

Professional Programming in Java

Session: 17

Java Logging API and ResourceBundle





- ◆ Describe the Log4J architecture
- ◆ Identify Log4J configuration options
- ◆ Explain the file appender
- ◆ Explain the JDBC appender
- ◆ Identify the ResourceBundle class



- ◆ 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, and Layouts





- ◆ 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:

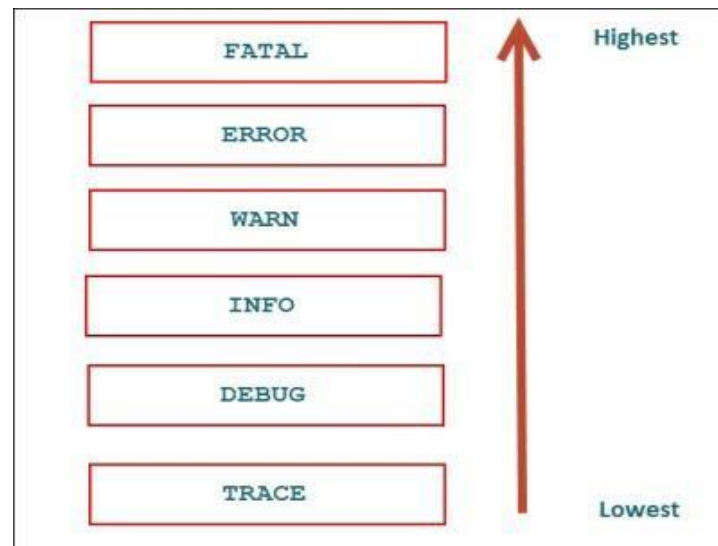
```
LoggerManager.getLogger(String loggerName)
```



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



- ◆ Loggers 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.



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.



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

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



- ◆ The steps to configure a NetBeans project to include the Log4J JARfiles are:

Step 1

- Download the Log4J binary file from the official Website, <https://logging.apache.org/log4j/2.x/download.html>.

Step 2

- Extract the compressed Log4J file into a suitable location.

Step 3

- Open NetBeans.

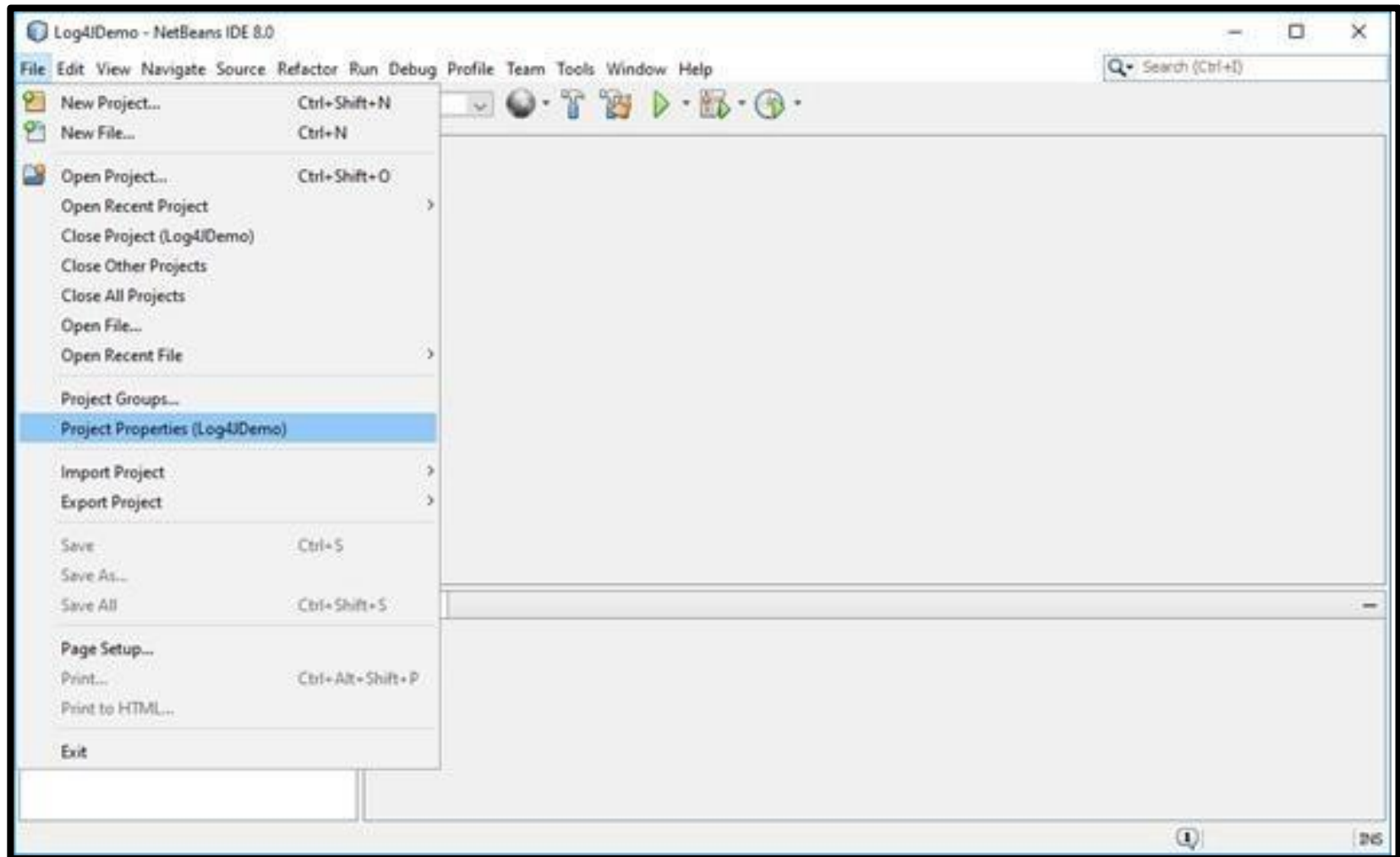
Step 4

- Create a **Log4JDemo** Java application project.

Step 5

- Select **Files** → **Project Properties (Log4JDemo)** from the main menu of NetBeans.

Project Configuration [2-5]

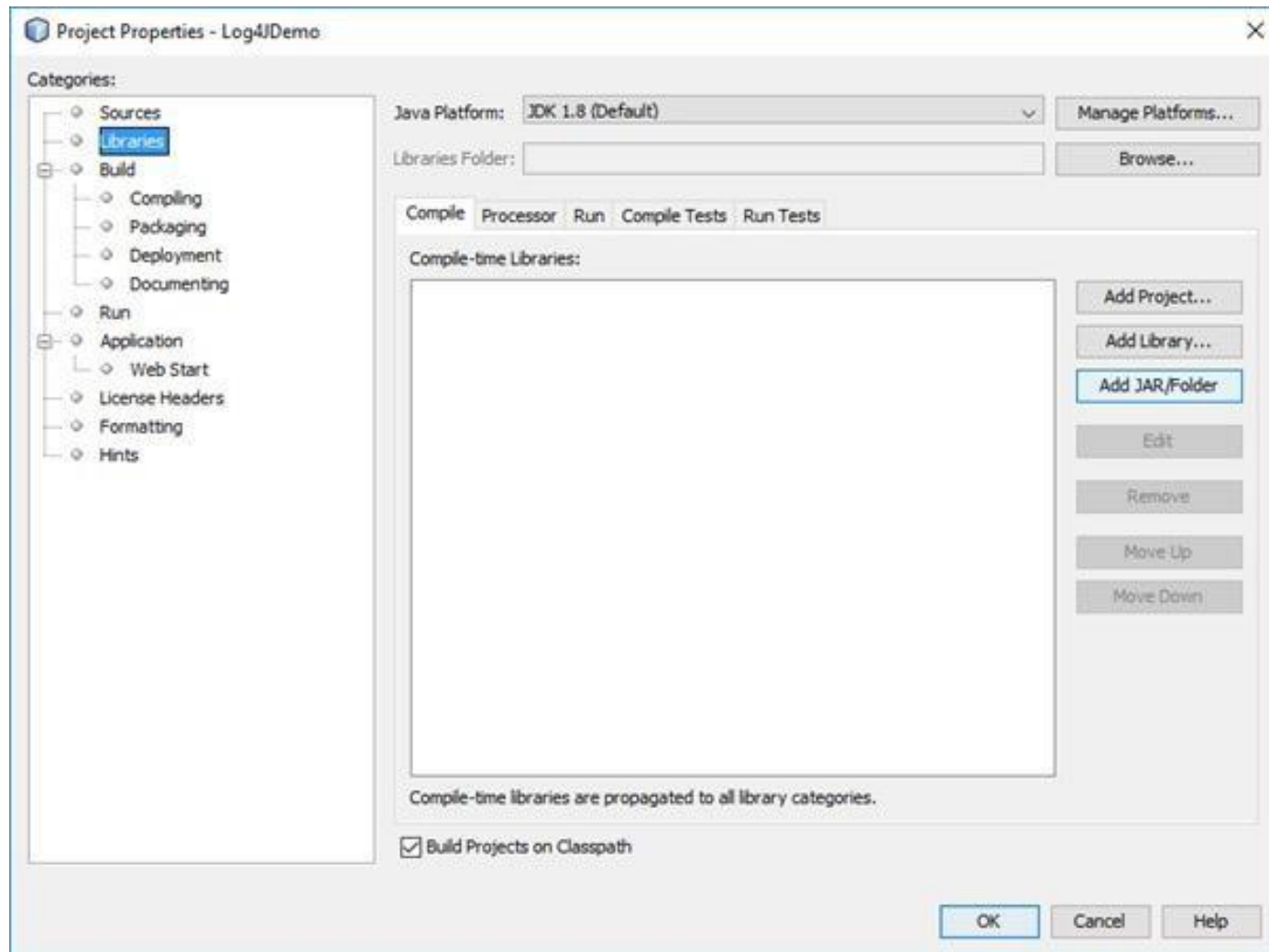


Project Configuration [3-5]



Step 6

- In the **Project Properties – Log4J Demo** dialog box, select **Libraries**, and then click **Add JAR/Folder**.

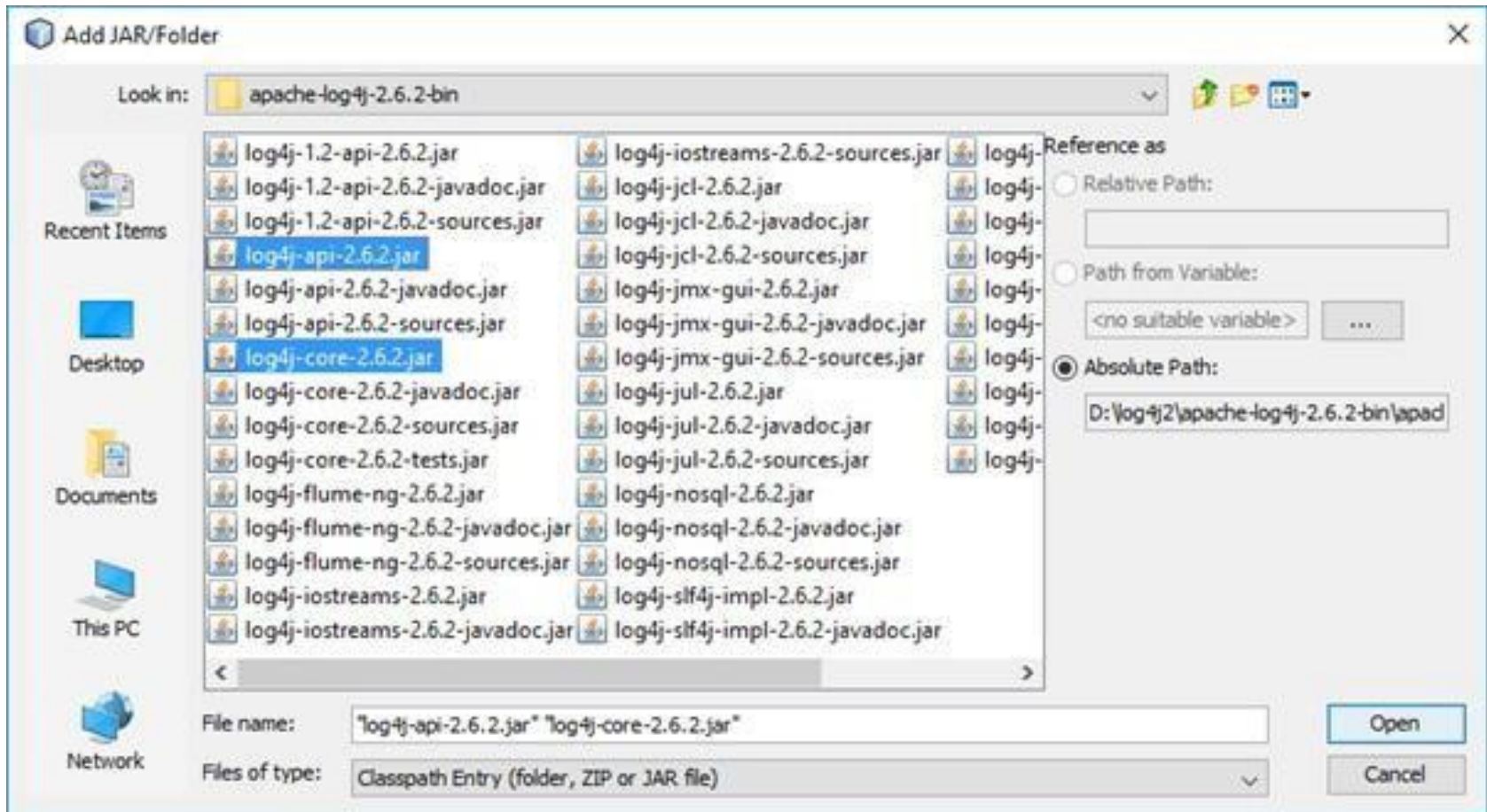


Project Configuration [4-5]



Step 7

- In the **Add JAR/Folder** dialog box, browse to downloaded Log4J directory, and select **log4j-api-2.x.x.jar** and **log4j-core-2.x.x.jar** files by pressing **Ctrl** button.





Step 8

- Click **Open**.

Step 9

- Click **OK** in the Project Properties – Log4J Demo dialog box. This adds the required Log4J JAR files to the project.



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

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



- ◆ Following code snippet demonstrates how a **LoggerDemo** 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 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");
    }
}
```

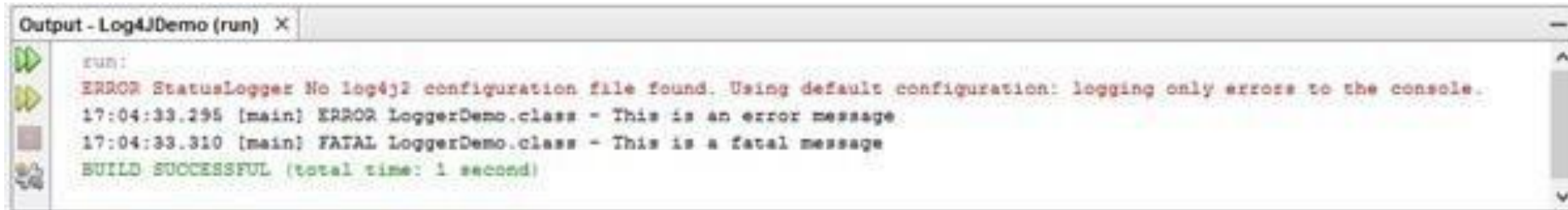



```
public static void main(String[] args){  
    LoggerDemo logger =new LoggerDemo();  
    logger.performLogging();  
}  
}
```

- ◆ The code calls the `getLogger()` method of `LogManager` passing the name of the class as parameter.
- ◆ The `getLogger()` method returns a `Logger` object for the class. The `performLogging()` method calls the log methods on the `Logger` object. The `main()` method calls the `performLogging()` method.



- ◆ Following figure displays the output of the **LoggerDemo** class:



```
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)
```

- ◆ The error message in the output is generated because no Log4J configuration file exists yet. As a result, Log4J uses the default configuration of the root logger. By default, root logger is configured with the **ERROR** log level. Therefore, only **ERROR** and **FATAL** messages got logged.



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

Code Snippet

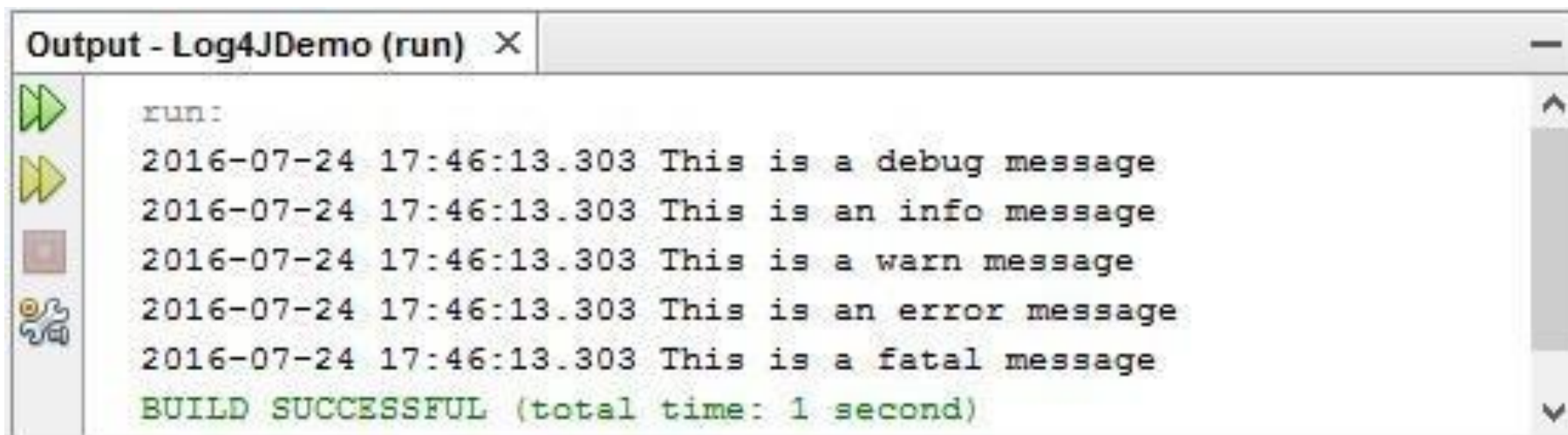
```
name = PropertiesConfig
appenders = consoleappender
appender.consoleappender.type = console
appender.consoleappender.name = STDOUT
appender.consoleappender.layout.type =
PatternLayout
appender.consoleappender.layout.pattern =
%d{yyyy-MM-dd HH:mm:ss.SSS} %msg%n
rootLogger.level = debug
rootLogger.appenderRefs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



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



```
run:
2016-07-24 17:46:13.303 This is a debug message
2016-07-24 17:46:13.303 This is an info message
2016-07-24 17:46:13.303 This is a warn message
2016-07-24 17:46:13.303 This is an error message
2016-07-24 17:46:13.303 This is a fatal message
BUILD SUCCESSFUL (total time: 1 second)
```



- 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>
```



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

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

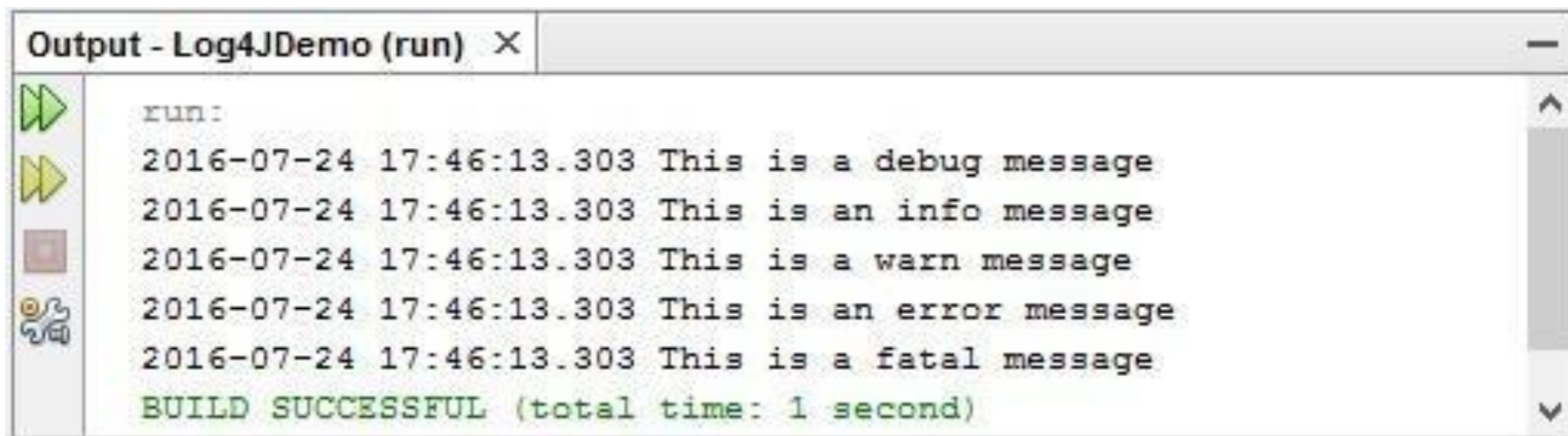
The `<PatternLayout>` element specifies the pattern layout to use with the appender and the `<pattern>` element specifies the formatting pattern to use.

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.



- ◆ Following figure displays the output on executing the LoggerDemo class:



```
run:
2016-07-24 17:46:13.303 This is a debug message
2016-07-24 17:46:13.303 This is an info message
2016-07-24 17:46:13.303 This is a warn message
2016-07-24 17:46:13.303 This is an error message
2016-07-24 17:46:13.303 This is a fatal message
BUILD SUCCESSFUL (total time: 1 second)
```



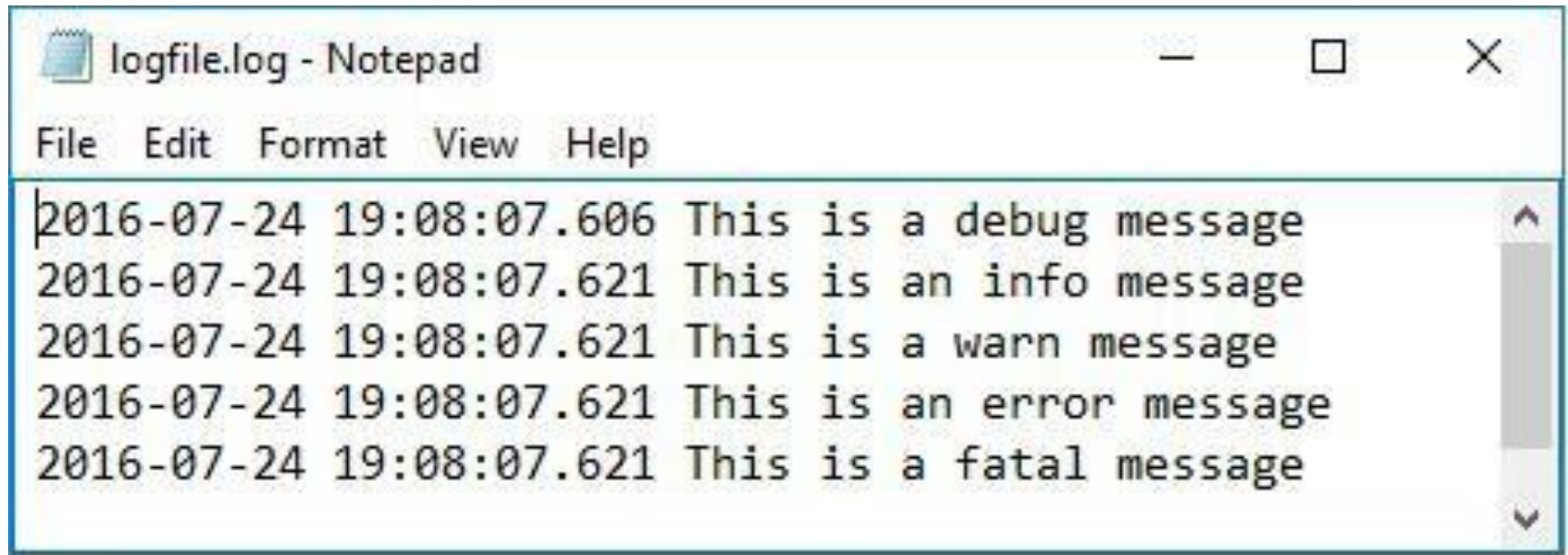

- ◆ Following code snippet demonstrates the use of a file appender:

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>
```



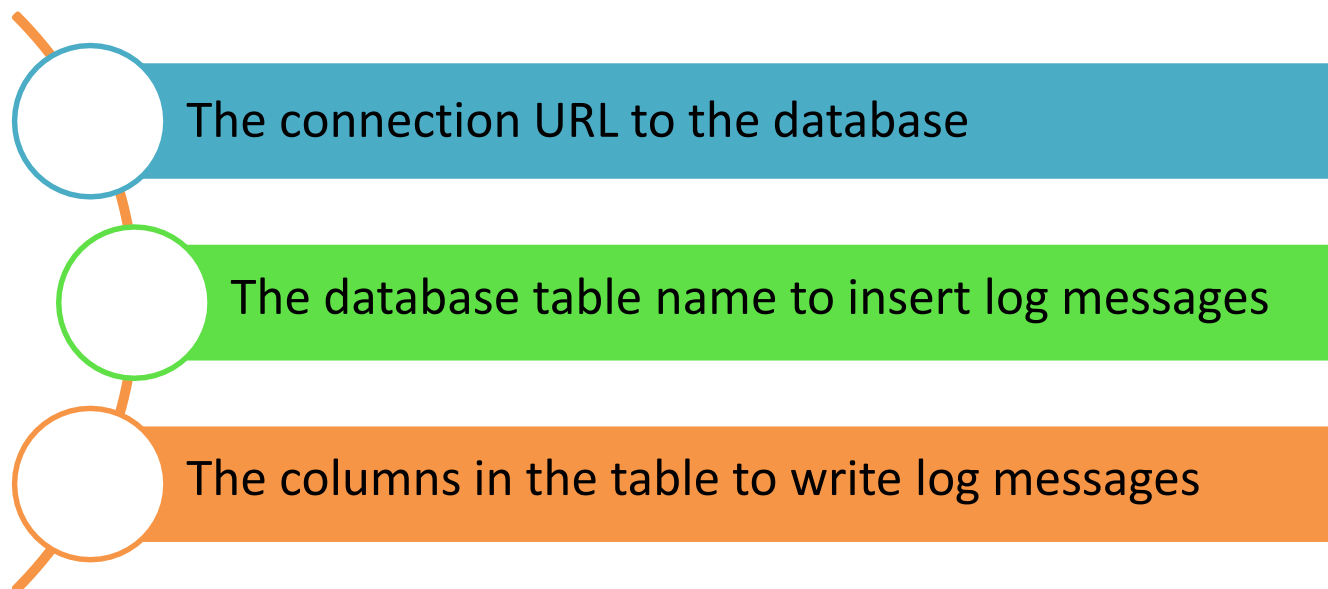
- ◆ Following figure displays the content of the `logfile.log` file:



```
logfile.log - Notepad
File Edit Format View Help
2016-07-24 19:08:07.606 This is a debug message
2016-07-24 19:08:07.621 This is an info message
2016-07-24 19:08:07.621 This is a warn message
2016-07-24 19:08:07.621 This is an error message
2016-07-24 19:08:07.621 This is a fatal message
```



- ◆ The <JDBC> element configures a JDBC appender.
- ◆ To use a JDBC appender, you need the following mandatory information:



- ◆ 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) ) ;
```



- ◆ To use the JDBC appender:

- ◆ The application needs a connection to the database

- ◆ To create a connection factory:

- ◆ Use the Apache commons-dbcp package

- ◆ This package relies on code in the commons-pool package to manage connection pool.

Note: The **commons-dbcp** package can be downloaded from https://commons.apache.org/proper/commons-dbcp/download_dbcp.cgi.



- ◆ Code snippet demonstrates MySqlConnectionFactory class that creates a connection to the LOG4JLOG database.

Code Snippet

```
package com.log4j.demo;
import java.sql.Connection;
import java.sql.SQLException;
import java.util.Properties;
import javax.sql.DataSource;
import org.apache.commons.dbcp.DriverManagerConnectionFactory;
import org.apache.commons.dbcp.PoolableConnection;
import org.apache.commons.dbcp.PoolableConnectionFactory;
import org.apache.commons.dbcp.PoolingDataSource;
import org.apache.commons.pool.impl.GenericObjectPool;
public class MySqlConnectionFactory {
    private static interface Singleton {
        final MySqlConnectionFactory INSTANCE = new
            MySqlConnectionFactory();
    }
    private final DataSource dataSource;
```



```
private MySqlConnectionFactory() {
    Properties properties = new Properties();
    properties.setProperty("user", "root");
    properties.setProperty("password", "root");
    GenericObjectPool pool = new GenericObjectPool();
    DriverManagerConnectionFactory connectionFactory = new
    DriverManagerConnectionFactory(
        "jdbc:mysql://127.0.0.1/log4jlog", properties);
    new PoolableConnectionFactory(connectionFactory, pool,
    null, "SELECT 1", 3, false, false,
        Connection.TRANSACTION_READ_COMMITTED);
    this.dataSource = new PoolingDataSource(pool);
}

public static Connection getDatabaseConnection() throws
SQLException {
    return Singleton.INSTANCE.dataSource.getConnection();
}}
```



- ◆ The code creates:

`AMySqlConnectionFactory` as a singleton.

- ◆ The class constructor uses:

`AProperties` object to set up the database user name and password credentials.

- ◆ The constructor then initializes:

`ADataSource` object from a `PoolableConnectionFactory` that it constructs.

- ◆ The `getDatabaseConnection()` static method:

Is responsible for returning a `Connection` object.



- ◆ Code snippet demonstrates the `log4j2.xml` file to configure the JDBC appender.

Code Snippet

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="error">
  <Appenders>
    <JDBC name="databaseAppender" tableName="LOG4JLOG.APP_LOG">
      <ConnectionFactory
        class="com.log4j.demo.MySqlConnectionFactory"
        method="getDatabaseConnection" />
      <Column name="EVENT_DATE" isEventTimestamp="true" />
      <Column name="LOG_LEVEL" pattern="%level" />
      <Column name="LOGGER" pattern="%logger" />
      <Column name="LOG_MESSAGE" pattern="%message" />
    </JDBC>
  </Appenders>
  <Loggers>
    <Root level="DEBUG">
      <AppenderRef ref="databaseAppender" />
    </Root>
  </Loggers>
</Configuration>
```



- ◆ The code uses the `<JDBC>` element to configure the `JDBCAppender`.
- ◆ The `name` and `tableName` attributes of the `<JDBC>` element specifies:
 - ◆ The appender name and the table name to which logging data will be inserted.
- ◆ The class and method attributes of the `<Appenders>` element specifies:
 - ◆ The connection factory class and the method that returns a connection.
- ◆ The `<Column>` maps:
 - ◆ The table columns with the logging data that the columns will hold.

The `<Loggers>` element associates the `JDBCAppender` with the root logger.



- ◆ Following figure demonstrates the output on executing the `LoggerDemo` class given in code snippet at the mysql prompt:

```
C:\windows\system32\cmd.exe - mysql -u root -p
mysql> use log4jlog;
Database changed
mysql> select * from app_log;
```

ID	EVENT_DATE	LOG_LEVEL	LOGGER	LOG_MESSAGE
1	2016-07-25 21:33:19.637	DEBUG	LoggerDemo.class	This is a debug message
2	2016-07-25 21:33:19.772	INFO	LoggerDemo.class	This is an info message
3	2016-07-25 21:33:19.817	WARN	LoggerDemo.class	This is a warn message
4	2016-07-25 21:33:19.916	ERROR	LoggerDemo.class	This is an error message
5	2016-07-25 21:33:19.989	FATAL	LoggerDemo.class	This is a fatal message

```
5 rows in set (0.01 sec)

mysql>
```



- ◆ 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.



- ◆ 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.



- ◆ 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.
- ◆ Java 8 provides equivalent binary versions of some functional interfaces that can accept two parameters.
- ◆ The JDBC appender redirects logging data to a database table.
- ◆ The ResourceBundle class enables creating localized programs based on user locales.