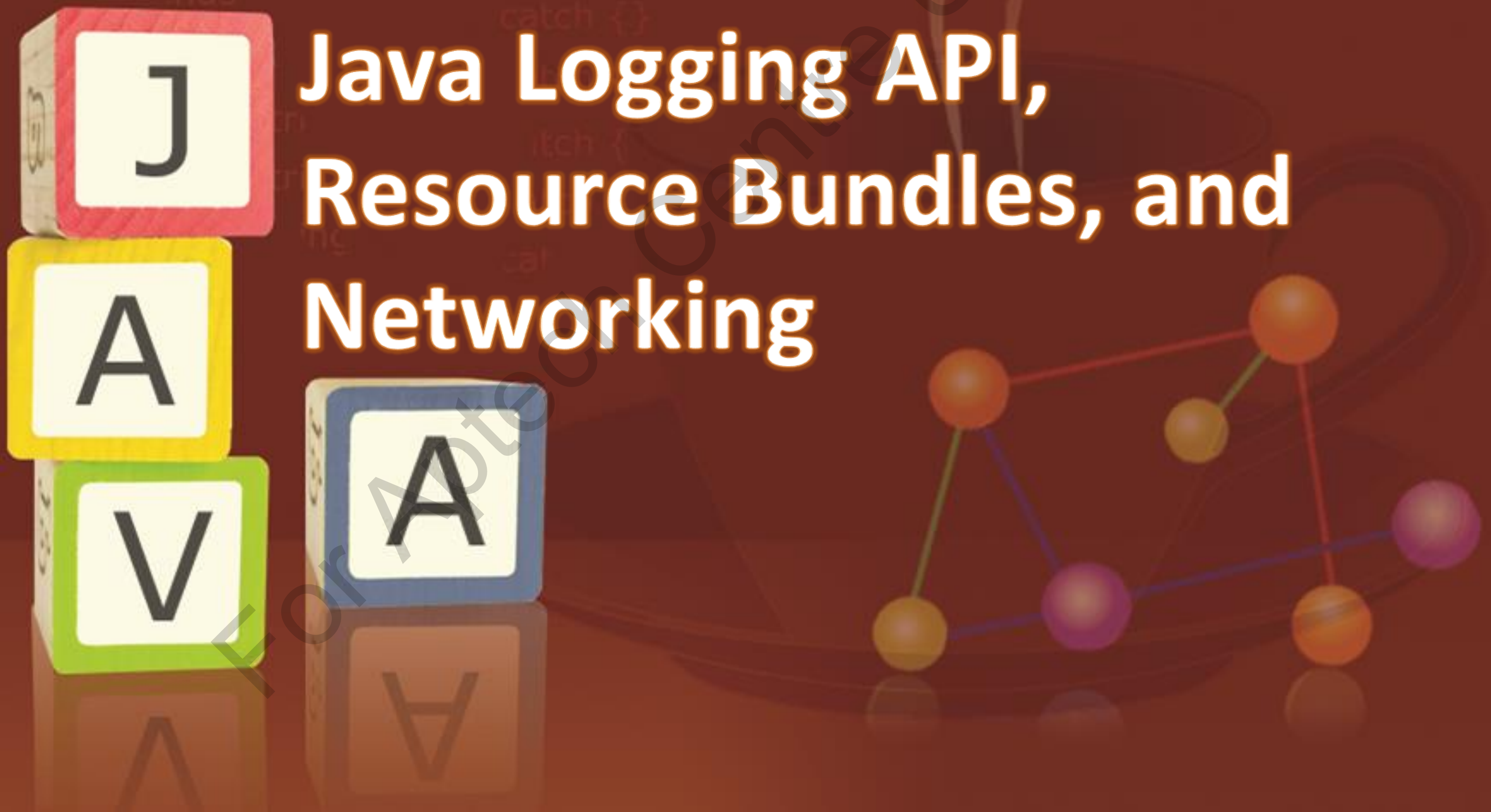
# Power Programming in Java



## Session: 13

## Java Logging API, Resource Bundles, and Networking

## Objectives

- Describe the Log4J architecture
- Identify Log4J configuration options
- Explain the file appender
- Explain the JDBC appender
- Identify the ResourceBundle class
- Explain the Javadoc tool and documentation in Java
- Explain java.net.package
- Explain Socket Programming and URL Processing

## Log4J Overview

- Is an open-source logging framework for Java applications
- Enables generating log messages from different parts of the application
- Allows debugging the application for errors and tracing the execution flow
- Assigns different level of importance, such as ERROR, WARN, INFO, and DEBUG
- Can be routed to different types of destinations, such as console, file, and database
- Is composed of three primary components:

Loggers | Appenders | Layouts

## Loggers [1-2]

- Logger
  - Is the primary Log4J component that is responsible for logging messages
- Developers can:

Create their own application-specific loggers

Use the Log4J root logger

- Log4J2 searches for an application-specific logger or uses the root logger
- The root logger can be instantiated and retrieved by calling method:

```
LoggerManager.getRootLogger()
```

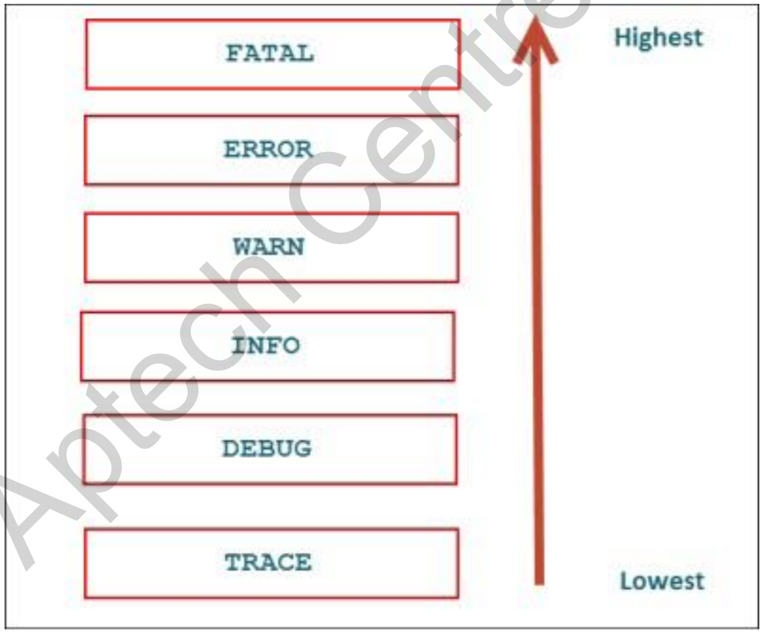
- Application loggers can be instantiated and retrieved by calling method:

```
LoggerManager.getLogger(String loggerName)
```

It is recommended to name the logger with the fully qualified name of the class that will perform logging.

## Loggers [2-2]

- Loggers are assigned log levels where TRACE is the lowest level. The levels move up from TRACE through DEBUG, INFO, WARN, and ERROR, until the highest FATAL level.
- When a higher level is assigned to a logger:
  - All log messages of that level and the levels below it are logged.



- For example, if the INFO level is assigned to a logger, then INFO, DEBUG, and TRACE messages are logged by the logger.

## Appenders

### What are Appenders?

- Loggers log messages to output destinations, such as console, file, and database. Such output destinations are known as appenders.
- Log4J provides a number of appender classes to log messages to various destinations.

### Example

`ConsoleAppender` logs messages to the console, `FileAppender` logs messages to a file, and `JDBCAppender` log messages to a relational database table.

- Log4J also allows defining custom appenders. A customer appender:

Extends from the `AppenderSkeleton` class that defines the common logging functionality.

The core method of `AppenderSkeleton` that a custom appender should override is the `append()` method.

## Layout

- Layouts:
  - Define how log messages are formatted in the output destination
  - Are associated with appenders
- Log4J provides built-in layout classes, such as:
  - `PatternLayout`, `Htmlayout`, `JsonLayout`, and `XmlLayout`

Log4J also supports custom layout that can be created by extending the abstract `AbstractStringLayout` class.

## Logging Methods

- For each of the log levels, Log4J defines a corresponding log method.

| Method | Description |
|---|---|
| `trace()` | Logs a method with the TRACE level. |
| `debug()` | Logs a method with the DEBUG level. |
| `info()` | Logs a method with the INFO level. |
| `warn()` | Logs a method with the WARN level. |
| `error()` | Logs a method with the ERROR level. |
| `fatal()` | Logs a method with the FATAL level. |
| `keySet()` | Returns a Set of all keys in the ResourceBundle. |

## Creating a Logging Application [1-2]

- If you are creating a Java Maven project, you do not require to download the log4j files as they are available in the Maven repository and you just have to add dependencies.
  1. Open Apache NetBeans.
  2. Create a **Log4JDemo** Java Maven application project.
  3. Under **Project Files** in **Projects** window, locate **`pom.xml`** and open it.
  4. Add the bolded code in **`pom.xml`**:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>logging</groupId>
    <artifactId>loggingdemo</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>jar</packaging>
    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <maven.compiler.source>15</maven.compiler.source>
        <maven.compiler.target>15</maven.compiler.target>
    </properties>
   <dependencies>
  <dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-api</artifactId>
    <version>2.13.0</version>
  </dependency>
  <dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-core</artifactId>
    <version>2.13.0</version>
  </dependency>
</dependencies>
</project>
```

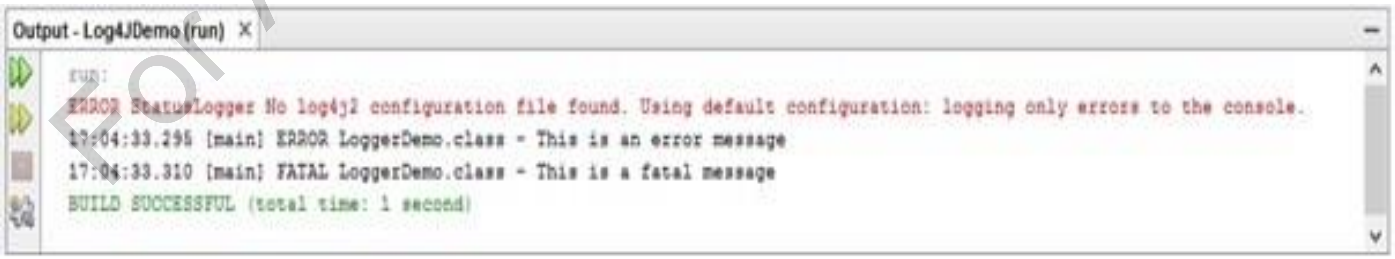
## Creating a Logging Application [2-2]

- Code Snippet shows a `LoggerDemo` class that uses all the log methods.

Code Snippet

```
package com.log4j.demo;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
public class LoggerDemo {
  private static Logger = LogManager.getLogger("LoggerDemo.class");
     public void performLogging(){
         logger.debug("This is a debug message");
         logger.info("This is an info message");
         logger.warn("This is a warn message");
         logger.error("This is an error message");
         logger.fatal("This is a fatal message");
     }
     public static void main(String[] args){
        LoggerDemo logger =new LoggerDemo();
        logger.performLogging();
     }
}
```

Output:



Output - Log4JDemo (run)

```
run:
ERROR StatusLogger No log4j2 configuration file found. Using default configuration: logging only errors to the console.
17:04:33.295 [main] ERROR LoggerDemo.class - This is an error message
17:04:33.310 [main] FATAL LoggerDemo.class - This is a fatal message
BUILD SUCCESSFUL (total time: 1 second)
```

## Properties File Configuration [1-2]

- Following code snippet demonstrates a `log4j2.properties` configuration file:

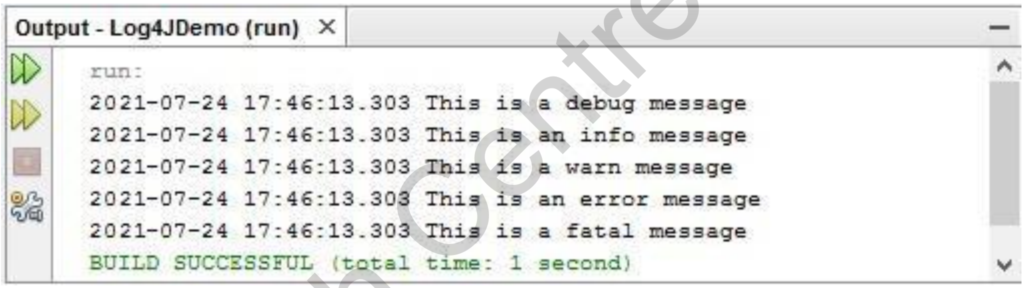
Code Snippet

```
name = PropertiesConfig
appenders = consoleappender
appender.consoleappender.type = console
appender.consoleappender.name = STDOUT
appender.consoleappender.layout.type = PatternLayout
appender.consoleappender.layout.pattern = %d{yyyy-MM-dd HH:mm:ss.SSS} %msg%n
rootLogger.level = debug
rootLogger.appenderRefs = stdout
rootLogger.appenderRef.stdout.ref = STDOUT
```

- In the configuration code:
  - The name and `appenders` properties specify the name of the configuration and the `appender` to use respectively.
  - The properties starting with appender configures the appender to use.
  - The `appender.consoleappender.type` property specifies console to use the Log4J console appender.
  - The appender.consoleappender.layout.type and `appender. consoleappender.layout.pattern` properties specifies the pattern layout to use for the appender and the specific pattern to use.
  - The `rootLogger.level` property configures the root logger with the DEBUG level.
  - The `rootLogger.appenderRefs` and `rootLogger.appenderRef.stdout.ref` properties associate the console appender with the root logger.

## Properties File Configuration [2-2]

- Following figure demonstrates how the root logger outputs all the log messages:



```
run:
2021-07-24 17:46:13.303 This is a debug message
2021-07-24 17:46:13.303 This is an info message
2021-07-24 17:46:13.303 This is a warn message
2021-07-24 17:46:13.303 This is an error message
2021-07-24 17:46:13.303 This is a fatal message
BUILD SUCCESSFUL (total time: 1 second)
```

## XML File Configuration [1-2]

- Following code snippet demonstrates a `log4j2.xml` configuration file:

Code Snippet

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration name="PropertiesConfig">
<Appenders>
 <Console name="consoleappender" target="STDOUT">
<PatternLayout>
 <pattern>
  %d{yyyy-MM-dd HH:mm:ss.SSS} %msg%n
 </pattern>>
</PatternLayout>
</Console>
</Appenders>
<Loggers>
 <Root level="DEBUG">
  <AppenderRef ref="consoleappender"/>
 </Root>
</Loggers>
</Configuration>
```

## XML File Configuration [2-2]

- A Log4J XML configuration file contains the `<Configuration>` root element

The `<Appenders>` element contains a `<Console>` element to configure a console appender.

The `<PatternLayout>` element specifies the pattern layout to use with the appender and the `<pattern>` element specifies the formatting pattern to use.

The `<Loggers>` element contains the <Root> element to configure the root logger.

The level attribute of the `<Root>` element assigns the `DEBUG` log level to the root logger the `ref` attribute of the `<AppenderRef>` element assigns the `console` appender to the root logger.

## File Appender

- Following code snippet demonstrates the use of a file appender:

Code Snippet

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration name="PropertiesConfig">
<Appenders>
 <File name="fileappender" fileName="applogs/logfile.log" >
  <PatternLayout>
   <pattern>
    %d{yyyy-MM-dd HH:mm:ss.SSS} %msg%n
   </pattern>
  </PatternLayout>
 </File>
</Appenders>
<Loggers>
 <Root level="DEBUG">
  <AppenderRef ref="fileappender"/>
 </Root>
</Loggers>
</Configuration>
```

# JDBC Appender

- The `<JDBC>` element configures a JDBC appender.
- To use a JDBC appender, you require the following mandatory information:
  - The connection URL to the database
  - The database table name to insert log messages
  - The columns in the table to write log messages
- To use the JDBC appender, a relational database server is required. You can download MySQL from:

  `http://dev.mysql.com/downloads/windows/installer/5.7.html`
- Following code snippet demonstrates the statements to create a database and a table:

Code Snippet

```
mysql>create database LOG4JLOG;
mysql>use LOG4JLOG;
mysql>CREATE TABLE applicationlog (ID varchar(100), LEVEL varchar(100),
LOGGER
varchar(100), MESSAGE varchar(100) );
```

## ResourceBundle Class [1-2]

- An object of the `ResourceBundle` class represents locale-specific information.

**Example**

A String, the program loads it from the `ResourceBundle` based on the current locale of the user.

- The `PropertyResourceBundle` and `ListResourceBundle` classes extend `ResourceBundle`.
- The `PropertyResourceBundle` is a concrete class to represent locale-specific information stored as key-value pairs in properties file.
- The `ListResourceBundle` is an abstract class to represent locale-specific information stored in list-based collections.

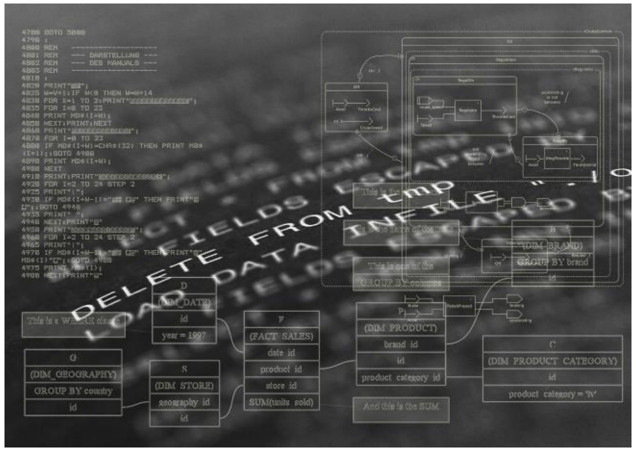
## ResourceBundle Class [2-2]

- Following table lists the key methods available in the `ResourceBundle` class:

| Method | Description |
|---|---|
| `getBundle()` | Returns a `ResourceBundle` object for the default locale. Overloaded version of this method accepts a Locale object to return a `ResourceBundle` object for the specified locale. |
| `getLocale()` | Returns the current locale of the user. |
| `getObject(String key)` | Returns the object for the corresponding key from the resource bundle. |
| `clearCache()` | Clears the cache of all resource bundles loaded by the class loader. |
| `containsKey(String key)` | Checks whether or not the specified key exists in the resource bundle. |
| `getKeys()` | Returns an `Enumeration` of all keys in the resource bundle. |
| `keySet()` | Returns a Set of all keys in the `ResourceBundle`. |

## Javadoc Tool

- The Javadoc tool:
  - Enables creating HTML-based API documentation of Java source code
  - Relies on documentation tags present in the source code
  - Is used to create documentation of packages, classes, interfaces, methods, and fields



## Documentation in Java

- Java API is a large collection of types where each type can have a large number of constructs, such as constructors, fields, and methods.
- Developers must:

Know the purpose of the types and its constructs

Provide documentation of the classes and their constructs

Describe everything that another developer would require to use the API

- In Java:
    - API documentation is created using documentation tags
- The Javadoc tool:

Generates API documentation in HTML

Can be found in the bin folder of the JDK path

Examples: Eclipse and NetBeans also have support for Javadoc

## Javadoc Tags [1-9]

- The Javadoc tags can be primarily divided into:

Class-level Tags

Method-level Tags

### Class-level Tags

| Tag | Description |
|---|---|
| `@author` | Inserts the author name of the class |
| `{@code}` | Inserts text in code format |
| `@since` | Inserts a Since heading used to specify from when this class exists |
| `@deprecated` | Inserts a comment to indicate that this class is deprecated and should no longer be used |

### Code Snippet

```
  * @author Carl Boynton
  * @author Andy Payne
  * @see Collection
  * @see Vector
  * @since JDK1.0
  */
public class MathDemo {
  /*Code implementation*/
}
```

The class `MathDemo` will have information indicating who are the authors of the code, which classes to see further, and since which version the class has been existing.

# Javadoc Tags [2-9]

## Method-level Tags

| Tag | Description |
|---|---|
| @param | Inserts a parameter that the method accepts |
| @return | Inserts the return type of the method |
| @throws | Inserts any exception that the method throws |
| @see | Inserts a See Also heading with a link or text points to closely related methods |
| @since | Inserts a Since heading with a text to specify from when this class exists |
| @deprecated | Inserts a comment to indicate that this method is deprecated, and should no longer be used |

## Code Snippet

```
/**
  @param num1 This is the first paramter to addInt method
  * @param num2 This is the second parameter to addInt method
  @return int This returns sum of numA and numB.
  @see MathDemo#addLong(long,long)
  */
public int addInt(int num1, int num2) {
       return num1 + num2;
}
```

The `@see` annotation in the code specifies an `addLong(long, long)` method in the `MathDemo` class. This method must be defined in the `MathDemo` class failing which the Javadoc tool will report a compilation error.

## Javadoc Tags [3-9]

- Following code snippet demonstrates the use of Javadoc tags and documentation comments:

Code Snippet

```
/**
  * The {@code MathDemo} class implements a calculation algorithm to add two integers.
  * @author Carl Boynton
  * @author Andy Payne
  * @see Math
  * @since JDK8.0
  */
public class MathDemo {
    /**
     * Constructs a MathDemo instance.
     */
public MathDemo() { }
     public long addLong(long num1, long num2) {
         return num1 + num2;
     }
     /**
          * This method is used to add two integers.
          * @param num1 This is the first parameter to addInt method
          * @param num2 This is the second parameter to addInt method
          * @return int This returns sum of numA and numB.
          */
     public int addInt(int num1, int num2) {
         return num1 + num2;}
```

## Javadoc Tags [4-9]

```
/**
  * This is the main method to use addInt method.
  * @param args Unused.
  * @exception java.io.IOException on input error.
  * @see java.io.IOException
*/
  public static void main(String[] args) throws
  java.io.IOException {
      MathDemo mathDemo = new MathDemo();
      System.out.println(mathDemo.addInt(5, 8));
 }
}
```

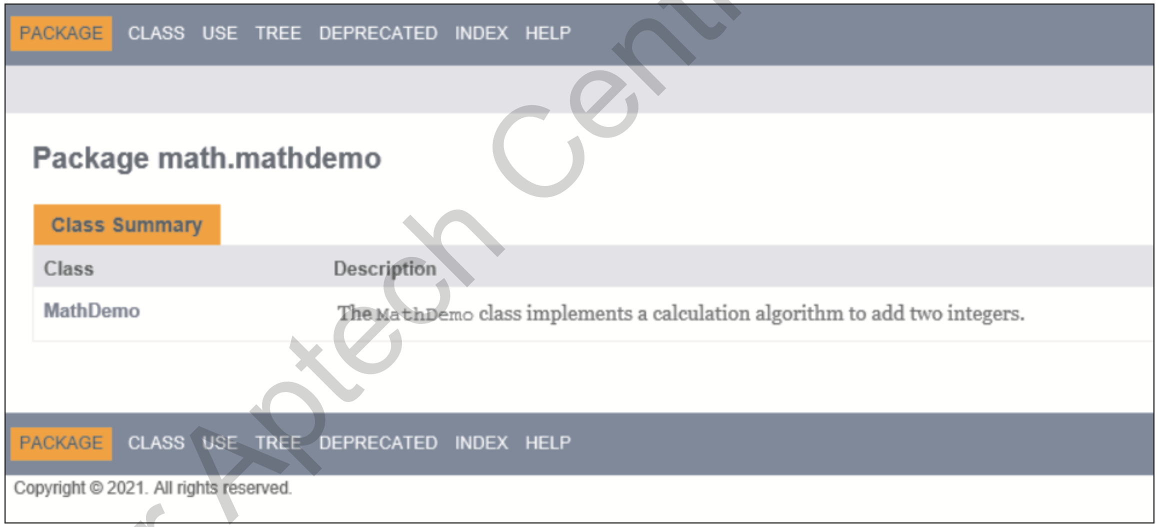
- The two ways to generate Java documentation for this code are:

At the command prompt using the Javadoc tool or using the IDE option.

Generating Javadoc is to use the Javadoc generation features of an IDE, such as NetBeans.

# Javadoc Tags [5-9]

- The command `javadoc MathDemo.java` given at the command prompt results in an HTML file containing the Javadoc generated documentation.
- Following figure displays the Javadoc generated documentation opened in the browser:


PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

Package math.mathdemo

Class Summary

| Class | Description |
|---|---|
| MathDemo | The `MathDemo` class implements a calculation algorithm to add two integers. |

PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

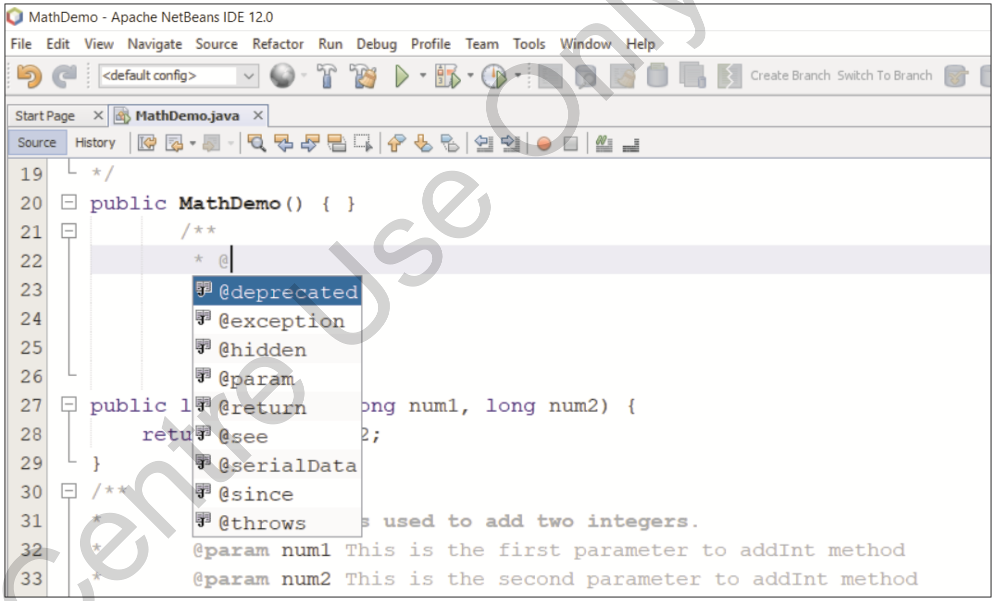
Copyright © 2021. All rights reserved.

## Javadoc Tags [6-9]

- NetBeans:
  - Enables automatically inserting Javadoc comments and tags, generating Javadoc, and viewing Javadoc documentation
  - Assists in writing Javadoc through hints and the code-completion feature
  - Can be automatically generated in source files
- For a Javadoc comment, typing /** and pressing the TAB and ENTER key:
  - Automatically generates a Javadoc comment block
- For a method, typing /** and pressing the TAB and ENTER key:
  - `@param` and `@return` tags
- For other tags:
  - A hint appears as a pop up as a Javadoc tag is typed
- On clicking a tag or pressing the Enter key:
  - The tag is inserted in the source file

# Javadoc Tags [7-9]

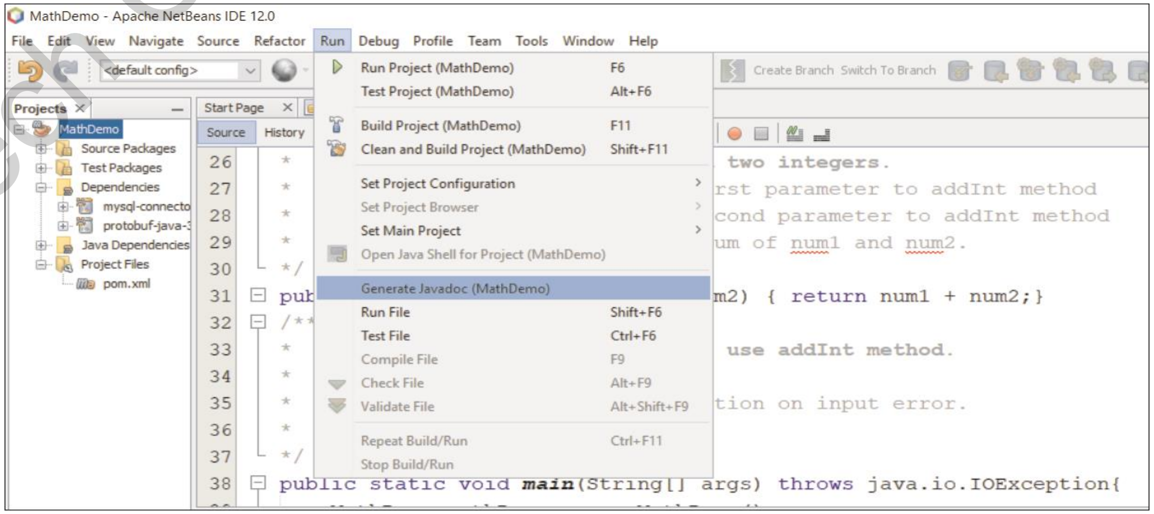
- Following figure displays the Javadoc tag code-completion pop-up:


MathDemo - Apache NetBeans IDE 12.0
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
<default config>
Create Branch Switch To Branch
Start Page
MathDemo.java
Source History
@deprecated
@exception
@hidden
@param
@return
@see
@serialData
@since
@throws

```
 */
public MathDemo() { }
    /**
     * @

public l          ong num1, long num2) {
    retu          2;
}
/**
 *                s used to add two integers.
 *      @param num1 This is the first parameter to addInt method
 *      @param num2 This is the second parameter to addInt method
```

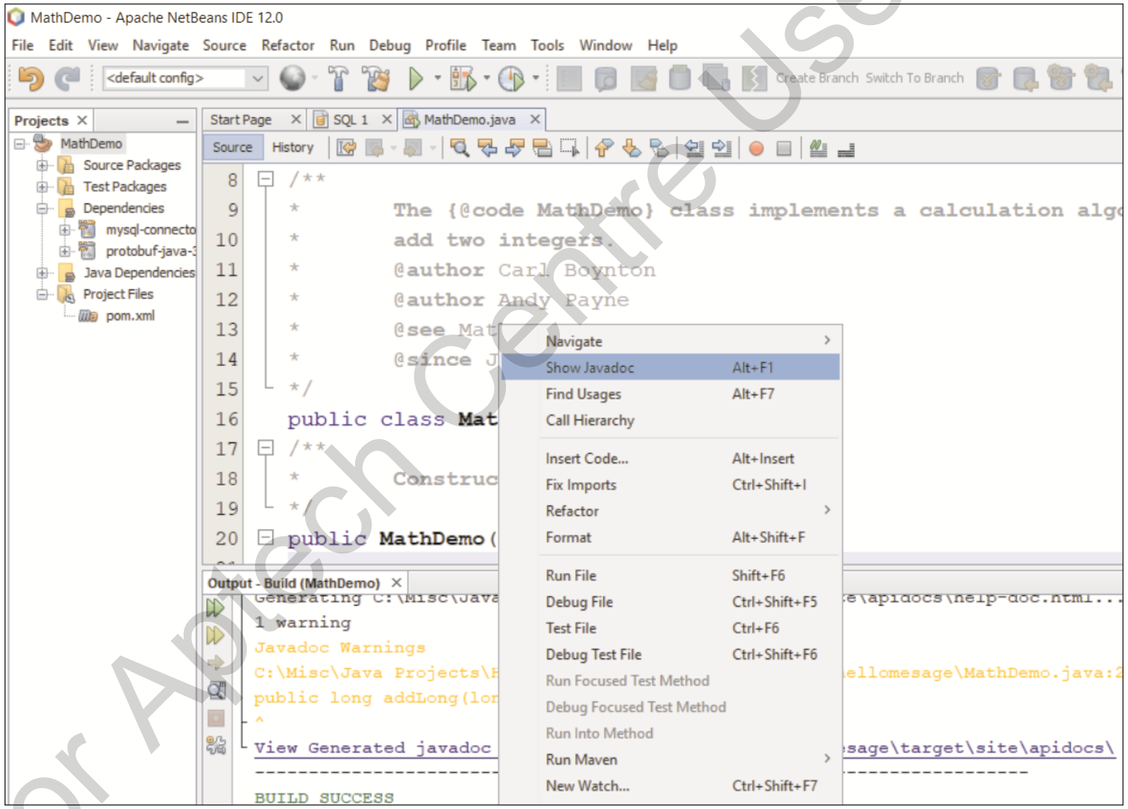
- Select **Run** → **Generate Javadoc** from the main menu of NetBeans to show the following figure that displays the Javadoc Documentation Generation in NetBeans:


MathDemo - Apache NetBeans IDE 12.0
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
<default config>
Create Branch Switch To Branch
Projects
MathDemo
Source Packages
Test Packages
Dependencies
mysql-connecto
protobuf-java-3
Java Dependencies
Project Files
pom.xml
Start Page
Source History
Run Project (MathDemo) F6
Test Project (MathDemo) Alt+F6
Build Project (MathDemo) F11
Clean and Build Project (MathDemo) Shift+F11
Set Project Configuration
Set Project Browser
Set Main Project
Open Java Shell for Project (MathDemo)
Generate Javadoc (MathDemo)
Run File Shift+F6
Test File Ctrl+F6
Compile File F9
Check File Alt+F9
Validate File Alt+Shift+F9
Repeat Build/Run Ctrl+F11
Stop Build/Run

```
 *    two integers.
 *    rst parameter to addInt method
 *    cond parameter to addInt method
 *    um of num1 and num2.
 */
pub   m2) { return num1 + num2;}
/**
 *    use addInt method.
 *
 *
 *    tion on input error.
 *
 */
public static void main(String[] args) throws java.io.IOException{
```

# Javadoc Tags [8-9]

- Select **Show Javadoc** from the contextual menu to show the following figure that displays Viewing Javadoc in NetBeans:

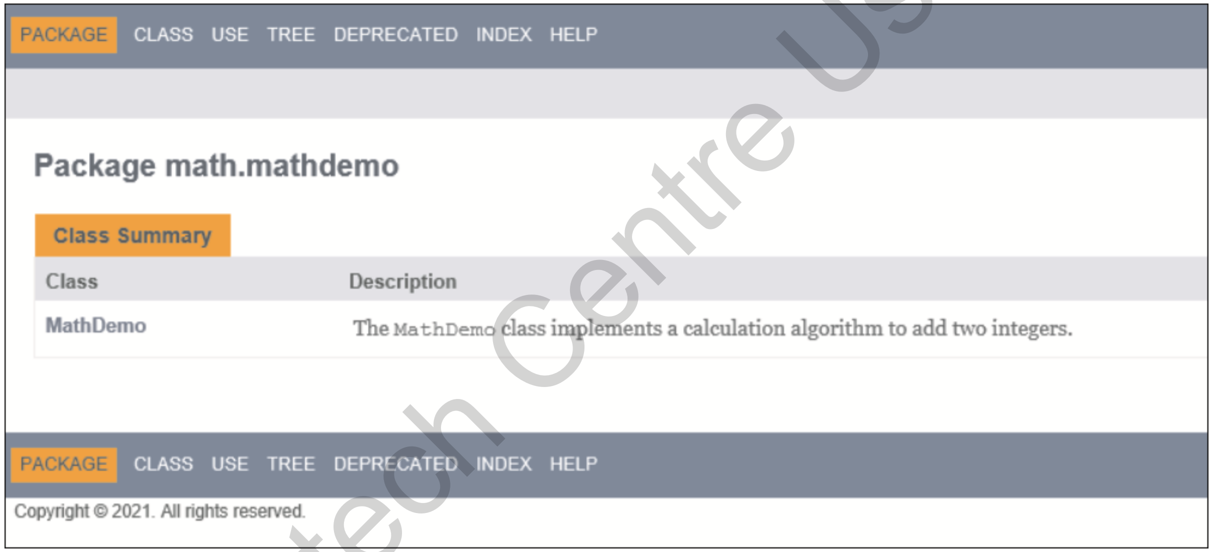


```
/**
 *        The {@code MathDemo} class implements a calculation alg
 *        add two integers.
 *        @author Carl Boynton
 *        @author Andy Payne
 *        @see Mat
 *        @since J
 */
public class Mat
/**
 *        Construc
 */
public MathDemo(
```

```
Generating C:\Misc\Java                                  e\apidocs\help-doc.html...
1 warning
Javadoc Warnings
C:\Misc\Java Projects\H                                  ellomesage\MathDemo.java:2
public long addLong(lon
^
View Generated javadoc                                   sage\target\site\apidocs\
------------------------                                 ------------------
BUILD SUCCESS
```

## Javadoc Tags [9-9]

- Following figure displays the Javadoc generated documentation in the browser:


PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

**Package math.mathdemo**

**Class Summary**

| Class | Description |
|---|---|
| MathDemo | The `MathDemo` class implements a calculation algorithm to add two integers. |

PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

Copyright © 2021. All rights reserved.

## java.net Package

- The `java.net` package:
  - Contains classes and interfaces for network programming
  - Creates transport layer client and server sockets
  - Performs communication over the Internet

- Some of the important classes of `java.net` package are:
  - DatagramPacket
  - DatagramSocket
  - Socket
  - ServerSocket
  - InetAddress
  - URL
  - URLConnection

## DatagramPacket Class [1-2]

- The `DatagramPacket` class:

  - Represents datagram packets for communication using the User Datagram Protocol (UDP) protocol
  - Contains information about itself and its destination
  - Are routed differently

**Note**: `DatagramPacket` object can have a maximum size of 65507 bytes.

## DatagramPacket Class [2-2]

- Following table demonstrates methods of the `DatagramPacket` class:

| Method | Description |
|---|---|
| `setData(byte[] buf)` | Sets the data of a packet as a `byte[]` |
| `setAddress(InetAddress iaddr)` | Sets the IP address of the computer to which a datagram packet must be sent |
| `setLength(int length)` | Sets the length of a packet as an int value |
| `getData()` | Returns the data of the packet as a `byte[]` |
| `getLength()` | Returns the length of data in a packet to be sent or in a packet that has been received |
| `getAddress()` | Returns the IP address of the computer to which a datagram packet is sent or the computer that sends a datagram packet |

## DatagramSocket Class

- The `DatagramSocket` class is responsible for sending and receiving datagram packets as `DatagramPacket` objects.
- Following table demonstrates methods of `DatagramPacket` class:

| Method | Description |
|---|---|
| `connect(InetAddress address, int port)` | Connects the socket to the IP address and port of a remote computer |
| `disconnect()` | Disconnects the socket |
| `send(DatagramPacket packet)` | Sends a `DatagramPacket` object to a destination |
| `receive(DatagramPacket packet)` | Receives a `DatagramPacket` object |

## Socket Class [1-2]

- The `Socket` class:

Represents the socket used by both the client and server for communicating

Is used for communication over the Transmission Control Protocol (TCP) protocol

- The TCP protocol:

Maintains a connection between endpoints that Socket objects represents

Guarantees both because both the client and server sockets remains connected

- To transmit data to a server:

A client creates an object of the `Socket` class.

The server obtains a `Socket` object by calling the `accept()` method of the `ServerSocket` class.

- A client can create a `Socket` to represent a connection to the server by:

Invoking `public Socket(String host,int port)` constructor of the `Socket` class

## Socket Class [2-2]

- Following table explains key methods of the `Socket` class:

| Method | Description |
|---|---|
| `connect(SocketAddress host, int timeout)` | Connects the client socket to the server socket. This method is required if a Socket object is created without initializing it with a connection to the server. |
| `getInputStream()` | Returns an `InputStream` object of the `Socket`. Both clients and servers use the `getInputStream()` method to receive data. |
| `getOutputStream()` | Returns an `OutputStream` object of the Socket. Both clients and servers use the `getOutputStream()` method to send data. |
| `close()` | Closes the Socket connection. |

## ServerSocket Class

- The `ServerSocket` class is used by servers to listen for incoming connections from clients.
- Following table explains key methods of `Socket` class:

| Method | Description |
|---|---|
| `bind(SocketAddress endPoint)` | Binds a `ServerSocket` object to a specified IP address and port number that the `SocketAddress` parameter represents. |
| `accept()` | Listens for a connection to be made to this socket and accepts it. The `accept()` method blocks until either a client connects to the server on the specified port or the socket times out. |
| `getLocalPort()` | Returns the port number as an int value that a `ServerSocket` object is listening to. |
| `setSoTimeout(int timeout)` | Sets a timeout in milliseconds after which a `ServerSocket` object stops accepting client connections. |
| `isClosed()` | Returns a boolean value to indicate whether or not a `ServerSocket` object is closed. |

## InetAddress Class

- The `InetAddress` class represents an Internet address to perform a Domain Name System (DNS) look-up and reverse look-up.
- Following table explains important methods of `InetAddress` class:

| Method | Description |
|---|---|
| `getAddress()` | Returns the IP address of the `InetAddress` object as a `byte[]` |
| `getByName(String host)` | Returns the IP address of the host passed as parameter as an `InetAddress` object |
| `getHostName()` | Returns the host name of the `InetAddress` object |
| `getAllByName(String host)` | Returns an array of its IP addresses for the host passed as parameter |
| `isReachable(int timeout)` | Returns a boolean to indicate whether or not the IP address represented by `InetAddress` is reachable |

# Inet4Address Class

- `Inet4Adress` class extends `InetAddress` class and represents an IPv4 address. In addition, it provides methods to interpret and display helpful information about IP addresses.
- Following table explains the important methods of the `Inet4Address` class:

| Method | Description |
|---|---|
| `equals(Object obj)` | Compares the current object against the specified object. |
| `getAddress()` | Returns the raw IP address of `Inet4Address` object. |
| `getHostAddress()` | Returns the IP address string in the textual presentation form. |
| `hashCode()` | Returns a `hashcode` for this IP address. |
| `isAnyLocalAddress()` | Checks if the `Inet4Address` is a wildcard address. |
| `isLinkLocalAddress()` | Checks if the `Inet4Address` is a link-local address. |
| `isLoopbackAddress()` | Checks if the `Inet4Address` is a loopback address. |
| `isMCGlobal()` | Checks if the multicast address has a global scope. |
| `isMCLinkLocal()` | Checks if the multicast address has a link scope. |
| `isMCNodeLocal()` | Checks if the multicast address has node scope. |
| `isMCOrgLocal()` | Checks if the multicast address has organization scope. |
| `isMCSiteLocal()` | Checks if the multicast address has site scope. |
| `isMulticastAddress()` | Checks if the `InetAddress` is an IP multicast address. |
| `isSiteLocalAddress()` | Checks if the `InetAddress` is a site-local address. |

## Inet6Address Class [1-2]

- Inet6Adress class extends InetAddress class and represents an IPv6 address.
- Following table explains important methods of Inet6Address class:

| Method | Description |
|---|---|
| `getByAddress(String host, byte[] addr, int scope_id)` | Used to create an Inet6Address object by setting an IPv6 scope id to the given value.<br>Parameters:<br>host: host<br>addr: raw IP address in network order<br>scope_id: scope id of the address |
| `getByAddress(String host, byte[] addr, NetworkInterface nif)` | Used to specify the network interface to be used with the address.<br>Parameters:<br>host: host<br>addr: raw IP address in network order<br>nif: network interface to be associated with this address |
| `getScopeId()` | Returns the scope id associated with this address or 0 if none set. |
| `getScopedInterface()` | Returns the network interface associated with this address or null if none set. |
| `getAddress()` | Returns raw IP address of this InetAddress object as an array. |
| `getHostAddress()` | Returns IP address in textual form. |
| `isAnyLocalAddress()` | Returns true if this address represents a local address. |
| `isLinkLocalAddress()` | Returns true if this address is a link local address. |
| `isLoopbackAddress()` | Returns true if this address is a loopback address. |
| `isMCGlobal()` | Returns true if this multicast address has global scope. |
| `isMCLinkLocal()` | Returns true if this multicast address has link scope. |

## Inet6Address Class [2-2]

| Method | Description |
|---|---|
| `isMCNodeLocal()`<br>`isMCOrgLocal()`<br>`isMCSiteLocal()` | Returns true if this multicast address has node scope.<br>Returns true if this multicast address has organization scope.<br>Returns true if this multicast address has site scope. |
| `isMulticastAddress()` | Returns true if this address is an IP multicast address. |
| `isSiteLocalAddress()` | Returns true if this address is a site local address. |
| `hashCode()` | Returns the `hashcode` associated with this address object. |
| `equals()` | Returns true if this IP address is same as that of the object specified.<br>Parameters:<br>obj: object to compare with |

## URL Class

- The `URL` class represents a Uniform Resource Locator (URL) that points to a resource on the Web.
- Following table explains the important methods of the `URL` class:

| Method | Description |
|---|---|
| `getPath()` | Returns the path of the URL as a String |
| `getQuery()` | Returns the query part of the URL as a String |
| `getPort()` | Returns the port of the URL as an int value |
| `getDefaultPort()` | Returns the default port for the protocol of the URL as an int value |
| `getProtocol()` | Returns the protocol of the URL as a String |
| `getHost()` | Returns the host of the URL as a String |
| `getFile()` | Returns the filename of the URL as a String |
| `openConnection()` | Opens a connection to the URL and returns a `URLConnection` object |

# URLConnection Class

- The `openConnection()` method of the URL class returns an implementation of the `URLConnection` class.
- Following table explains the important methods of the `URLConnection` class:

| Method | Description |
|---|---|
| `getURL()` | Returns the URL that the `URLConnection` object is connected to as a URL object |
| `setDoInput(boolean input)` | Accepts a boolean value to indicate whether the `URLConnection` object will be used for input. The default value is true |
| `setDoOutput(boolean output)` | Accepts a boolean value to indicate whether the `URLConnection` object will be used for output. The default value is false |
| `getInputStream()` | Returns the input stream of the `URLConnection` as an `InputStream` object. This method is called to read from a URL |
| `getOutputStream()` | Returns the output stream of the `URLConnection` as a `OutputStream` object. This method is called to write to a URL |
| `getContent()` | Returns an Object of the contents of the `URLConnection` |
| `getContentEncoding()` | Returns the content-encoding header field of the of the `URLConnection` as a String object |
| `getContentLength()` | Returns the content-length header field of the of the `URLConnection` as an int value |
| `getContentType()` | Returns the content-type header field of the of the `URLConnection` as a String object |
| `getLastModified()` | Returns the last-modified header field as an int value |
| `getExpiration()` | Returns the expires header field as a long value |
| `getIfModifiedSince()` | Returns the `ifModifiedSince` field of the `URLConnection` object as a long value |

## URLPermission Class

- The URLPermission class is used to represent permission to access the resources of given URL. The actions represent the request methods and headers; The given URL acts as the permission's name. For example,

```
"POST,GET:Header1,Header2"
```

- In this example, there are two requests methods and two headers separated by a colon. If the list of request methods and header is empty, then the separator colon is not necessary. In actions string, there is no permission for whitespace.

- Following table explains the important methods of the URLPermission class:

| Method | Description |
|---|---|
| `equals(Object p)` | Checks whether the two URLPermission objects are equal or not. |
| `getActions()` | Returns the actions in String format, which is currently a normalized method list and sends a request for header list. |
| `hashCode()` | Returns the hash value of this object. |
| `implies(Permission p)` | Checks whether the object implies the given permission or not. |

## HttpURLPermission Class

- HttpURLPermission is a class that represents permission to access a set of resources or a resource stated by a given https or http URL. A set of user-settable request methods and request headers are also represented by HttpURLPermission.
- URL string is the permission's name. Request methods and headers are actions string. Request methods and headers are actions string. The range of method and header names has no limitation in this class.

Syntax

```
public class HttpURLPermission extends java.security.Permission
```

- The URL

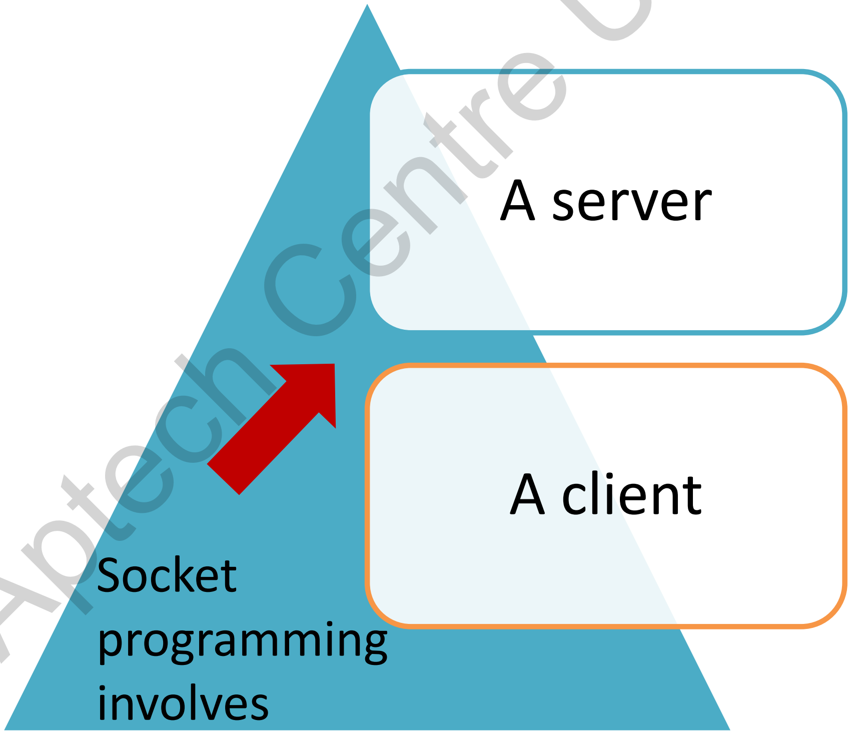
The path component of the URL is specified in a similar way to the path in FilePermission.

Following are three different ways with example URLs:

| http://www.oracle.com/a/b/c.html | http://www.oracle.com/a/b/* | http://www.oracle.com/a/b/- |
|---|---|---|
| URL that identifies a specific (single) resource. | The '*' character references all resources in the same 'directory' - means all the resources with the same number of path components, and which only differ in the final path component, represented by the '*'. | The '-' character refers to all resources recursively below the preceding path (For example, http://www.oracle.com/a/b/c/d/e.html matches this example). |

## Socket Programming and URL Processing

- The `openConnection()` method of the URL class returns an implementation of the `URLConnection` class.



## Client/Server Program [1-3]

- In a client/server programs that use TCP/IP:
  - A server is created to listen for client connections.
  - Then, a client is created to connect with the server and exchange data packages.
- Following are the steps to create a server class:

Step 1
- Create the server socket as a `ServerSocket` object

Step 2
- Wait for a client request

Step 3
- Create input and output streams to receive and send data respectively

Step 4
- Perform communication with the client

Step 5
- Close the socket

## Client/Server Program [2-3]

- Following Code Snippet demonstrates use of `ServerSocket` to create a server:

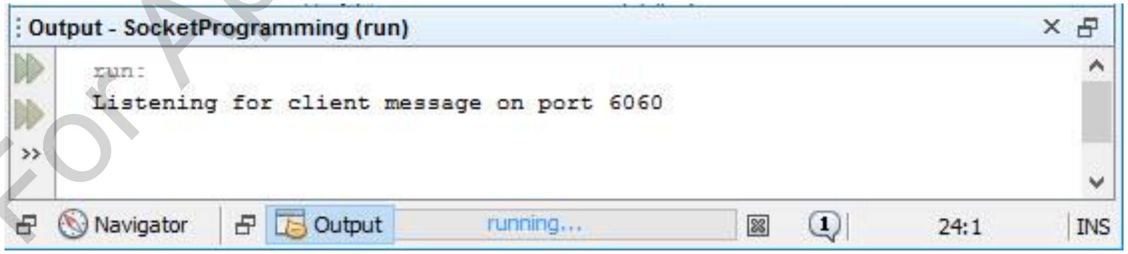
Code Snippet

```
package com.io.demo;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;
import java.net.SocketTimeoutException;
public class Server extends Thread {
   private ServerSocket serverSocket;
   public Server(int port) throws IOException {
      serverSocket = new ServerSocket(port);
  }
  public void run() {
      while (true) {
         try {
                  System.out.println("Listening for client message on port " +
                    serverSocket.getLocalPort());
                  Socket =  serverSocket.accept();
                  DataInputStream in = new DataInputStream(
                    socket.getInputStream());
                  DataOutputStream out = new
                      DataOutputStream(socket.getOutputStream());
                  out.writeUTF("Hello from server.");
              }
              catch (SocketTimeoutException sTException)    {
               sTException.printStackTrace();
              }
```

## Client/Server Program [3-3]

```
       catch (IOException ioException) {
              ioException.printStackTrace();
          } finally {
              try {
                  serverSocket.close();
              }
        catch (IOException ioException) {
                  ioException.printStackTrace();
              }
          }
     }
   }
   public static void main(String[] args) {
   try {
            Thread = new Server(6060);
            thread.start();
         }
          catch (IOException e) {
            e.printStackTrace();
          }
       }
}
```

**Output of the server**



```
run:
Listening for client message on port 6060
```

## Creating a Client [1-2]

- Following are the steps to create a client:

Step 1
- Create a socket as a Socket object

Step 2
- Create input and output streams to receive and send data respectively

Step 3
- Perform communication with the server

Step 4
- Close the socket

## Creating a Client [2-2]

- Following code snippet demonstrates the use of the `Socket` class to create a client:

Code Snippet

```
package com.io.demo;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.Socket;

public class Client {
    public static void main(String[] args) {
        try {
            Socket clientSocket = new Socket("localhost", 6060);
            InputStream inFromServer = clientSocket.getInputStream();
           System.out.println("Message received from server: " +
             in.readUTF());
            clientSocket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

## URL Processing [1-2]

- URL is an address of a resource in the Internet.
- Following code snippet demonstrates the use of the `URL` and `URLConnection` classes:

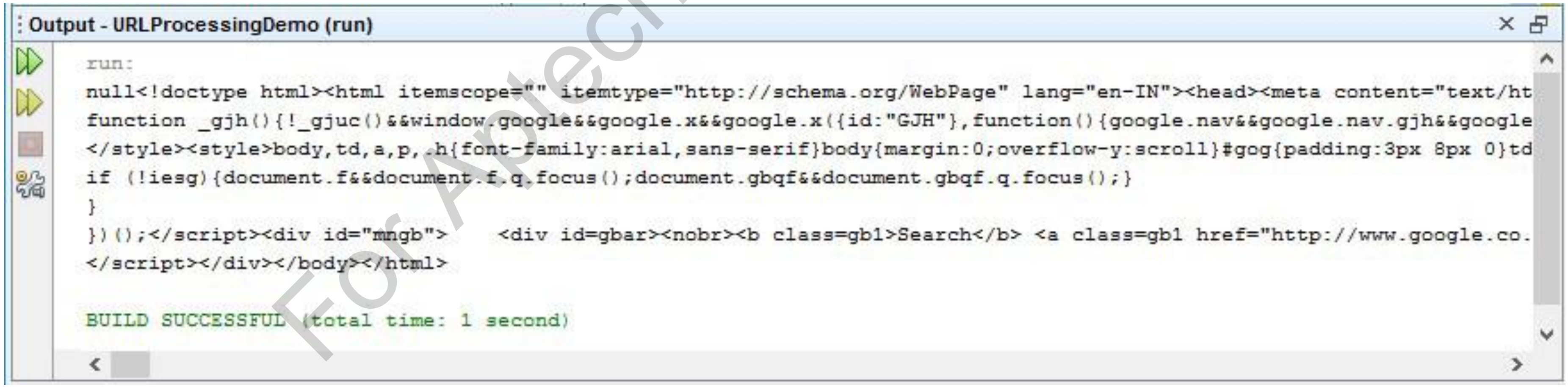
Code Snippet

```
package com.io.demo;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.Socket;
public class Client {
    public static void main(String[] args) {
        try {
            Socket clientSocket = new
            Socket("localhost", 6060);
            InputStream inFromServer =
              clientSocket.getInputStream();
```

## URL Processing [2-2]

```
                DataInputStream in = new
                              DataInputStream(inFromServer);
                System.out.println("Message received from
                server: " + in.readUTF());
                clientSocket.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
       }
}
```

- Following figure displays the output of the `URLProcessingDemo` class:



Output - URLProcessingDemo (run)

```
run:
null<!doctype html><html itemscope="" itemtype="http://schema.org/WebPage" lang="en-IN"><head><meta content="text/ht
function _gjh(){!_gjuc()&&window.google&&google.x&&google.x({id:"GJH"},function(){google.nav&&google.nav.gjh&&google
</style><style>body,td,a,p,.h{font-family:arial,sans-serif}body{margin:0;overflow-y:scroll}#gog{padding:3px 8px 0}td
if (!iesg){document.f&&document.f.q.focus();document.gbqf&&document.gbqf.q.focus();}
}
})();</script><div id="mngb">    <div id=gbar><nobr><b class=gb1>Search</b> <a class=gb1 href="http://www.google.co.
</script></div></body></html>

BUILD SUCCESSFUL (total time: 1 second)
```

## Summary

- The Log4J architecture is composed of loggers, appenders, and layouts.
- Properties and XML files are two most common approaches to specify Log4J configuration options.
- The file appender redirects logging data to a file.
- The JDBC appender redirects logging data to a database table.
- The `ResourceBundle` class enables creating localized programs based on user locales.
- The Javadoc tool relies on documentation tags present in the source code to create API documentation.
- Javadoc can be generated using the Javadoc tool from the command line or the in-built Javadoc options of NetBeans.
- Classes and interfaces of the `java.net` package supports network programming.
- Socket programming over UDP is supported by the `DatagramPacket` and `DatagramSocket` classes.
- Socket programming over TCP is supported by the `Socket` and `ServerSocket` classes of the `java.net` package.
- URL processing can be done by the `URL` and `URLConnection` classes.