supplemental, 458
- Character.Subset class, 458
- Character.UnicodeBlock class, 458
- characteristics( ), 527, 528
- CharArrayReader class, 304, 674–675
- CharArrayWriter class, 304, 675–676
- charAt( ), 153, 419, 434, 493
- CharBuffer class, 691
- chars( ), 493
- CharSequence interface, 413, 430, 435, 493, 994
- Charsets, 416, 693
- charValue( ), 373, 455
- Check boxes, 751
  - AWT, 776, 782, 840–843
  - JavaFX, 1142–1145
  - Swing, 1049–1051
  - and Swing menus, 1081, 1082–1083
- checkAll( ), 892
- Checkbox class
  - AWT, 798, 840–842
  - JavaFX, 1142, 1145
- CheckboxGroup class, 798, 842–843
- CheckboxMenuItem class, 798, 870, 871, 872
- checked... methods, 550, 551–552
- checkedCollection( ), 550, 551
- checkedList( ), 550, 551
- checkedMap( ), 550, 551
- checkedSet( ), 550, 551
- checkID( ), 892, 895
- CheckMenuItem class, 1172, 1183
- Choice class, 798, 844–846
- Choice controls, 782, 844–848
- ChoiceBox control, 1154
- Class class, 281–282, 283, 285, 286, 340, 458, 460, 473–477, 699, 1001, 1002, 1003
  - methods, table of some, 474–475
- .class filename extension, 24, 112
- class keyword, 24, 109
- CLASS retention policy, 281
- Class(es), 109–128
  - abstract, 181–184, 185, 199–200
  - access levels of, 190–191
  - adapter, 791–793
  - anonymous, 16. *See also* Inner classes
  - character, regular expression, 995, 999
  - and code, 23, 109, 190
  - in collections, storing user-defined, 529–530
  - constructor. *See* Constructor
  - controlling access to. *See* Access control
  - as a data type, 109, 110, 113, 114, 116, 126
  - definition of, 19
  - encapsulation achieved through, 126–127
  - final, 185
  - general form of, 109–110
  - generic. *See* Generic class
  - inner. *See* Inner classes
  - instance of a, 19, 109, 111
  - and interfaces, 187, 196, 197–201, 361–362
  - libraries, 23, 34
  - literal, 283
  - member. *See* Member, class
  - name and source file name, 23, 24
  - nested, 149–151
  - packages as containers for, 187, 190, 194
  - public, 191
  - scope defined by a, 46
  - type for bounded types, using a, 347–349
- ClassCastException, 226, 502, 504, 506, 508, 510, 531, 534, 542, 550, 557, 559
- ClassDefinition class, 496
- ClassFileTransformer interface, 496
- ClassLoader class, 477
- classModifiers( ), 1005
- ClassNotFoundException, 227, 685
- CLASSPATH, 188, 189, 1008
- classpath option, 188, 189
- ClassValue class, 493
- clear( ), 502, 503, 532, 570, 581, 588, 690, 1166
- Client/server model, 8, 10, 727
  - and sockets, 731–734
- clone( ), 185, 471–473, 492, 563, 570, 581, 587, 588, 593, 1226
- Cloneable interface, 471–473
- CloneNotSupportedException, 227, 471
- Cloning, potential dangers of, 471–472, 473
- close( ), 310, 312–314, 315, 316, 317, 318, 495, 606, 619, 621, 626, 630, 648, 649, 650, 651, 652, 656, 658, 667, 668, 671, 674, 675, 683, 684, 685, 686, 701, 732, 734, 743, 966
  - within a finally block, calling, 312–314, 649
- Closeable interface, 310, 316, 626, 648, 651, 654, 665, 667, 668, 670, 679, 691
- Closures, 382
- COBOL, 4
- CODE, 760, 761
- Code base, 764
- Code blocks, 28, 30–31, 45, 82–83
  - and the break statement, 104–106
  - and scopes, 45, 46–47
  - static, 145, 326
  - synchronized, 249–250
- Code point, definition of, 458
- Code, unreachable, 108, 219
- CODEBASE, 760
- codePointAt( ), 431, 438, 458, 459
- codePointBefore( ), 431, 438, 459
- codePointCount( ), 431, 438
- codePoints( ), 493



- collect( ), 967, 982–985
- Collection interface, 501–504, 969, 971, 975
  - methods defined by, table of, 502–503
- Collection-view, 499, 531, 571–572
- Collection(s), 337, 497–577
  - algorithms, 499, 550–556, 561
  - into arrays, converting, 503, 513–514
  - and autoboxing, 500, 514
  - classes, 510–521
  - concurrent, 916, 943
  - cycling through, 499, 500, 521–529
  - dynamically typesafe view of a, 550
  - and the for-each version of the for loop, 97, 101, 500, 525–526
  - Framework. *See* Collections Framework
  - generic nature of, 500
  - interfaces, 499, 501–510
  - and iterators, 499, 500, 504, 521–529
  - and legacy classes and interfaces, 561–577
  - modifiable versus unmodifiable, 501
  - and primitive types, 442, 500, 514
  - random access to, 530
  - storing user-defined classes in, 529–530
  - and the stream API, 577, 965, 969
  - stream API stream to obtain a, using a, 982–985
  - and synchronization, 510, 550
  - and type safety, 500, 550
  - when to use, 577
- Collections class, 403, 499, 550, 555, 561
  - algorithms defined by, table of, 551–555
- Collections Framework, 97, 101, 274, 497–577
  - advantages of generics as applied to the, 337, 500
  - JDK 5 changes to, 500
  - legacy classes and interfaces, 561–577
  - and method inferences, 402
  - overview, 498–499
- Collector interface, 982
- Collectors class, 982
- Color class, AWT, 798, 815–818, 1207, 1209
  - constants, 754–755
- Color class, JavaFX, 1121, 1166
- Combo box, JavaFX, 1151–1154
  - enabling users to edit a, 1151, 1153–1154
- Combo boxes, Swing, 1061–1063
- ComboBox class, 1151
- ComboBoxBase class, 1151
- ComboBoxModel interface, 1061
- Comment, 24–25
  - documentation, 32–33, 1235–1241
- Common Gateway interface (CGI), 10, 1211–1212
- commonPool( ), 951, 954
- Compact profiles, 336
- Comparable interface, 358, 361, 423, 493–494, 586, 645
- Comparable<Path> interface, 694
- Comparator interface, 403, 404, 501, 536, 539, 542, 971
- comparator( ), 506, 520, 534
- Comparators, 518, 519, 520, 539, 540, 542–550
  - using a lambda expression with, 546–547
- compare( ), 403–404, 443, 444, 448, 449, 450, 452, 460, 542, 544–545, 971
- compareAndSet( ), 917, 946
- compareTo( ), 269, 270, 423–424, 443, 445, 448, 449, 450, 452, 456, 460, 492, 493, 545, 587, 645
- compareToIgnoreCase( ), 424
- compareUnsigned( ), 450, 452
- comparing( ), 544
- comparingByKey( ), 536
- comparingByValue( ), 536
- Compilation unit, 23
- compile( ), 994
- Compiler class, 481
- Compiler, Java, 23–24
  - and main( ), 25
- Component class, 749, 751, 754, 755, 757, 771, 781, 788, 798, 800–801, 805, 811, 822, 834, 856, 883, 886, 1024, 1025, 1028, 1029, 1036, 1037, 1071
- ComponentAdapter class, 792
- componentAdded( ), 783
- ComponentEvent class, 772, 774, 775, 781
- componentHidden( ), 783
- ComponentListener interface, 782, 783, 792
- componentMoved( ), 783
- componentRemoved( ), 783
- componentResized( ), 783
- Components, AWT, 1021–1022, 1024
  - lightweight versus heavyweight, 883
  - and overriding paint( ), 882–883
- Components, Swing, 1024–1025, 1041–1068
  - architecture, 1023–1024
  - class names for, table of, 1024–1025
  - heavyweight, 1025
  - lightweight, 1022, 1041
  - painting, 1036–1040
  - and pluggable look and feel, 1022–1023
  - and tabbed panes, 1053–1055
- componentShown( ), 783
- ComponentUI, 1024
- compute( ), 949–950, 954, 958, 960, 963
- concat( ), 427
- Concurrency utilities, 14, 915–964
  - versus traditional multithreading and synchronization, 964
- Concurrent API, 915–916
  - packages, 916–917
- Concurrent collection classes, 916, 943
- Concurrent program, definition of, 915
- ConcurrentHashMap class, 917, 943
- ConcurrentLinkedDeque, 943
- ConcurrentLinkedQueue class, 917, 943
- ConcurrentSkipListMap class, 943
- ConcurrentSkipListSet class, 943
- Condition class, 944
- connect( ), 732
- Console class, 680–682
  - methods, table of, 681
- Console I/O, 26, 93, 301, 305–309, 680–682
- console( ), 467, 680
- const keyword, 34
- Constants, 32
- Constructor class, 282, 285, 286, 496, 1002
- Constructor reference, 404–408
  - for an array, 408
  - to generic classes, 405–408

- Constructor(s), 114, 121–124
    - in class hierarchy, order of execution of, 174–175
    - default, 114, 123
    - enumeration, 267–269
    - factory methods versus overloaded, 729
    - generic, 359
    - object parameters for, 135–136
    - overloading, 132–134
    - parameterized, 123–124
    - reference. *See* Constructor reference(s)
    - and super(), 167–170, 174, 336
    - this() and overloaded, 334–336
  - constructorModifiers(), 1005
  - Consumer<T> predefined functional interface, 409, 494, 526, 636, 972, 987
  - Container class, 749, 798, 801, 834, 856, 858, 860, 1024, 1025, 1029, 1037, 1071
  - Container, JavaFX, 1106–1107
  - Container(s), Swing, 1024
    - lightweight versus heavyweight, 1025
    - panes, 1025. *See also* Content pane
    - top-level, 1024, 1025
  - ContainerAdapter class, 792
  - ContainerEvent class, 772, 774–775
  - ContainerListener interface, 782, 783, 792
  - Containment hierarchy, 1024, 1025
  - contains(), 431, 502, 503, 516, 563, 570
  - containsAll(), 502, 503
  - Content pane, 1025, 1028–1029, 1033, 1040, 1054, 1056, 1064, 1067
    - default layout manager of JFrame, 1029, 1032
  - ContentDisplay enumeration, 1129–1130, 1133
  - contentEquals(), 431
  - Context switching, 233, 248, 261
    - rules for, 235
  - ContextMenu class, 1172–1173, 1185–1188
  - ContextMenuEvent class, 1188
  - continue statement, 106–107
  - Control class, 1107, 1112, 1115, 1170
  - Control statements. *See* Statements, control
  - Control(s), AWT, 833, 834–855
    - action events, using an anonymous inner class or lambda expression to handle, 839–840
    - definition of an, 833
    - fundamentals, 834–835
  - Control(s), JavaFX, 1107, 1112, 1114, 1125–1170
    - adding an image to a, 1125, 1128–1133
    - adding a tooltip to a, 1170
    - disabling, 1170
    - and effects and transforms, 1164–1170
  - convert(), 942, 943
  - ConvolveOp built-in convolution filter, 910
  - Convolution filters, 902, 907, 910
  - Cookie class, 1222, 1224–1225, 1230, 1232
    - methods, table of, 1226
  - CookieHandler class, 741
  - CookieManager class, 741
  - CookiePolicy interface, 741
  - Cookies, 741, 1224–1225
    - example servlet using, 1230–1232
  - CookieStore interface, 741
  - copy(), 696, 708–709
  - copyOf(), 521, 522, 557
  - copyOfRange(), 557–558
  - CopyOnWriteArrayList class, 917, 943
  - CopyOnWriteArraySet class, 943
  - copySign(), 480
  - cos(), 38, 477
  - cosh(), 477
  - count(), 967, 973, 990
  - countDown(), 924–925
  - CountDownLatch class, 916, 917, 923–925
  - CountedCompleter class, 948
  - countStackFrames(), 482
  - createImage(), 886, 895, 896, 900
  - createLineBorder(), 1040
  - CropImageFilter class, 899, 900–901
  - Currency class, 604–605
    - methods, table of, 604
  - currentThread(), 237, 482
  - currentTimeMillis(), 467, 469
  - CustomMenuItem class, 1172
  - CyclicBarrier class, 916, 917, 925–927
- ## D
- Data types, 27. *See also* Type(s); Types, primitive
  - DatagramPacket class, 742, 743–745
    - methods, list of some, 744
  - Datagrams, 728, 742–745
    - server/client example, 744–745
  - DatagramSocket class, 692, 742–743, 744–745
  - DataInput interface, 667, 668, 669, 685
  - DataInputStream class, 303, 667, 668–669
  - DataOutput interface, 667, 668, 669, 683
  - DataOutputStream class, 303, 667–669
  - Date and time. *See* Time and date; Time and date API
  - Date class, 586–588, 1010
    - methods, table of, 587
  - DateFormat class, 587, 596, 1009–1011, 1015
  - DateTimeFormatter class, 1015–1017
  - Deadlock, 255–257, 482, 1030
  - decode(), 448, 449, 450, 452, 820
  - Decoder class, 635
  - Decrement operator (--), 30, 61, 64–65
  - decrementAndGet(), 917, 946
  - decrementExact(), 480
  - deepEquals(), 558
  - deepHashCode(), 560
  - deepToString(), 560
  - default
    - clause for annotation member, 287–288
    - to declare a default interface method, using, 208
    - statement, 84–85
  - DefaultMutableTreeNode class, 1064
  - defaults Properties instance variable, 572–573
  - DelayQueue class, 943
  - Delegation event model, 770–771, 1115
    - and Beans, 1202
    - and event listeners, 770, 771, 782–785
    - and event sources, 770–771, 781–782
    - and Swing, 1030
    - using, 785–791
  - delete operator, 125

- delete( ), 436–437, 644, 696
- deleteCharAt( ), 436–437
- deleteOnExit( ), 645
- delimiter( ), 629
- Delimiters, 579–580
  - Scanner class, 621, 628–629
- @Deprecated built-in annotation, 290, 292
- Deque interface, 501, 509–510, 515, 520
  - methods, table of, 509–510
- descendingIterator( ), 507, 509, 510
- destroy( ), 461, 464, 482, 484, 749, 751, 753, 756, 1033, 1212, 1215, 1217
- destroyForcibly( ), 461
- Destructors versus finalize( ), 126
- Dialog boxes, 876–882
  - file, 880–882
- Dialog class, 798, 876
- Diamond operator (<>), 372–373
- Dictionary class, 498–499, 561, 568–569
  - abstract methods, table of, 569
- digit( ), 456
- Dimension class, 798, 802, 814
  - reflection example using the, 1002–1003
- Directories as File objects, 643, 645–646
  - creating, 648
- Directory, listing the contents of a
  - using list( ), 645–647
  - using listFiles( ), 647–648
  - using NIO, 714–717
- Directory tree, obtaining a list of files in a, 717–718
- DirectoryStream<Path> class, 714
- DirectoryStream.Filter interface, 716
- dispose( ), 876
- distinct( ), 990
- divideUnsigned( ), 450, 452
- DLL (dynamic link library), 326, 328
- do-while loop, 90–93
- Document base, 764
- Document interface, 1043
- Document/view methodology, 601
- @Documented built-in annotation, 290, 291
- doDelete( ), 1226, 1227
- doGet( ), 1226, 1227, 1228
- doHead( ), 1226, 1227
- Domain name, 728, 729
- Domain Naming Service (DNS), 728
- doOptions( ), 1226
- doPost( ), 1227, 1229
- doPut( ), 1227
- DosFileAttributes class, 699, 714
- DosFileAttributeView interface, 699
- Dot operator (.), 111, 117–118, 146, 170, 188, 194, 211
- doTrace( ), 1227
- Double buffering, 889–892
- Double class, 272, 273, 442–446, 454
  - methods, table of, 444–446
- double data type, 35, 38–39, 42
  - and automatic type conversion, 48
  - and automatic type promotion, 50–51
- DoubleAccumulator class, 947
- DoubleAdder class, 947
- DoubleBinaryOperator functional interface, 560

- DoubleBuffer class, 691
- doubles( ), 597–598
- DoubleStream interface, 968, 969
- DoubleSummaryStatistics class, 635
- doubleToLongBits( ), 445
- doubleToRawLongBits( ), 445
- doubleValue( ), 273, 347, 442, 443, 445, 448, 449, 450, 452
- drawArc( ), 812, 813–814
- drawImage( ), 887, 890, 891–892
- drawLine( ), 811, 813–814, 1036
- drawOval( ), 812, 813–814
- drawPolygon( ), 813–814
- drawRect( ), 812, 813–814, 1036
- drawRoundRect( ), 812, 813–814
- drawString( ), 319, 748, 754, 756, 767, 825, 832
- Duration class, 1018
- Dynamic link library (DLL), 325–326, 328
- Dynamic method
  - dispatch, 178–181
  - lookup, 198
  - resolution, 196, 198, 199, 204

## E

---

- E (Math constant), 477
- Early binding, 184
- echoCharIsSet( ), 852
- Eclipse IDE, 1212, 1213
- Edit control, 852
- Effect class, 1165
- Effects, 1165–1166
  - list of some built-in, 1165
  - program demonstrating, 1167–1170
- element( ), 508
- elementAt( ), 563
- elementCount Vector data member, 563
- elementData Vector data member, 563
- elements( ), 563, 569, 570
- ElementType enumeration, 291, 496
- ElementType.TYPE\_USE, 293, 206
- else, 81–84
- empty( ), 567, 584, 585
- EMPTY\_LIST static variable, 555
- EMPTY\_MAP static variable, 555
- EMPTY\_SET static variable, 555
- EmptyStackException, 567, 568
- Encapsulation, 5, 18–19, 20, 22–23, 126–127, 167
  - and access control, 141
  - and scope rules, 46
- Encoder class, 635
- end( ), 994
- endsWith( ), 422, 694
- ensureCapacity( ), 433, 513, 563
- entrySet( ), 531, 532, 536, 539, 572
- enum, 263, 492, 521, 541
- Enum class, 269, 492
  - methods, table of, 492
- EnumConstantNotPresentException, 226
- enumerate( ), 482, 485, 488
- Enumeration interface, 561–562, 564–566, 568, 579, 580, 663

- Enumeration(s), 14, 263–272, 492, 566
    - = = relational operator and, 264, 269
    - as a class type in Java, 263, 267–269
    - constants, 263, 264, 267, 268, 269
    - constructor, 267–269
    - restrictions, 269
    - values in switch statements, using, 264–265
    - variable, declaring an, 264
  - EnumMap class, 537, 541–542
  - EnumSet class, 511, 521
    - factory methods, table of, 522
  - Environment properties, list of, 470
  - equals(), 153, 185, 186, 269, 270, 280, 420, 443, 445, 448, 449, 450, 452, 458, 460, 471, 491, 492, 502, 504, 532, 537, 542, 545, 558, 569, 581, 584, 587, 588, 730, 820
    - versus =, 422–423
  - equalsIgnoreCase(), 420
  - Erasure, 341, 373–376
    - and ambiguity errors, 375–377
    - bridge methods and, 374–375
  - err, 304, 305, 467
  - Error class, 214–215, 223, 230, 680
  - Errors
    - ambiguity, 375–376
    - assertions to check for, using, 328–330
    - autoboxing/unboxing and prevention of, 278–279
    - automatic type promotions and compile-time, 50
    - compile-time versus run-time, 344
    - generics and prevention of, 342–344
    - raw types and run-time, 364
    - run-time, 12, 213, 322. *See also* Exception handling
    - unreachable code, 108, 219
  - Event
    - and applets, 751, 769
    - bubbling, 1115
    - change, 1138–1139, 1147, 1151, 1164
    - definition of an, 770
    - design patterns for a Java Bean, 1202
    - dispatch chain, 1115
    - dispatching thread and Swing, 1029–1030, 1033, 1034, 1035
    - driven programs, 769, 1029
    - filter, 1115
    - listeners, 770–771, 782–785
    - loop with polling, 234, 251
    - model, delegation. *See* Delegation event model
    - multicasting and unicast, 771, 1202
    - sources, 770–771, 781–782
    - timestamp, 773
  - Event class, 1115
  - Event handling, 751, 769–795
    - and adapter classes, 791–793
    - event classes, 771–781
    - and inner classes, 151, 793–795, 839, 840
    - and JavaFX, 1112, 1114–1119
    - keyboard, 788–791
    - and lambda expressions, 839–840, 1033
    - mouse, 785–788
    - and Swing, 771, 1022, 1030–1033
    - See also* Delegation event model
  - Event listener interfaces, 782–785
    - and adapter classes, 791–793
    - table of commonly used, 782
  - EventHandler interface, 1115, 1118, 1119
  - EventListener interface, 635
  - EventListenerProxy class, 635
  - EventObject class, 635, 771–772, 1115
  - EventSetDescriptor class, 1202, 1204, 1206
  - Exception, definition of an, 213
  - Exception class, 214–215, 227, 229, 230
  - Exception classes
    - and generics, 379
    - hierarchy of the built-in, 214–215
  - Exception handling, 12, 93, 102, 213–232, 312, 313–314, 315
    - block, general form of, 214
    - and chained exceptions, 13, 230–231
    - and creating custom exceptions, 227–229
    - and the default exception handler, 215–216, 222
    - and lambdas, 394–395
    - and the more precise (final) rethrow feature, 231, 232
    - multi-catch, 231–232
    - and suppressed exceptions, 228, 318
    - and uncaught exceptions, 215–216, 495, 762
  - Exceptions, built-in, 226–227
    - checked, table of, 227
    - run-time, constructors for, 223
    - unchecked, table of, 226
  - Exceptions, I/O, 649
  - exchange(), 927, 928–929
  - Exchanger class, 916, 917, 927–929
  - exec(), 460, 461–462, 464, 465
  - execute(), 937, 951, 960–961
  - Execution point, 491
  - Executor interface, 917, 937
  - Executors, 916
    - using, 937–939
  - Executors class, 917, 937
  - ExecutorService interface, 917, 937, 940
  - exists(), 643, 696, 712
  - exit(), 1078, 1180
  - exitValue(), 461, 464
  - exp(), 478
  - expm1(), 478
  - Expression lambda, 387. *See also* Lambda expression(s)
  - Expressions
    - and autoboxing/unboxing, 276–277
    - automatic type promotion in, 50–51
    - regular. *See* Regular expressions
  - extends, 161, 163, 206, 347, 352, 365
    - and bounded wildcard arguments, 353, 356
  - Externalizable interface, 683, 1203
- ## F
- 
- false, 34, 40, 41, 43, 75, 76, 123
  - FALSE, 458
  - FAT file system, 699, 714
  - Field class, 282, 285, 286, 496, 1002
  - Field, final, 146–147
  - fieldModifiers(), 1005

- fields array, 588
- File attribute(s)
  - File to access, using, 642–645
  - interfaces, 698–699
  - NIO to access, using, 699, 712–714
  - view interfaces, 699
- File class, 620, 642–648, 665, 679, 712
  - instance into a Path instance, converting a, 645, 695
  - methods, 643–645, 652
- file( ), 465
- File(s)
  - to a buffer, map a, 693, 704–705, 707–708, 721–723, 724–725
  - close( ) to close a, using, 310, 312–314, 315, 318, 656
  - I/O, 309–318, 642–648. *See also* NIO; NIO and channel-based I/O
  - path to a, obtaining a, 698, 700
  - pointer, 699, 670
  - source, 23–24
  - system, accessing the, 700
  - try-with-resources to automatically close a, using, 310, 315–318, 656
- FileChannel class, 692, 693, 701, 704, 705–706, 720
- FileDialog class, 798, 880–882
- FileFilter interface, 648
- FileInputStream class, 303, 309–310, 652–654, 692, 720, 721, 722, 723
- FilenameFilter interface, 646–647
- FileNotFoundException, 310, 313, 649, 652, 654, 672
- FileOutputStream class, 303, 309–310, 314, 654–656, 692, 723
- FileReader class, 304, 620, 672
- Files class, 642, 693, 695–697, 699, 708, 709, 712, 714, 717
  - methods, table of a sampling of, 696–697
- FileStore class, 700
- FileSystem class, 700
- FileSystems class, 700
- FileVisitor interface, 717–718
- FileVisitResult enumeration, 718
- FileWriter class, 304, 673–674
- fill( ), 552, 558
- fillArc( ), 812, 813–814
- fillInStackTrace( ), 228
- fillOval( ), 812, 813–814, 1120
- fillPolygon( ), 813
- fillRect( ), 812, 813–814, 1120
- fillRoundRect( ), 812, 813–814
- fillText( ), 1120
- filter( ), 584, 586, 967, 972–973, 980
- FilteredImageSource class, 895, 899–900
- FilterInputStream class, 303, 659, 668
- FilterOutputStream class, 303, 659, 667
- FilterReader class, 304
- FilterWriter class, 304
- final, 146–147
  - to prevent class inheritance, 185
  - to prevent method overriding, 184
- Finalization, 126
- finalize( ), 126, 185, 471
- finally block, 213, 214, 224–226, 312–313, 649
- find( ), 695, 994, 996–997, 998
- findInLine( ), 629–630
- findWithinHorizon( ), 630
- Finger protocol, 735
- fire( ), 1136, 1138, 1139, 1175
- first( ), 506, 863
- firstElement( ), 563
- firstKey( ), 532
- flatMap( ), 584, 586, 982
- flatMapToDouble( ), 982
- flatMapToInt( ), 982
- flatMapToLong( ), 982
- flip( ), 581, 690, 707
- Float class, 272, 273, 442–444, 446, 454
  - methods, table of, 443–444
- float data type, 35, 38, 42
  - and type promotion, 50–51
- Floating-point(s), 35, 38–39
  - literals, 42–43
- FloatBuffer class, 691
- floatValue( ), 273, 442, 443, 445, 448, 449, 450, 452
- floor( ), 478, 507
- floorDiv( ), 478
- floorMod( ), 478
- FlowLayout class, 799, 856–857, 1032
- FlowPane class, 1107, 1110, 1111, 1118–1119, 1187
- flush( ), 606, 648, 652, 661, 671, 681, 683, 684
- Flushable interface, 648, 651, 654, 665, 667, 670, 679, 680
- FocusAdapter class, 792
- FocusEvent class, 772, 774, 775
- focusGained( ), 783
- FocusListener interface, 782, 783, 792
- focusLost( ), 783
- Font class, AWT, 799, 820, 821, 822, 824
  - methods, table of some, 820
- Font class, JavaFX, 1120
- Font(s), 819–825
  - creating and selecting, 822–824
  - determining available, 821–822
  - information, obtaining, 824
  - metrics to manage text output, using, 825–832
  - terminology used to describe, 825
- FontMetrics class, 799, 825–827
  - methods, table of some, 826
- for loop, 29–30, 40, 93–102
  - enhanced. *See* For-each version of the for loop
  - variations, 96–97
- For-each version of the for loop, 14, 93, 97–101
  - and arrays, 97–101
  - and the break statement, 98–99
  - and collections, 97, 101, 500, 501, 525–526
  - general form, 97
  - and the Iterable interface, 494, 500, 525
  - and maps, 531
- forceTermination( ), 936
- forDigit( ), 456
- forEach( ), 494, 532, 967, 968, 972, 977
- forEachOrdered( ), 977
- forEachRemaining( ), 522, 523, 526, 527–528, 988

- Fork/Join Framework, 15, 235, 261, 636, 915–916, 917, 937, 947–964
    - advantages to using the, 947–948
    - classes, main, 948–951
    - tips for using the, 963–964
  - Fork/Join Framework divide-and-conquer strategy, 950, 951–954, 963
    - and the sequential processing threshold
    - interaction with the level of parallelism, 955–958
  - Fork/Join Framework tasks, 949
    - asynchronous execution of, 960–961
    - cancelling, 961
    - completion status of, 961
    - and the parallelism level, 950, 963
    - restarting, 961
    - starting, 951, 960–961
    - and subtasks, 952
    - that do not return a result, 948, 949, 958
    - that return a result, 948, 950, 958
  - fork( ), 949, 951, 954, 958, 960, 962
  - ForkJoinPool class, 917, 937, 948, 949, 950–951, 952, 954, 955, 958, 960–961, 963
    - common pool, 950, 951, 954–955, 958, 963
    - and work stealing, 951, 962
  - ForkJoinTask class, 917, 948–949, 950, 951, 954, 955, 961, 962, 963–964
  - Format flags, 614–617
  - Format specifiers (conversions), 605, 605–619
    - argument index with, using an, 618–619
    - and format flags, 614–617
    - and specifying minimum field width, 612–613
    - and specifying precision, 614
    - suffixes for the time and date, table of, 611
    - table of, 607–608
    - uppercase versions of, 617–618
  - format( ), 431, 606, 607–608, 618, 666, 667, 679, 680, 681, 1009–1010, 1015
  - FormatStyle enumeration, 1015, 1016
  - Formattable interface, 635
  - FormattableFlags class, 635
  - Formatted input, using Scanner to read, 620–630
  - Formatter class, 605–620, 666
    - closing an instance of the, 619–620
    - constructors, 605–606
    - methods, table of, 606
    - See also* Format specifiers
  - forName( ), 474, 1002
  - FORTAN, 4, 5
  - Frame class, 799, 800, 801, 802, 803, 805
  - Frame window(s), 802–810
    - creating a stand-alone, 809–810
    - handling events in, 805–809
    - within an AWT-based applet, creating, 803–804
  - Frank, Ed, 6
  - freeMemory( ), 462–563
  - from( ), 465, 587, 592
  - FTP (File Transfer Protocol), 728, 735
  - Function<T,R> predefined functional interface, 409, 543, 637, 978
  - Functional interfaces, 16, 292, 382, 383–384, 386, 393
    - and their abstract methods, table of, 636–639
    - generic, 389–391
    - predefined, 408–409
  - @FunctionalInterface built-in annotation, 290, 292
  - Future interface, 917, 940–942
  - FXCollections class, 1146, 1151
- ## G
- 
- Garbage collection, 12, 125, 126, 139, 462–463, 496
    - and images, 893
  - gc( ), 462, 463, 467
  - Generic class
    - and casting, 370
    - example program with one type parameter, 338–342
    - example program with two type parameters, 345–346
    - general form of a, 346
    - hierarchies, 364–372
    - and instanceof, 368–370
    - overriding methods in a, 371–372
    - and raw types, 362–364
    - and type inference, 372–373
  - Generic constructors, 359
  - Generic interfaces, 338, 360–362
    - and classes, 361–362
  - Generic method, 338, 350, 356–359, 377
  - Generics, 13, 14, 274, 337–379
    - and annotations, 299
    - and ambiguity errors, 375–376
    - and arrays, 377–379
    - and casts, 338, 341, 344
    - and the Collections Framework, 337, 500
    - and compatibility with pre-generics code, 362–364, 373
    - and exception classes, 379
    - restrictions when using, 377–379
    - type checking and, 341, 342–344, 363, 379
  - GenericServlet class, 1215, 1217, 1220, 1225
  - get( ), 504, 505, 516, 531, 532, 537, 568, 569, 570, 581, 584, 585, 589, 638, 698, 700, 704, 705, 723, 741, 940, 942, 946, 971, 972, 985
    - and buffers, 691, 692, 703, 721
  - getActionCommand( ), 773, 837, 847, 1045, 1051, 1052, 1078
  - getActiveThreadCount( ), 963
  - getAddListenerMethod( ), 1206
  - getAddress( ), 730, 744
  - getAdjustable( ), 774
  - getAdjustmentType( ), 774, 850
  - getAlignment( ), 835
  - getAllByName( ), 729, 730
  - getAllFonts( ), 821
  - getAndSet( ), 917, 946, 947
  - getAnnotation( ), 282, 286, 297–298, 474, 489
  - getAnnotations( ), 285–286, 474, 489
  - getAnnotationsByType( ), 287, 298–299, 474, 489
  - getApplet( ), 761, 765
  - getAppletContext( ), 749, 765
  - getArrivedParties( ), 936
  - getAsDouble( ), 586
  - getAsCent( ), 826
  - getAsInt( ), 586

getAsLong( ), 586  
 getAttribute( ), 1218, 1219, 1225, 1232  
 getAttributeNames( ), 1225, 1232  
 getAudioClip( ), 749, 765, 767  
 getAvailableFontFamilyNames( ), 821  
 getBackground( ), 755  
 getBeanInfo( ), 1206  
 getBlue( ), 816  
 getButton( ), 779  
 getByAddress( ), 730  
 getByName( ), 729  
 getBytes( ), 420, 654  
 getCalendarType( ), 592  
 getCause( ), 228, 230–231  
 getChannel( ), 692, 720, 721, 722, 723  
 getChars( ), 419–420, 434–435, 673  
 getChild( ), 775  
 getChildren( ), 1112–1113, 1160, 1161  
 getClass( ), 185, 186, 281–282, 340, 471, 473, 476, 1003  
 getClickCount( ), 779  
 getCodeBase( ), 749, 764  
 getColor( ), 817  
 getCommonPoolParallelism( ), 958  
 getComponent( ), 774, 1084, 1085, 1086  
 getConstructor( ), 282, 474  
 getConstructors( ), 474, 1002  
 getContainer( ), 775  
 getContentLengthLong( ), 737, 738  
 getContentPane( ), 1029, 1032  
 getContents( ), 633  
 getContentType( ), 737, 738  
 getCookies( ), 1223, 1231  
 getDate( ), 737, 738  
 getDateInstance( ), 1009–1010  
 getDateTimeInstance( ), 1011  
 getDeclaredAnnotation( ), 286  
 getDeclaredAnnotations( ), 286, 474, 489  
 getDeclaredAnnotationsByType( ), 287, 298, 474, 489  
 getDeclaredMethods( ), 475, 1003  
 getDefault( ), 593, 595  
 getDescent( ), 826  
 getDirectionality( ), 458  
 getDirectory( ), 881  
 getDisplayCountry( ), 595  
 getDisplayLanguage( ), 595  
 getDisplayName( ), 595, 604  
 getDocumentBase( ), 749, 764  
 getEchoChar( ), 852  
 getErrorStream( ), 461  
 getEventSetDescriptors( ), 1202, 1209  
 getExpiration( ), 737, 738  
 getExponent( ), 480  
 GetField inner class, 685  
 getField( ), 282, 475  
 GetFieldID( ), 327  
 getFields( ), 475, 1002  
 getFile( ), 881  
 getFileAttributeView( ), 699  
 getFiles( ), 882  
 getFirst( ), 509, 515  
 getFont( ), 820, 824, 826, 1120  
 getForeground( ), 755  
 getForkJoinTaskTag( ), 962  
 getFreeSpace( ), 645  
 getGraphics( ), 757, 811, 890  
 getGraphicsContext2D( ), 1120  
 getGreen( ), 816  
 getHeaderField( ), 737  
 getHeaderFields( ), 737, 741  
 getHeight( ), 826, 1037  
 getHostAddress( ), 730  
 getHostName( ), 731  
 getHour( ), 1017  
 getHvalue( ), 1158  
 getIcon( ), 1042  
 getID( ), 483, 593, 772  
 getImage( ), 749, 765, 886–887  
 getInetAddress( ), 732, 743  
 getInitParameter( ), 1218  
 getInitParameterNames( ), 1218  
 getInputStream( ), 461, 464, 732, 737  
 getInsets( ), 860, 1037  
 getInstance( ), 589, 591, 604  
 getInteger( ), 450  
 GetIntField( ), 327  
 getItem( ), 777, 844, 847, 872, 1048, 1050  
 getItemCount( ), 844, 847  
 getItems( ), 1174, 1179, 1189  
 getItemSelectable( ), 777, 847  
 getKey( ), 537, 539  
 getKeyChar( ), 778  
 getKeyCode( ), 778  
 getLabel( ), 836, 840, 871  
 getLast( ), 509, 515  
 getLastModified( ), 737, 738, 1227  
 getLeading( ), 826  
 getListenerType( ), 1206  
 getLocale( ), 632, 749  
 getLocalGraphicsEnvironment( ), 821  
 getLocalHost( ), 729  
 getLocalizedMessage( ), 228  
 getLocalPort( ), 732, 743  
 getLocationOnScreen( ), 779  
 getLong( ), 452  
 getMaximum( ), 849  
 getMenuComponentCount( ), 1073  
 getMenuComponents( ), 1073  
 getMenuCount( ), 1071  
 getMenus( ), 1173  
 getMessage( ), 223, 228  
 getMethod( ), 282, 284–285, 475, 1206, 1223  
 getMethodDescriptors( ), 1202  
 getMethods( ), 475, 1002  
 getMinimum( ), 849  
 getMinimumSize( ), 856  
 getModifiers( ), 773, 776, 1003  
 getModifiersEx( ), 776  
 getMonth( ), 1017  
 getN( ) getter method design pattern, 1200, 1201  
 getName( ), 236, 238, 340, 475, 483, 485, 490, 643, 694, 712, 820, 1004, 1206, 1226, 1231  
 getNameCount( ), 694  
 getNewState( ), 781  
 GetObjectClass( ), 327  
 getOffset( ), 593, 744  
 getOldState( ), 781

- getOppositeComponent(), 775
- getOppositeWindow(), 781
- getOutputStream(), 461, 464, 732, 1219
- getParallelism(), 958
- getParameter(), 749, 761–762, 1219, 1221, 1228, 1229
- getParameterNames(), 1219, 1221
- getParent(), 485, 643, 694, 712, 936, 1161
- getPath(), 1063–1064, 1226
- getPhase(), 931
- getPoint(), 778
- getPoolSize(), 963
- getPort(), 732, 743, 744
- getPreciseWheelRotation(), 780
- getPreferredSize(), 856
- getPriority(), 236, 246, 483
- getProperties(), 468, 572
- getProperty(), 468, 470, 573, 574, 575
- getPropertyDescriptors(), 1202, 1203, 1208, 1209
- getQueuedTaskCount(), 962
- getRed(), 816
- getRegisteredParties(), 936
- getRemoveListenerMethod(), 1206
- getRGB(), 817
- getRuntime(), 461, 462
- getScreenX(), 1188
- getScreenY(), 1188
- getScript(), 595
- getScrollAmount(), 780
- getScrollType(), 780
- getSecurityManager(), 490
- getSelectedCheckbox(), 842
- getSelectedIndex(), 844, 846, 1059
- getSelectedIndexes(), 847
- getSelectedItem(), 844, 846, 1062
- getSelectedItems(), 847, 1150
- getSelectedText(), 852, 854
- getSelectedToggle(), 1142
- getSelectedValue(), 1059
- getSelectionModel(), 1147, 1160
- getServletConfig(), 1217
- getServletContext(), 1218
- getServletInfo(), 1217
- getServletName(), 1218
- getSession(), 1223, 1232
- getSize(), 802, 814, 820
- getSource(), 772, 838, 1052, 1115
- getStackTrace(), 228, 483, 491
- getState(), 259–261, 483, 840, 871
- getStateChange(), 777, 847
- getSubElements(), 1072
- getSuperclass(), 475, 476
- getSuppressed(), 228, 318
- getSurplusQueuedTaskCount(), 962
- getTarget(), 1180
- getText(), 835, 852, 854, 1042, 1044, 1045, 1050, 1154, 1180
- getTimeInstance(), 1010–1011
- getTransforms(), 1166
- getUnarrivedParties(), 936
- getTotalSpace(), 645
- getUsableSpace(), 645
- getValue(), 537, 539, 774, 849, 1226, 1231, 1151–1152, 1161
- getValue(), 1158
- getWheelRotation(), 780
- getWhen(), 773
- getWidth(), 1037
- getWindow(), 781
- getWriter(), 1215, 1219
- getX(), 778, 1084, 1086
- getXOnScreen(), 779, 1084, 1086
- getY(), 778
- getYear(), 1017
- getYOnScreen(), 779
- GIF image format, 885–886, 887
- Glass pane, 1025
- Glassfish, 1212, 1213
- Glob, 715–716
- Glow class, 1165
  - program demonstrating, 1167–1170
- Gosling, James, 6
- goto keyword, 34
- Goto statement, using labeled break as form of, 104–106
- grabPixels(), 897
- Graphical User Interface. *See* GUI (Graphical User Interface)
- Graphics
  - and JavaFX retained mode, 1106, 1119
  - context, 319, 753, 811
  - sizing, 814–815
- Graphics class, 319, 753, 754, 799, 811, 817, 824, 887, 890
  - drawing methods, 811–814
- Graphics2D class, 811
- GraphicsContext class, 1119–1123
- GraphicsEnvironment class, 799, 821
- GregorianCalendar class, 588, 591–592, 596, 1013
- Grid bag layouts, 865–870
- GridBagConstraints class, 799, 866–868
  - constraint fields, table of, 866–867
- GridBagLayout class, 799, 865, 866, 868, 870
- gridheight constraint field, 866, 868
- GridLayout class, 799, 861–862
- GridPane class, 1107
- gridwidth constraint field, 866, 868
- Group class, 1107
- group(), 699, 994
- GIU (Graphical User Interface), 301, 319, 321, 797, 833
  - applets based on the, 751
  - approaches to the, 1105
  - effects and transforms to customize the look of a JavaFX, using, 1164–1170
  - programs, handling events generated by, 769–795
- GZIP file format, 639

## H

- handle(), 1115, 1118, 1179
- hasCharacteristics(), 527, 528
- Hash code, 516
- Hash table, 516
- hashCode(), 185, 280, 443, 445, 448, 449, 450, 452, 458, 460, 471, 490, 491, 492, 502, 532, 537, 560, 569, 581, 584, 587, 820



- Hashing, 516, 517
- HashMap class, 537–539, 540, 541, 569
- HashSet class, 511, 516–517, 969
  - from a stream API stream, obtaining a, 985
- Hashtable class, 511, 561, 569–572, 573
  - and iterators, 571–572
  - legacy methods, table of, 570
- hasMoreElements(), 562, 580
- hasMoreTokens(), 580
- hasNext(), 522, 523, 986, 987
- hasNextX() Scanner methods, 621, 624
  - table of, 622
- Headers, 737
- HeadlessException, 802, 835
- headMap(), 534, 535
- headSet(), 506, 507
- HEIGHT, 760, 761
- Hexadecimals, 41, 42–43
  - as character values, 43
- Hierarchical abstraction and classification, 18
  - and inheritance, 19, 161
- High surrogate char, 458
- highestOneBit(), 450, 452
- Histogram, 897–899
- Hoare, C.A.R., 236
- Holzmann, Gerard J., 895
- HotSpot technology, 10
- HSB (hue-saturation-brightness) color model, 816
- HSBtoRGB(), 816
- HSPACE, 760, 761
- HTML (Hypertext Markup Language), 1211, 1215
  - file for an applet, 320–321, 748, 760–761
  - and javadoc, 1235, 1236, 1239
- HTML editor, 1156
- HTTP, 728, 735
  - GET requests, handling, 1227–1228
  - and HttpURLConnection class, 739
  - port, 728
  - POST requests, handling, 1227, 1229–1230
  - requests, 1211, 1212, 1222, 1227
  - response, 1211, 1212, 1215, 1222, 1224
  - and URLConnection class, 737
- HTTP session
  - stateful, 741
  - tracking, 1232–1233
- HttpCookie class, 741
- HttpServlet class, 1222, 1225, 1227
  - methods, table of, 1226
- HttpServletRequest interface, 1222, 1228, 1229, 1232
  - methods, table of several, 1223
- HttpServletResponse interface, 1222–1223, 1224
  - methods, table of, 1224
- HttpSession interface, 1222, 1223, 1232
  - methods, table of several, 1225
- HttpURLConnection class, 739–741
  - methods, sampling of, 739
- hypot(), 480
- Identifiers, 24, 32, 34, 44, 45
- IdentityHashMap class, 537, 541
- IEEEremainder(), 480
- if statement, 28–29, 30, 40, 41, 81–84
  - boolean variable used to control the, 82, 278
  - nested, 83
  - and recursive methods, 140
  - switch statement versus, 88–89
- if-else-if ladder, 83–84
- IllegalAccessException, 224, 227
- IllegalArgumentException, 226, 502, 504, 506, 508, 510, 521, 531, 534, 558
- IllegalFormatException, 608
- IllegalMonitorStateException, 226
- IllegalStateException, 226, 502, 510, 994, 1223
- IllegalThreadStateException, 226
- Image class
  - AWT, 799, 885, 886–887, 890, 895, 897
  - JavaFX, 1225–1227
- ImageConsumer interface, 897–899
- ImageFilter class, 899
- ImageIcon class, 1041, 1042
- ImageObserver interface, 887, 888–889, 892
- ImageProducer interface, 886, 895, 897, 899
- imageUpdate(), 888–889
  - bit flags, table of, 889
- Images, 885–913
  - creating, loading, displaying, 886–888
  - double buffering and, 889–892
  - file formats for web, 885–886
  - filters for, 899–912
  - stream model for, 899–900
- Imaging, 885
- ImageView class, 1125–1127, 1128, 1130
- IMG tag, 761
- implements clause, 197
  - and generic interfaces, 361, 362
- import statement, 194–195
  - and static import, 331–334
- in, 304, 305, 464, 467, 620, 680
- Increment operator (++), 30, 61, 64–66
- incrementExact(), 480
- indexOf(), 424–426, 438–439, 504, 505, 563–564
- IndexOutOfBoundsException, 226, 504
- InetAddress class, 731
- Inet6Address class, 731
- InetAddress class, 729–731, 742
- InetSocketAddress class, 743
- infinity (IEEE floating-point specification value), 446
- inForkJoinPool(), 962
- INHERIT, 465
- InheritableThreadLocal class, 488
- Inheritance, 5, 18, 19–21, 22–23, 142, 144, 161–186
  - and annotations, 299
  - and enumerations, 269
  - final and, 184–185
  - and interfaces, 187, 196, 206–207, 210–211, 212
  - multilevel, 171–174
  - and multiple superclasses, 163, 187
- @Inherited built-in annotation, 290, 291
- init(), 750, 751, 753, 755–756, 759, 788, 792, 793, 803, 832, 1212, 1215, 1217
  - and JavaFX, 1107, 1108, 1110
  - and Swing, 1033, 1035

---

**I**

- Icon interface, 1042
- Icons
  - Swing button, 1045
  - Swing label, 1042

- `initCause()`, 228, 230
- Inline method calls, 184
- Inner classes, 149–151, 793–794
  - anonymous, 795, 839–840, 1052, 1071, 1085–1086, 1115, 1119
- InnerShadow class, 1165
  - program demonstrating, 1167–1170
- InputEvent class, 772, 775–776, 777, 778, 1079
- InputMismatchException, 624
- InputStream class, 302, 303, 305, 620, 650, 651, 652, 656, 659, 660, 662, 663, 668, 685, 688, 710, 1220
  - methods, table of, 651
  - objects, concatenating, 663
- InputStreamReader class, 304, 305
- `insert()`, 435–436, 854, 1072
- `insertSeparator()`, 1042
- Insets class, 799, 860–861, 1037
- Instance of a class, 19, 109, 111, 114
  - See also* Object(s)
- Instance variables
  - accessing, 111, 116, 117–118, 120
  - default values of, 123
  - definition of, 19, 110
  - hiding, 125
  - and interfaces, 207
  - static, 145–146
  - as unique to their object, 111, 112–113
  - using super to access hidden, 170–171
- instanceof operator, 322–324, 530
  - and generic classes, 368–370
- Instant class, 587, 1018
- InstantiationException, 227
- Instrumentation interface, 496
- int, 27, 35, 36, 37
  - and automatic type conversion, 48
  - and automatic type promotion, 50–51, 69–70, 72
  - and integer literals, 41
- IntBuffer class, 691
- Integer class, 272, 273, 274, 447, 454–455, 971
  - constructors, 273
  - methods, table of, 450–451
- Integer(s), 35, 36–38, 66–67
  - literals, 41–42
- interface keyword, 187, 196
  - and annotations, 280
- Interface methods
  - default, 16, 197, 207–211, 381, 383
  - static, 211–212
  - traditional, 196, 197–198, 383
- Interface(s), 187, 196–212
  - functional. *See* Functional interfaces
  - general form of, 196–197
  - generic. *See* Generic interfaces
  - implementing, 197–200
  - and the inheritance hierarchy, 196
  - inheritance of, 206–207, 211
  - member, 200
  - methods. *See* Interface methods
  - nested, 200–201
  - reference variables, 198–199, 204
  - types for bounded types, using, 349
  - variables, 197, 204–206
- interfaceModifiers(), 1005
- Internet, 3, 6, 7, 8, 12, 16, 727
  - addresses, obtaining, 729–731
  - addressing scheme, 728
  - and portability, 7, 8, 9
  - and security, 8–9
- Internet Engineering Task Force (IETF) BCP 47, 595
- Internet Protocol (IP)
  - addresses, 728
  - definition of, 727
- InterNIC, 732, 734
- InterruptedException, 227, 237–238, 897
- Introspection, 1200–1203, 1206, 1209
- Introspector class, 1205, 1206
- ints(), 597–598
- IntStream interface, 968, 969, 981
- IntSummaryStatistics class, 635
- intValue(), 273, 274, 442, 444, 445, 448, 449, 450, 452
- InvalidPathException, 698
- invoke(), 949, 951, 955, 960
- invokeAll(), 949, 954, 958, 962
- invokeAndWait(), 1030, 1035
- invokeLater(), 1030, 1035
- I/O, 26, 301–318, 641–688
  - and applets, 319, 321
  - channel-based, 13, 302, 689. *See also* NIO; NIO
  - and channel-based I/O
  - classes, list of, 641–642
  - console, 26, 93, 301, 305–309, 680–682
  - error handling, 312–315
  - exceptions, 649
  - file, 309–318, 642–648
  - formatted. *See* I/O, formatted
  - interfaces, list of, 642
  - new. *See* NIO
  - redirection, 465
  - streams. *See* Streams, I/O
- I/O, formatted, 14
  - format specifiers. *See* Format specifiers
  - using Formatter, 605–620. *See also* Formatter class
  - using printf(), 155, 666–667, 680
  - using Scanner, 620–630. *See also* Scanner class
- io package. *See* java.io package
- IOException, 680
- IOError, 680
- IOException, 93, 305, 310, 313, 314, 649, 651, 652, 656, 662, 670, 673, 683, 684, 685, 695, 714, 717, 732, 736, 742
- ipadx constraint field, 866, 868
- ipady constraint field, 866, 868
- IPv4 (Internet Protocol, version 4), 728, 729, 730, 731
- IPv6 (Internet Protocol, version 6), 728, 729, 730, 731
- isAbsolute(), 644, 695
- isAlive(), 236, 243–246, 461, 483
- isAltDown(), 776
- isAltGraphDown(), 776
- isAnnotationPresent(), 286, 288, 490
- isBound(), 732, 743, 1206
- isCancelled(), 961
- isClosed(), 732
- isCompletedAbnormally(), 961
- isCompletedNormally(), 961
- isConnected(), 732, 743
- isConstrained(), 1206
- isControlDown(), 776

- isDigit( ), 456, 457, 458
- isDirectory( ), 645–646, 696, 698
- isEditable( ), 852, 854
- isEmpty( ), 431, 502, 503, 532, 564, 568, 569, 570, 581
- isEnabled( ), 871, 1091
- isExecutable( ), 696, 712
- isFile( ), 644
- isFinite( ), 444, 445
- isHidden( ), 645, 696, 699, 712
- isIndeterminate( ), 1145
- isInfinite( ), 444, 445, 446–447
- isLeapYear( ), 592
- isLetter( ), 456, 457, 458
- isLightweight( ), 883
- isLowercase( ), 456, 457
- isMetaDown( ), 776
- isMulticastAddress( ), 731
- isMultipleMode( ), 882
- isNaN( ), 444, 445, 446–447
- ISO-Latin-1 character set, 39, 43
- isPopupTrigger( ), 779, 1084, 1086
- isPresent( ), 584, 585, 971
- isPropertyName( ), 1201
- isPublic( ), 1003–1004
- isQuiescent( ), 963
- isReadable( ), 696, 712
- isSelected( ), 1048, 1050, 1052, 1072, 1134, 1135, 1145
- isSet array, 588
- isSet( ), 589
- isShiftDown( ), 778
- isShutdown( ), 963
- isTemporary( ), 775
- isTerminated( ), 963
- isTimeSet, 588
- isUppercase( ), 456, 457
- isWhitespace( ), 456, 457
- isWritable( ), 696, 699, 712
- ItemEvent class, 772, 776–777, 839, 840, 844, 847, 872, 1048, 1050
- ItemListener interface, 782, 783, 840, 844, 872, 1048, 1050
- ItemSelectable interface, 777
- itemStateChanged( ), 783, 840, 844, 1048, 1050
- Iterable interface, 431, 494, 500, 501, 525, 531, 562
- Iterable<Path> interface, 694, 714
- Iteration statements, 81, 89–102
- Iterator, 499, 500, 504, 521–529
  - and maps, 531
  - obtaining an, 523, 524
  - and PriorityQueue, 520
  - and stream API streams, 986–987
  - and synchronized collections, 550
- Iterator interface, 499, 501, 521, 523–525, 526, 562
  - methods, table of, 522
- iterator( ), 494, 502, 504, 523, 714, 966, 986

## J

- J2SE 5, new features of, 13, 14
- JApplet class, 747, 1025, 1033, 1035
- Java

- API packages, table of core, 991–993
- and C, 3, 5, 7, 11

- and C++, 3, 7, 11
- and C#, 8
- design features (buzzwords), 10–13
- history of, 3, 6–8, 13–16
- and the Internet, 3, 6, 7–9, 12, 16, 727
- as interpreted language, 9, 10, 12
- keywords, 33–34
- as a strongly typed language, 10, 11, 35, 41
- versions of, 13–14
- and the World Wide Web, 6, 7, 11
- Java Archive (JAR) files, 639
- Java Beans, 476, 496, 991, 1001, 1199–1209
  - advantages of, 1200
  - API, 1204–1206
  - customizers, 1203
  - demonstration program, 1206–1209
  - introspection, 1200–1203, 1206, 1209
  - properties. *See* Property, Java Bean
  - serialization, 1203
- java filename extension, 23
- Java Community Process (JCP), 16
- Java Control Panel, 748
- Java EE SDK, 1212, 1216
- Java Foundation Classes (JFC), 1022
- java (Java application launcher). *See* Application launcher (java)
- Java Native Interface (JNI), 325
- Java Network Launch Protocol (JNLP), 748, 760, 1111
- java package, 188, 189, 194
- Java SE 7, 14–16
- Java SE 8, 15–16
- Java Virtual Machine (JVM), 9–10, 12, 13, 16, 24, 25, 461, 496
- java.applet package, 301, 319, 747
- java.awt package, 769, 772, 798, 885, 886, 1032
  - classes, tables of some, 798–800
- java.awt.Dimension class, reflection example using the, 1002–1003
- java.awt.event package, 769, 771, 772, 782, 791, 1030, 1032
  - event classes, table of commonly used, 772
  - interfaces, table of commonly used, 772
- java.awt.event.InputEvent class. *See* InputEvent class
- java.awt.event.KeyEvent class. *See* KeyEvent class
- java.awt.image package, 885, 895, 899, 910, 913
- java.beans package, 1202, 1204–1206
  - classes, table of, 1204–1205
  - interfaces, tables of, 1204
- java.io package, 301, 302–304, 310, 316, 641–642, 648, 689, 712
  - classes, list of, 641–642
  - interfaces, list of, 642
- java.io.Externalizable interface, 683, 1203
- java.io.IOException. *See* IOException
- java.io.Serializable interface, 682–683, 687, 962, 1203
- java.lang package, 194, 226, 281, 290, 304, 310, 316, 358, 361, 413, 441–495, 648
  - classes and interfaces, list of, 441
  - implicit importation of the, 194
- java.lang.annotation package, 280, 290, 297, 495, 496
- java.lang.annotation.RetentionPolicy enumeration, 281, 496
- java.lang.image package, 897

- java.lang.instrument package, 495, 496
- java.lang.invoke package, 495, 496
- java.lang.management package, 496
- java.lang.ref package, 496
- java.lang.reflect package, 281, 286, 496, 991, 992, 1001
  - classes, table of, 1002
- java.net package, 727, 741
  - classes and interfaces, list of, 728–729
- java.nio package, 302, 641, 645, 689, 690
- java.nio.channels package, 689, 691, 693
- java.nio.channels.spi package, 689
- java.nio.charset package, 689, 693
- java.nio.charset.spi package, 689
- java.nio.file package, 689, 693, 694
- java.nio.file.attribute package, 689, 693, 698
- java.nio.file.spi package, 689, 693
- java.nio.file.WatchService, 719
- java.rmi package, 991, 992, 1006
- java.text package, 991, 993, 1009
- java.time package, 588, 991, 993, 1013, 1018
- java.time.format package, 1013, 1015
- java.util package, 497–498, 561, 579, 769, 771, 971, 986
  - classes, list of top-level, 497–498
  - interfaces defined by, list of, 498
- java.util.concurrent package, 635, 636, 916–917, 942, 948
- java.util.concurrent.atomic package, 635, 636, 916, 917, 946, 947
- java.util.concurrent.locks package, 635, 636, 916, 917, 943, 944, 946
- java.util.function package, 16, 408–409, 526, 543, 560, 579, 635, 636, 972, 973, 978, 985
  - functional interfaces defined by, table of, 636–639
- java.util.jar package, 635, 639
- java.util.List class. *See* List class
- java.util.logging package, 635, 639
- java.util.prefs package, 635, 639
- java.util.regex package, 636, 639, 991, 993
- java.util.spi package, 636, 639
- java.util.stream package, 16, 636, 639, 966, 982
- java.util.zip package, 636, 639
- javac (Java compiler), 23–24, 188, 293, 364, 1112
- javadoc, 1235–1241
  - tags, 1235–1239
  - utility program, 1235, 1239
- JavaFX, 16, 301, 797, 833, 1105–1123
  - event handling, 1112, 1114–1119
  - images, support for, 1125–1127
  - launcher thread, 1112
  - layout panes, 1107, 1110, 1111, 1118–1119, 1178, 1187, 1196
  - menus. *See* Menus, JavaFX
  - nodes. *See* Node(s), JavaFX
  - packages, 1106
  - repainting, 1106, 1119, 1121
  - scene, 1106–1107, 1110, 1111, 1112
  - scene graph, 1107, 1112–1114, 1118, 1119, 1126, 1157, 1196
  - stage, 1106, 1107, 1110, 1112
  - versus Swing, 1106, 1119
- JavaFX application
  - class, 1107–1108
  - compiling and running a, 1111–1112
  - launching a, 1108
  - skeleton, 1108–1111
  - thread, 1112
- javafx.application package, 1106, 1107, 1110 1080
- javafx.beans.value package, 1139
- javafx.collections package, 1113, 1146
- javafx.event package, 1115, 1116
- javafx.geometry package, 1119
- javafx.scene package, 1106, 1110
- javafx.scene.canvas package, 1119
- javafx.scene.control package, 1112, 1115, 1125, 1136, 1142, 1171
- javafx.scene.effect package, 1196
- javafx.scene.image package, 1125
- javafx.scene.input package, 1181
- javafx.scene.layout package, 1106, 1107, 1110
- javafx.scene.paint package, 1121
- javafx.scene.paint.Color class, 1121, 1166
- javafx.scene.shape package, 1123
- javafx.scene.text package, 1120, 1170
- javafx.scene.transform package, 1166, 1196
- javafx.stage package, 1106, 1110
- javafx.stage.PopupWindow, 1185
- javafxpackager tool, 1108, 1112
- javah.exe, 326, 327
- javap, 375
- javax.imageio package, 913
- javax.servlet package, 1215, 1216–1220
  - interfaces and classes, list of core, 1216–1217
- javax.servlet.http package, 1216, 1222–1227
  - interfaces and classes, list of some, 1222
- javax.swing package, 1024, 1026, 1027, 1041, 1063
  - classes, list of, 1024–1025
- javax.swing.event package, 1030, 1043, 1058, 1063
- javax.swing.table package, 1066, 1067
- javax.swing.tree package, 1063
- JButton class, 1025, 1032, 1041, 1045–1047, 1070, 1091
- JCheckBox class, 1041, 1045, 1047, 1049–1951, 1091
- JCheckBoxMenuItem class, 1070, 1081, 1082–1083
- JComboBox class, 1041, 1061–1063
- JComponent class, 1024, 1025, 1033, 1036, 1037, 1041, 1045, 1071, 1081
- JDialog class, 1025, 1101
- JDK 8 (Java SE 8 Development Kit), 15–16, 23
- JFormattedTextField class, 1101
- JFrame class, 1025, 1026, 1027, 1029, 1040, 1072, 1074
- JIT (Just-In-Time) compiler, 10, 12
- JLabel class, 1025, 1026, 1028, 1030, 1036, 1041–1043, 1074
- JLayeredPane class, 1025
- JList class, 1041, 1058–1060
- JMenu class, 1070, 1071, 1072–1073, 1074
  - mnemonic, 1078
- JMenuBar class, 1070, 1071–1072, 1074
- JMenuItem class, 1070, 1071, 1072, 1073, 1074, 1081, 1082
  - accelerator key, 1079, 1080
  - action to create a, using an, 1091
  - and action events, 1073, 1074, 1077
  - mnemonic, 1078, 1079–1080
- jni.h, 327, 328

jni\_md.h, 328  
 JNLP (Java Network Launch Protocol), 748, 760, 1111  
 join( ), 236, 243–246, 430–431, 483, 949, 958, 960  
 JOptionPane class, 1101  
 Joy, Bill, 6  
 JPanel class, 1025, 1037, 1040, 1055, 1056  
 JPEG image file format, 886, 887  
 JPopupMenu class, 1070, 1083–1086, 1091  
   and mouse events, 1084, 1085–1086  
 JRadioButton class, 1041, 1045, 1047, 1051–1053, 1091  
 JRadioButtonMenuItem class, 1070, 1082–1083, 1091  
 JRootPane class, 1025  
 JScrollBar class, 1025  
 JSeparator class, 1070, 1072  
 JScrollPane class, 1041, 1056–1057, 1058, 1064, 1066, 1067  
 JSpinner class, 1101  
 JTabbedPane class, 1041, 1053–1055  
 JTable class, 1041, 1066–1068  
 JTextComponent class, 1043  
 JTextField class, 1041, 1043–1044  
 JToggleButton class, 1041, 1045, 1047–1049, 1051  
 JToggleButton.ToggleButtonModel class, 1048  
 JToolBar class, 1069, 1087–1089  
   adding an action to a, 1091  
 JTree class, 1041, 1063–1066  
 Jump statements, 81, 102–108  
 Just In Time (JIT) compiler, 10, 12  
 JVM (Java Virtual Machine), 9–10, 12, 13, 16, 24, 25, 461, 482, 496  
 JWindow class, 1025

## K

Kernighan, Brian, 4  
 Key codes, virtual, 777–778, 790  
 KeyAdapter class, 792  
 Keyboard events, handling, 788–791  
 KeyCombination class, 1181  
 keyCombination( ), 1181  
 KeyEvent class, 772, 774, 775, 777–778, 1078, 1079  
 KeyListener interface, 782, 784, 788–791, 792  
 keyPressed( ), 784, 788, 789  
 keyReleased( ), 784, 788  
 keys( ), 568, 569, 570  
 keySet( ), 531, 532, 572, 632, 741  
 KeyStroke class, 1079  
 keyTyped( ), 784, 788, 789  
 Keywords, table of Java, 33

## L

### Label

AWT standard control, 835–836  
 Swing, 1026, 1028, 1041–1043  
   used with break statement, 104–106  
   used with continue statement, 107

### Label class

AWT, 799, 835  
 JavaFX, 1112, 1128  
 Label, JavaFX, 1112–1114  
   adding an image to a, 1128–1130

Labeled class, 1112, 1115

Lambda expression(s), 15–16, 381–396, 408–409

  as arguments, passing, 391–394  
   block, 382, 387–389  
   body, 382, 387–388  
   and comparators, 546–547  
   definition of, 382  
   and exceptions, 394–395  
   and generics, 389  
   to handle action events, 839–840, 1033, 1052, 1071, 1115, 1119  
   parameters, 382–383  
   and the stream API, 965  
   target type, 382, 383, 384, 389–390, 391, 393, 395  
   and variable capture, 395–396

Lambda arrow operator ( $\rightarrow$ ), 16, 61, 382

last( ), 506, 863

lastElement( ), 563, 564

lastIndexOf( ), 424, 425–426, 438–439, 504, 505, 563, 564

lastKey( ), 534

Late binding, 184

launch( ), 1108, 1110

Layered pane, 1025

Layout managers, AWT, 801, 833, 855–870

  default, 833, 855, 856

Layout panes, JavaFX, 1107, 1110, 1111, 1118–1119, 1178, 1187, 1196

LayoutManager interface, 856

Lazy behavior (stream API stream), 968

length instance variable of arrays, 147–149

length( ), 153, 416, 433, 493, 581

Lexer (lexical analyzer), 579

Libraries, class, 23, 24

Library, compact profiles of the API, 336

Lindholm, Tim, 6

LineNumberInputStream deprecated class, 642

LineNumberReader class, 304

lines( ), 676, 695, 969

LinkedBlockingDeque class, 943

LinkedBlockingQueue class, 943

LinkedHashMap class, 537, 540–541

LinkedHashSet class, 511, 517–518

LinkedList class, 511, 515–516

  example program using the, 529–530

  from a stream API stream, obtaining a, 985

LinkedTransferQueue, 943

### List

  controls, 846–848

  items, 773, 776, 782

List class, 799, 846, 847, 1113, 1146, 1150, 1151

List interface, 501, 504, 511, 515, 516, 524, 556, 562, 563, 1173

  from a stream API stream, obtaining a, 982–984

  methods, table of, 505

List, Swing, 1058–1060

List view, 1146–1151

  change events, handling, 1147

  multiple selections in a, enabling, 1150–1151

  scrollbars, 1149–1150

list( ), 573, 695

  and directories, 643, 645–647

- `list()`, `ThreadGroup`, 485, 487
- `listFiles()`, 647–648
- `ListIterator` interface, 501, 521, 524–525, 526
  - methods, table of, 523
- `listIterator()`, 505, 524
- `ListModel`, 1058
- `ListResourceBundle` class, 633
- `ListSelectionEvent` class, 1058, 1059, 1067
- `ListSelectionListener` interface, 1058, 1059
- `ListSelectionModel` interface, 1058–1059, 1067
- `ListView` class, 1146–1151
- Literals, 32, 41–44
  - class, 283
  - regular expression, 995
  - string, 43–44, 416
- `load()`, 462, 468, 573, 576–577
- `loadLibrary()`, 326, 462, 468
- `LocalDate` class, 1013, 1014, 1015, 1017, 1018
- `LocalDateTime` class, 1013, 1014–1015, 1017–1018
- `Locale` class, 430, 594–596, 1009, 1010
- `Locale Data Markup Language (LDML)`, 595
- `Locale.Builder` class, 595
- `LocalTime` class, 1013, 1014, 1017, 1018
- `Lock` interface, 917, 944
  - methods, table of, 944
- `lock()`, 917, 944
- `lockInterruptibly()`, 944
- Locks, 943–946
- `log()`
  - math method, 478
  - Servlet method, 1218, 1220
- `log10()`, 478
- `log1p()`, 478
- Logical operators
  - bitwise, 67–69
  - Boolean, 75–77
- `long`, 35, 36, 37–38
  - and automatic type conversion, 48
  - and automatic type promotion, 50
  - literal, 41–42
- `Long` class, 272, 273, 447, 454–455
  - methods, table of, 452–453
- `LongAccumulator` class, 947
- `LongAdder` class, 947
- `longBitsToDouble()`, 445
- `LongBuffer` class, 691
- `longs()`, 597–598
- `LongStream` interface, 968, 969
- `longValue()`, 273, 442, 444, 445, 448, 449, 450, 452
- Look and feels, 1022–1023
- `lookup()`, 1007
- `loop()`, 767
- `Loop(s)`, 81
  - Boolean object to control, using a, 278
  - continue statement and, 106–107
  - do-while, 90–93
  - for. *See* for loop
  - infinite, 96–97, 103
  - nested, 102, 104, 105–106
  - with polling, event, 234, 251
  - while, 89–90
- Low surrogate char, 458
- `lowestOneBit()`, 450, 452

## M

- `main()`, 25–26, 110, 142, 145
  - and applets, 26, 110, 320, 321, 748
  - and the java application launcher, 25
  - and command-line arguments, 25, 154–155
  - and Swing programs, 1029–1030
  - and windowed applications, 809–810
- `main` (default name of main thread), 238
- `makeGUI()`, 1035
- `MalformedURLException`, 735
- `Map` interface, 531–533, 534, 536, 537, 538, 541, 568, 569, 570, 571–572
  - methods, table of, 532–533
- `map()`, 584, 586, 693, 704, 705, 707, 722, 724
  - and stream API streams, 967, 978–981
- `Map(s)`, 499, 530–542
  - classes, 537–542
  - collection-view of a, obtaining a, 499, 531
  - flat, 982
  - interfaces, 531–537
  - and stream API streams, 978–982
  - submaps of, 534
- `Map.Entry` interface, 531, 536, 539
  - methods, table of non-static, 537
- `MapMode.PRIVATE`, 704
- `MapMode.READ_ONLY`, 704
- `MapMode.READ_WRITE`, 704, 707
- `MappedByteBuffer` class, 691, 704
- `mapToDouble()`, 981
- `mapToInt()`, 981–982
- `mapToLong()`, 981
- `mark()`, 651, 652, 657, 660, 663, 671, 676, 691
- `markSupported()`, 651, 660, 663, 670, 671, 676
- `Matcher` class, 993, 994–995, 996, 997, 999, 1001
- `matcher()`, 994
- `matches()`, 431, 994, 996, 1001
- `Math` class, 45, 131, 477–481
  - rounding methods, table of, 478–479
  - and static import example, 331–333
- `max()`, 403–404, 444, 445, 450, 452, 479, 553, 556, 967, 972
- `MAX_EXPONENT`, 443
- `MAX_PRIORITY`, 246, 482
- `MAX_RADIX`, 455
- `MAX_VALUE`, 443, 447, 455
- `MediaTracker` class, 799, 885, 892–895
- `Member` class, 19, 110
  - access and inheritance, 163–164
  - access, table of, 191
  - controlling access to, 141–144
  - static, 145–146
- `Member` interface, 496, 1001
- Memory
  - allocation using new, 52, 53, 113–114
  - deallocation, 125
  - leaks, 310, 315, 649
  - management, in Java, 11–12, 125
  - and the Runtime class, 462–463
- `MemoryImageSource` class, 895–896, 897, 899
- Menu bars and AWT menus, 833, 870–876
  - action command string of, 872
  - and events, 872

- Menu class
  - AWT, 799, 870, 871
  - JavaFX, 1172, 1173, 1174, 1175, 1179
- Menu item as an event source, AWT, 773, 776, 782
- Menu(s), JavaFX, 1171–1196
  - accelerator keys, 1171, 1180–1181
  - check menu items, 1172, 1183–1185
  - classes, table of core, 1172
  - context menu, 1171, 1172–1173, 1185–1188
  - demonstration program, 1191–1196
  - events, handling, 1172, 1175, 1179–1180
  - and images, 1174, 1182–1183
  - main, creating a, 1172, 1173–1174, 1175–1180
  - menu bar, 1171, 1173–1174, 1175
  - mnemonics, 1171, 1181
  - popup, 1172, 1185
  - radio menu items, 1172, 1183–1185
  - standard menu, 1171
  - and submenus, 1172, 1174, 1179
  - and toolbars, 1171, 1173, 1189–1190
  - and tooltips, 1189
- Menu(s), Swing, 1069–1101
  - accelerator keys, 1069, 1078, 1079–1080, 1093
  - action command string, 1069–1070, 1078
  - action to manage multiple components of a,
    - using an, 1069, 1089–1094
  - and check boxes, 1081, 1082–1083
  - classes, interaction of core, 1069–1070
  - demonstration program, 1095–1101
  - events, 1069–1070, 1073, 1081, 1082, 1084, 1085–1086
  - and images, 1080–1081
  - main, creating a, 1074–1078
  - menu bar, 1069, 1071–1072, 1074
  - mnemonics, 1069, 1073, 1078, 1079–1080, 1093
  - popup, 1069, 1070, 1083–1086
  - and radio buttons, 1081, 1082–1083
  - and submenus, 1070, 1072, 1077
  - and toolbars, 1069, 1070, 1087–1089
  - and tooltips, 1081
- MenuBar class
  - AWT, 799, 870, 871
  - JavaFX, 1172, 1173, 1175
- MenuDragMouseEvent, 1071
- MenuEvent, 1071
- MenuItem class
  - AWT, 799, 870–871, 872, 1081
  - JavaFX, 1172, 1173, 1174–1175, 1179, 1180, 1183
- MenuEvent, 1071
- MenuKeyEvent, 1071
- MenuItemListener, 1071
- Metadata, 280. *See also* Annotation(s)
- Method class, 282, 285, 286, 496, 1002, 1003, 1206
- Method reference(s), 381, 396–404
  - and the Collections Framework, 402
  - and generics, 401–404
  - to instance methods, 397–401
  - to static methods, 396–397
  - to a superclass version of a method, 401
- Method(s), 19, 110, 115–121
  - abstract. *See* Abstract method(s)
  - and annotations, 280, 299
  - and autoboxing/unboxing, 275–276
  - bridge, 374–375
  - calling, 117, 118
  - default interface, 16, 197, 207–211, 381, 383
  - dispatch, dynamic, 178–181
  - and the dot (.) operator, 111, 117, 118
  - factory, 729
  - final, 147, 184
  - general form, 116
  - generic, 338, 350, 356–359, 377
  - getter, 1200
  - hidden, using super to access, 170–171, 176
  - inlining, 184
  - interface. *See* Interface methods
  - lookup, dynamic, 198
  - native, 325–328, 491
  - overloading, 129–134, 158–160, 177
  - overriding. *See* Overriding, method
  - and parameters, 116, 119–121
  - passing an object to, 137–138
  - recursive, 139–141
  - reference. *See* Method reference(s)
  - resolution, dynamic, 196, 198, 199, 204
  - returning an object from, 138–139
  - returning a value from, 118–119, 121
  - scope defined by, 46–48
  - setter, 1200
  - static, 145–146, 211–212, 332–333, 396–397
  - subclasser responsibility, 182
  - synchronized, 236, 247–249
  - type inference and, 358, 372–373
  - varargs. *See* Varargs
  - variable-arity, 155
- MethodDescriptor class, 1202, 1205, 1206
- MethodHandle class, 496
- methodModifiers(), 1005
- MethodType class, 496
- MIME (Multipurpose Internet Mail Extensions), 1211, 1215
- min(), 444, 445, 450, 452, 479, 553, 556, 967, 971, 972
- minimumLayoutSize(), 856
- MIN\_EXPONENT, 443
- MIN\_NORMAL, 443
- MIN\_PRIORITY, 246, 482
- MIN\_RADIX, 455
- MIN\_VALUE, 443, 447, 455
- mkdir(), 648
- mkdirs(), 648
- Model-Delegate component architecture, 1023–1024
- Model-View-Controller (MVC) component architecture, 1023
- Modifier class, 1003, 1005
  - "is" methods, table of, 1004
- Modulus operator (%), 61, 63
- Monitor, 236, 247, 249, 251
- Mouse events, handling, 785–788
- MouseAdapter class, 792, 793, 794, 1084, 1085, 1207
- mouseClicked(), 784, 792, 1084
- mouseDragged(), 784, 791, 792, 891
- mouseEntered(), 784, 1084
- MouseEvent class, 772, 774, 775, 778–779, 1084
- mouseExited(), 784, 1084

MouseListener interface, 782, 784, 785–788, 792, 793, 1084, 1085–1086  
 MouseMotionAdapter class, 791, 792  
 MouseMotionListener interface, 771, 782, 784, 785–788, 791, 792, 793  
 mouseMoved( ), 784, 791, 891  
 mousePressed( ), 784, 793–794, 1084, 1086, 1207  
 mouseReleased( ), 784, 1084, 1086  
 MouseWheelEvent class, 772, 779–780  
 MouseWheelListener interface, 782, 784, 785, 792  
 mouseWheelMoved( ), 784  
 Multi-core systems, 234–235, 261, 915, 916, 947–948, 952  
 MultipleSelectionModel class, 1147, 1149  
 multiplyExact( ), 480  
 Multitasking, 233  
   preemptive, 235  
 Multithreaded programming, 7, 11, 12, 233–261  
   and context switching. *See* Context switching  
   effectively using, 261  
   and multi-core versus single-core systems, 234  
   and spurious wakeup, 251  
   and StringBuilder class, 439  
   and synchronization. *See* Synchronization  
   and threads. *See* Thread(s)  
   versus the concurrency utilities, traditional, 915, 964  
   and parallel programming, 948  
   versus single-threaded system, 234  
 MutableComboBoxModel, 1061  
 MutableTreeNode interface, 1064  
 MVC (Model-View-Controller) component  
   architecture, 1023

## N

NAME, 760, 761  
 Name-space collisions  
   between instance variables and local  
   variables, 125  
   packages and, 187, 194, 334  
 Naming class, 1006, 1007  
 NaN, 443, 446  
 nanoTime( ), 468, 469, 955  
 @Native built-in annotation, 290  
 native modifier, 325  
 Natural ordering, 494, 452  
 naturalOrder( ), 543  
 Naughton, Patrick, 6  
 NavigableMap interface, 531, 534, 539  
   methods, table of, 535–536  
 NavigableSet interface, 501, 507–508, 518, 519  
   methods, table of, 507  
 negateExact( ), 480  
 Negative numbers in Java, representation of, 66–67  
 NEGATIVE\_INFINITY, 443  
 NegativeArraySizeException, 226, 557  
 .NET Framework, 8  
 NetBeans, 1112, 1212, 1213  
 Networking, 727–745  
   basics, 727–728  
   classes and interfaces, list of, 728–729  
   new, 52, 53, 113–114, 121, 123, 125, 139, 182, 222, 223  
     autoboxing and, 275  
     constructor reference and, 404, 408  
     and enumerations, 264, 267  
     and type inference, 372–373  
 NEW, 260  
 New I/O. *See* NIO  
 newByteChannel( ), 693, 696, 701, 702, 703, 704, 705, 706, 707, 708  
 newCachedThreadPool( ), 937  
 newCondition( ), 944  
 newDirectoryStream( ), 696, 714, 715–717  
 newFileSystem( ), 700  
 newFixedThreadPool( ), 937  
 newInputStream( ), 697, 709, 710–711  
 Newline, inserting a, 612  
 newOutputStream( ), 697, 709, 711  
 newScheduledThreadPool( ), 937  
 next( ), 522, 523, 623, 863, 986, 987  
 nextAfter( ), 479  
 nextBoolean( ), 596, 623  
 nextBytes( ), 596  
 nextDouble( ), 205, 596, 623, 625, 628  
 nextDown( ), 479  
 nextElement( ), 562, 580, 665  
 nextFloat( ), 596, 623  
 nextGaussian( ), 596  
 nextInt( ), 596, 623, 628  
 nextLong( ), 596, 623  
 nextToken( ), 580  
 nextUp( ), 479  
 nextX( ) Scanner methods, 621, 624, 625, 628  
   table of, 623  
 NIO, 641, 689–725  
   and directories, 714–719  
   packages, list of, 689  
   pre-JDK 7 NIO versus new, 720  
   reading a file using pre-JDK 7, 720–723  
   for path and file system operations, using, 712–719  
   and the stream API, 695  
   for stream-based I/O, using, 700, 709–711  
   writing to a file using pre-JDK 7, 723–725  
 NIO and channel-based I/O  
   copying a file using, 708–709  
   reading a file using, 701–705  
   writing to a file using, 705–708  
 NIO.2, 689, 700, 712  
 Node class, 1107, 1111, 1115, 1119, 1126, 1128, 1160, 1165, 1166, 1170, 1172, 1173, 1187  
 Node(s), JavaFX, 1107, 1110, 1111, 1113, 1118, 1119  
   disabling, 1170  
   effects and transforms to alter the look of, using, 1164–1170  
   hierarchy, 1107  
   scrolling capabilities to, adding, 1157–1159  
   text, 1170  
   tree, 1160–1161, 1164  
 noneMatch( ), 990  
 NORM\_PRIORITY, 246, 482  
 NoSuchElementException, 506, 508, 510, 534, 562, 624, 630  
 NoSuchFieldException, 227



NoSuchMethodException, 227, 282  
 NOT operator  
     bitwise unary (~), 66, 67, 68–69  
     Boolean logical unary (!), 75–76  
 NotDirectoryException, 714  
 notepad, 464, 467  
 notify(), 185, 186, 251, 253–255, 258–259, 471, 915, 944, 964  
 notifyAll(), 185, 186, 251, 471  
 notifyObservers(), 598–599  
 NotSerializableException, 687  
 now(), 1014–1015  
 null, 34, 123  
     alternative to using, 584  
 Null statement, 90  
 NullPointerException, 223, 226, 502, 504, 506, 508, 510, 521, 531, 534, 557, 570, 631, 665  
     using Optional to prevent a, 584, 586  
 nullsFirst(), 543  
 nullsLast(), 543  
 Number class, 273, 347, 442  
 NumberFormatException, 226, 273, 762  
 numberOfLeadingZeros(), 450, 452  
 numberOfTrailingZeros(), 451, 453  
 Numbers, formatting, 609–610, 612–618

## O

Oak, 6  
 Object class, 185–186, 338, 340, 373, 471–473  
     as a data type, problems with using the, 342–344  
 Object class methods  
     and functional interfaces, 382  
     table of, 185, 471  
 Object reference variables  
     and abstract classes, 182, 184  
     and argument passing, 136, 137–138  
     assigning, 115  
     declaring, 113  
     and cloning, 471–472  
     and dynamic method dispatch, 178–181  
     to superclass reference variable, assigning  
         subclass, 166, 170  
 OBJECT tag, 320, 748, 761  
 Object-oriented programming (OOP), 5, 6, 17–23, 109  
     model in Java, 11  
 Object(s), 19, 109, 114  
     bitwise copy (clone) of, 471  
     creating/declaring, 111, 113–114  
     initialization with a constructor, 121, 123–124  
     to a method, passing, 137–138  
     monitor, implicit, 236, 249  
     as parameters, 134–136  
     returning, 138–139  
     serialization of. *See* Serialization  
     type at run time, determining, 322–324  
 Object.notify(). *See* notify()  
 Object.wait(). *See* wait()  
 ObjectInput interface, 685  
     methods defined by, table of, 685  
 ObjectInputStream class, 303, 685  
     methods defined by, table of, 686  
 ObjectOutputStream interface, 683, 684  
     methods defined by, table of, 683  
 ObjectOutputStream class, 303, 684  
     methods defined by, table of, 684  
 Objects class, 635  
 Observable class, 598–601  
     methods, table of, 598  
 observableArrayList(), 1146, 1149, 1151  
 ObservableList, 1113, 1114, 1146, 1149, 1150, 1151, 1173  
 ObservableValue, 1139  
 Observer interface, 598–601  
 Octals, 41  
     as character values, 43  
 of(), 521, 522, 584, 585, 990  
 offer(), 508, 520  
 offerFirst(), 509, 510, 515  
 offerLast(), 509, 510, 515  
 offsetByCodePoints(), 431, 438  
 ofLocalizedDate(), 1015  
 ofLocalizedDateTime(), 1015  
 ofLocalizedTime(), 1015  
 ofNullable(), 585, 586  
 ofPattern(), 1016–1017  
     pattern letters, 1016–1017  
 onAdvance(), 933–934, 936  
 open(), 693  
 openConnection(), 736, 738–739  
 OpenOption interface, 695  
 Operator(s)  
     arithmetic, 61–66  
     assignment. *See* Assignment operator(s)  
     bitwise, 66–74  
     Boolean logical, 75–77  
     conditional-and, 77  
     conditional-or, 77  
     diamond (<>), 372–373  
     parentheses and, 41, 79  
     precedence, table of, 78  
     relational, 28, 40, 41, 74–75  
     ternary if-then-else (?:), 75, 77–78  
 Optional class, 584–586, 971, 972, 973  
     methods, table of, 584–585  
 OptionalDouble class, 584, 586  
 OptionalInt class, 584, 586  
 OptionalLong class, 584, 586  
 OR operator  
     bitwise (|), 66, 67, 68–69  
     bitwise exclusive (^), 66, 67, 68–69  
     Boolean logical (|), 75–76  
     Boolean logical exclusive (^), 75–76  
 OR operator, short-circuit (||) Boolean logical, 75, 76–77  
 Oracle, 14, 1212  
 Ordinal value, enumeration constant's, 269  
 ordinal(), 269, 270, 492  
 orElse(), 585  
 out output stream, 26, 34, 304, 305, 308, 309, 464, 467, 620, 665, 666, 680  
 out(), 606, 608  
 OutputStream class, 302, 303, 308, 650, 651, 654, 659, 661, 665, 667, 679, 684, 711, 1220  
     methods, table of, 652  
 OutputStreamWriter class, 304

Overloading methods, 129–134, 158–160, 177, 375–376  
 @Override, built-in annotation, 290, 292  
 Overriding, method, 175–181  
   and abstract classes, 181–184  
   and bridge methods, 374–375  
   and dynamic method dispatch, 178–181  
   final to prevent, using, 184  
   in a generic class, 371–372  
   and run-time polymorphism, 178, 179, 181

## P

Package(s), 142, 187–196, 212  
   access to classes contained in, 190–194, 195  
   built-in standard Java classes and, 194  
   core Java API, table of, 991–993  
   the default, 188, 194  
   defining, 188  
   finding, 188–189  
   importing, 194–196  
   Swing, 1024  
   version data, obtaining, 489  
 Package class, 286, 489–490  
   methods, table of, 489–490  
 package statement, 188, 194  
 Paint class, 1121  
 Paint mode, setting, 818–819  
 paint(), 319, 751–752, 753, 754, 755–756, 757, 759, 786, 805, 811, 887, 891, 895, 1033, 1036, 1037  
   lightweight AWT components and overriding, 882–883  
 Paintable area, computing, 1037  
 paintBorder(), 1036  
 paintChildren(), 1036  
 paintComponent(), 1036, 1037, 1040  
 Painting in Swing, 1036–1040  
 Panel class, 749, 799, 800, 801, 863  
 Panes, Swing container, 1025. *See also* Content pane  
 Parallel processing, 16, 381, 526, 528  
   of a stream API stream, 965, 968, 969, 975–977, 984, 986, 987, 989  
 Parallel programming. *See* Programming, parallel  
 parallel(), 966, 975  
 parallelPrefix(), 560  
 parallelSetAll(), 560  
 parallelSort(), 559  
 parallelStream(), 502, 504, 969, 975, 976  
 PARAM NAME and VALUE, 760, 761  
 Parameter(s), 25, 116, 119–121  
   applets and, 761–764  
   and constructors, 123–124  
   final, 147  
   and lambda expressions, 382–383, 385–387, 395  
   objects as, 134–136  
   and overloaded constructors, 134  
   and overloaded methods, 129, 177  
   and the scope of a method, 46  
   servlet, reading, 1220–1222  
   type. *See* Type parameter(s)  
   variable-length (varargs), 157, 521  
 Parameterized types, 338, 340  
 parameterModifiers(), 1005

Parent class, 1007, 1111, 1115  
 parse(), 1017–1018  
 parseBoolean(), 460  
 parseByte(), 448, 454  
 parseDouble(), 445  
 parseFloat(), 444  
 parseInt(), 451, 454  
 parseLong(), 453, 454  
 parseShort(), 449, 454  
 parseUnsignedInt(), 451  
 parseUnsignedLong(), 453  
 Parsing, definition of, 579  
 Pascal, 4  
 PasswordField class, 1156  
 Passwords, reading, 680  
 Path interface, 642, 645, 694–695, 700, 701, 712, 714, 720  
   converting a File object into an instance of the, 645, 695, 712  
   instance for stream-based I/O, using a, 709–711  
   methods, table of a sampling of, 694–695  
   obtaining an instance of the, 698, 700, 701, 702, 703–704, 707  
 Paths class, 698, 700  
 Pattern class, 993–994, 997, 1000, 1001  
 Pattern matching, regular expressions, 995–1001  
 PatternSyntaxException, 995  
 Payne, Jonathan, 6  
 peek(), 508, 567  
 peekFirst(), 509, 515  
 peekLast(), 509, 515  
 Peers, native, 883, 1021–1022  
 Period class, 1018  
 Persistence (Java Beans), 1203  
 Phaser class, 916, 917, 930–936  
   compatibility with fork/join, 963  
 PI (Math constant), 477  
 PIPE, 465  
 Pipeline for actions on stream API streams, 16, 381, 968, 980  
 PipedInputStream class, 303  
 PipedOutputStream class, 303  
 PipedReader class, 304  
 PipedWriter class, 304  
 PixelGrabber class, 897–899  
 Platform class, 1180  
 Platform.exit(), 1180  
 play(), 750, 767  
 Pluggable look and feel (PLAF), 1022–1023, 1024  
 PNG file format, 886, 887  
 Point class, 778, 779, 799  
 Pointers, 59, 113  
 poll(), 508, 520  
 pollFirst(), 507, 509, 515  
 Polling, 234, 251  
 pollLast(), 507, 509, 515  
 Polygon class, 799, 813  
 Polymorphism, 5, 18, 21–23  
   and dynamic method dispatch, 178–181, 182  
   and interfaces, 196, 199, 204  
   and overloaded methods, 129, 131, 132  
 pop(), 509, 510, 567