Logging

CS 240 – Advanced Programming Concepts

Java Logging

- Java has built-in support for logging
- Logs contain messages that provide information to
  - Software developers (e.g., debugging)
  - System administrators
  - Customer support agents
- Programs send log messages to “loggers”
  - There can be one or more
- Each message has a “level”
  - SEVERE, WARNING, INFO, CONFIG, FINE, FINER, FINEST

Java Logging

- Loggers have a method for each message level that are used to enter messages in the log
  - severe, warning, info, config, fine, finer, finest
- Rather than removing debugging log messages from the code, we leave them in
- Loggers can be configured to include or omit log messages based on their levels
  - Logger.setLevel(level) method
    - ALL (include all messages)
    - OFF (omit all messages)
    - SEVERE, WARNING, INFO, CONFIG, FINE, FINER, FINEST (include all messages at a particular level and higher)

Java Logging

- Each logger has one or more “handlers” associated with it
- Handlers represent destinations to which the log messages should be sent
  - ConsoleHandler (sends messages to the console)
  - FileHandler (sends messages to a file)
  - SocketHandler (sends messages to a network socket)
- Like loggers, handlers can also be configured to include or omit log messages based on their levels
  - Handler.setLevel(level) method
    - ALL (include all messages)
    - OFF (omit all messages)
    - SEVERE, WARNING, INFO, CONFIG, FINE, FINER, FINEST (include all messages at a particular level and higher)

Java Logging

- Each handler has a “formatter” which defines the format used to encode its messages
  - SimpleFormatter
  - XMLFormatter

Programmatic Configuration

```
public class ProgrammaticConfigurationExample {

    private static Logger logger;

    static {
        try {
            initLog();
        } catch (IOException e) {
            System.out.println("Could not initialize log: " + e.getMessage());
        }
    }

    private static void initLog() throws IOException {

        Level logLevel = Level.FINEST;

        logger = Logger.getLogger("ProgrammaticConfigurationExample");
        logger.setLevel(logLevel);
        logger.setUseParentHandlers(false);

        Handler consoleHandler = new ConsoleHandler();
        consoleHandler.setLevel(logLevel);
//        consoleHandler.setFormatter(new SimpleFormatter());
        logger.addHandler(consoleHandler);

        FileHandler fileHandler = new FileHandler("log.txt", false);
        fileHandler.setLevel(logLevel);
//        fileHandler.setFormatter(new SimpleFormatter());
        logger.addHandler(fileHandler);
    }
```

File Configuration Example

```
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 * Specify the configuration file by adding the following VM option
 * in the run configuration or you will get the default logging
 * configuration:
 *
 *      -Djava.util.logging.config.file=logging.properties
 *
 */
public class FileConfigurationExample {

    private static Logger logger =
        Logger.getLogger("FileConfigurationExample");

}
```

Logging Configuration File

```
handlers=java.util.logging.FileHandler, java.util.logging.ConsoleHandler
java.util.logging.ConsoleHandler.level=FINE
java.util.logging.ConsoleHandler.formatter=java.util.logging.SimpleFormatter

java.util.logging.FileHandler.level=FINE
java.util.logging.FileHandler.pattern=log.txt

# Write 10MB before rotating this file
java.util.logging.FileHandler.limit=10000000

# Number of rotating files to be used
java.util.logging.FileHandler.count=4
java.util.logging.FileHandler.formatter=java.util.logging.SimpleFormatter

# Log format to use
java.util.logging.SimpleFormatter.format=%1$tY-%1$tm-%1$td %1$tH:%1$tM:%1$tS
%4$s %2$s %5$s%6$s%n

.level=FINEST
```

Log Formatting

- Use format specifiers from the java.util.Formatter class
- Available variables to log:
  - %1$ The date/time the message was created
  - %2$ The method that called the log method
  - %3$ The name of the logger
  - %4$ The level the message was logged at
  - %5$ The message
  - %6$ The throwable

Logging Messages

- Logging messages with specific levels
  - severe(message)
  - Same for warning, info, config, fine, finer, finest
  - log(level, message)
- Logging method enter/exit
  - entering(className, methodName)
  - exiting(className, methodName)
  - Logged at FINER level
- Logging throwing an exception
  - throwing(className, methodName, throwable)
  - Logged at FINER level
- Logging catching an exception
  - log(level, message, throwable)

Logging Message

```
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 * Specify the configuration file by adding the following VM option in the run configuration
 * or you will get the default logging configuration:
 *
 *      -Djava.util.logging.config.file=logging.properties
 *
 */
public class FileConfigurationExample {

    private static Logger logger = Logger.getLogger("FileConfigurationExample");

    public static void main(String [] args) {
        logger.info("This is my info log message.");
        logger.fine("This is my fine log message");

        try {
            throw new Exception("An exception occurred");
        } catch (Exception ex) {
            logger.severe("This is my exception message");
            logger.log(Level.SEVERE, "This is my message logged with the exception", ex);
        }
    }
}
```

Logging Messages

```
import java.util.logging.*;

public class Server {

    private static Logger logger = Logger.getLogger("Server");

    private void run() {
        logger.info("Initializing HTTP Server");
        try {
            server = HttpServer.create(new InetSocketAddress(SERVER_PORT_NUMBER),
                                       MAX_WAITING_CONNECTIONS);
        } catch (IOException e) {
            logger.log(Level.SEVERE, e.getMessage(), e);
            return;
        }

        logger.info("Creating contexts");
        server.createContext("/games/list", new ListGamesHandler());
        server.createContext("/routes/claim", new ClaimRouteHandler());

        logger.info("Starting HTTP Server");
        server.start();
    }
}
```

Logging Messages

```
class ClaimRouteHandler implements HttpHandler {

    private static Logger = Logger.getLogger("ClaimRouteHandler");

    @Override
    public void handle(HttpExchange exchange) throws IOException {
        logger.entering("ClaimRouteHandler", "handle");

        try {
            ...
            logger.fine(reqData);
            ...
        }
        catch (IOException e) {
            logger.log(Level.SEVERE, e.getMessage(), e);
            ...
        }

        logger.exiting("ClaimRouteHandler", "handle");
    }
    ...
}
```

Log4J

- Java didn't have a logging API until version 1.4
  - Before 1.4, we had to log with System.out.println(…) or System.err.println(…)
  - Or, use an external framework
- Log4J became very popular and widely used before 1.4
- Log4J is still more commonly used than the built-in java.util.logging API
- Log4J is very similar to java.util.logging