

Debugging & Logging

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

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

public class Server {

    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("tickettoride");
        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);
    }
}
```

Initializing Logging

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 Messages

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

public class 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;
        }

        server.setExecutor(null); // use the default executor

        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;  
  
    static {  
        logger = Logger.getLogger("tickettoride");  
    }  
  
    @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");  
    }  
    ...  
}
```

Android Studio Debugger

- Running programs with and without debugger
- Running multiple programs at the same time (with or without debugger)
- Swapping between programs
- Breakpoints (step into, step over, return, continue)
- Variables (locals and parameters)
- Watches
- Runtime stack (frames)
- Stopping programs