De ce logging?

Avantaje:

- Debugging
- Auditing

Dezavantaje

- diminuarea vitezei de rulare
- dificultatea urmăririi output-ului

log4j

Un mecanism de logare pentru Java.

Se pot defini:

- Loggere
  - Pot fi ierarhizate
- Priorități mesajelor de logare
- Appendere
- Formate de logare
- Fișiere de configurare

Priorități ale mesajelor

1. FATAL
2. ERROR
3. WARN
4. INFO
5. DEBUG

Pragul de prioritate (log level) inhibă generarea mesajelor cu prioritate inferioară.

Ierarhizare

- Se poate construi o ierarhie de loggere
  - Se construiește prin simbolul **.**
  - Loggerul **idp.gui** părinte al loggerul **idp.gui.Gui**

- Se moștenesc:
  - toate appenderele de la toți strămoșii
  - pragul de prioritate de la părintele direct

Appendere

- Reprezintă destinația mesajului de logare
- Un logger poate poseda oricâți appenderi
- Pragul de logare stabilit cu Threshold
- **Tipuri**:
  - ConsoleAppender
  - FileAppender
  - RollingFileAppender
  - JDBCAppender
  - SocketAppender

Layout

- Reprezintă formatarea mesajului captat
- Se atașează unui appender

- **Tipuri**:
  - SimpleLayout: mesajul original
  - PatternLayout:
    - indicatori asemănători celor folosiți la printf

Exemplu log4j

```
import org.apache.log4j.*;
public class Test {
    static Logger logger = Logger.getLogger(“Test”);
    public static void main(String[] args) {
        BasicConfigurator.configure();
        logger.info("Hello");
    }
}
```

Exemplu log4j

```
import org.apache.log4j.*;  => IMPORT
public class Test {
    static Logger logger = Logger.getLogger("Test");
    public static void main(String[] args) {
        BasicConfigurator.configure();
        logger.info("Hello");
    }
}
```

Exemplu log4j

```
import org.apache.log4j.*;
public class Test {
    static Logger logger = Logger.getLogger(“Test”);
                                        =>instanță de logger
    public static void main(String[] args) {
        BasicConfigurator.configure();
        logger.info("Hello");
    }
}
```

Exemplu log4j

```
import org.apache.log4j.*;
public class Test {
    static Logger logger = Logger.getLogger("Test");
    public static void main(String[] args) {
        BasicConfigurator.configure();
                                    => configurare default
        logger.info("Hello");
    }
}
```

Exemplu log4j

```
import org.apache.log4j.*;
public class Test {
    static Logger logger = Logger.getLogger(“Test”);
    public static void main(String[] args) {
        BasicConfigurator.configure();
        logger.info("Hello"); => logare de informații
    }
}
```

Output

- **0 [main] INFO Test – Hello**

- Semnificație:
  - durata_de_la_inceputul_execuției [thread] prioritate nume - mesaj

Fișiere de configurare

- Folosite pentru modificarea comportamentului loggerelor fără a schimba codul sursă

- Permit specificarea:
  - Nivelelor de logare
  - Appenderelor folosite
  - Layouturile pentru fiecare appender

Diagnostic Contexts

- Ce se întâmplă în cazul claselor derivate din Thread?
  - Se loghează același mesaj, chiar dacă mesajele provin din thread-uri separate
- Soluție:
  - Definirea unor **identificatori** ai fiecărui fir
  - Utilizare MDC (mapped diagnostic context)
    - Funcționalitate de dicționar
    - Asocierile cheie-valoare se mențin la nivel de **fir**.

Utilizare MDC

```
MDC.put("name", "Deirdre“+id_thread_unic);
 // Se afișează cu %X{name}
logger.info("Hello");
MDC.remove("name");
```