

# Modulo II – Tópicos em Java – Logging

*Prof. Eduardo Bezerra  
e  
Prof. Ismael H F Santos*

## Ementa

- **Modulo II - Tópicos em JAVA**
  - Logging
  - Log4j
    - Componentes
      - Logger (classe)
      - Appender (interface)
      - Layout (classe)
    - Configuração
    - Resumo

## Bibliografia

- *Linguagem de Programação JAVA*
  - Ismael H. F. Santos, Apostila UniverCidade, 2002
- *The Java Tutorial: A practical guide for programmers*
  - Tutorial on-line: <http://java.sun.com/docs/books/tutorial>
- *Java in a Nutshell*
  - David Flanagan, O'Reilly & Associates
- *Just Java 2*
  - Mark C. Chan, Steven W. Griffith e Anthony F. Iasi, Makron Books.
- *Java 1.2*
  - Laura Lemay & Rogers Cadenhead, Editora Campos

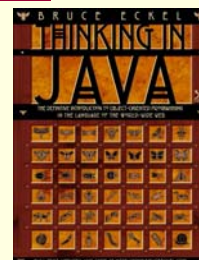
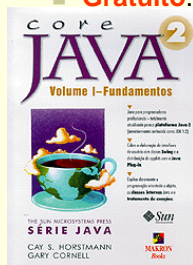
April 05

Prof. Ismael H. F. Santos - [ismael@tecgraf.puc-rio.br](mailto:ismael@tecgraf.puc-rio.br)

3

## Livros

- **Core Java 2**, Cay S. Horstmann, Gary Cornell
  - Volume 1 (Fundamentos)
  - Volume 2 (Características Avançadas)
- **Java: Como Programar**, Deitel & Deitel
- **Thinking in Patterns with JAVA**, Bruce Eckel
  - **Gratuito.** <http://www.mindview.net/Books/TIJ/>

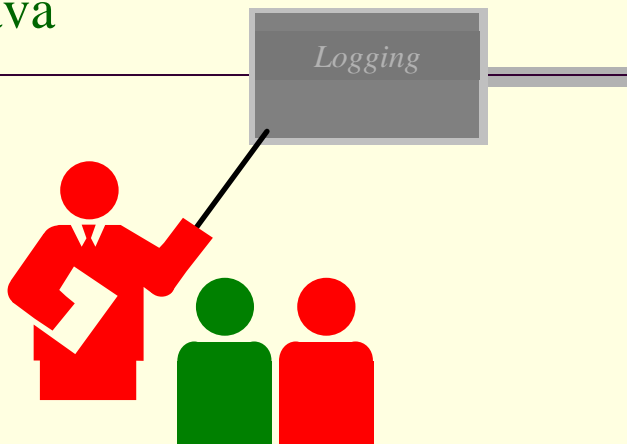


April 05

Prof. Ismael H. F. Santos - [ismael@tecgraf.puc-rio.br](mailto:ismael@tecgraf.puc-rio.br)

4

## POO-Java



April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

5

## Logging

- **Logging:** manutenção do registro do comportamento de uma aplicação.
- **Por que realizar logging?**
  - Mensagens de log podem ajudar na depuração da aplicação.
  - Mensagens de log fornecem o contexto da execução de uma aplicação

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

6

# Logging

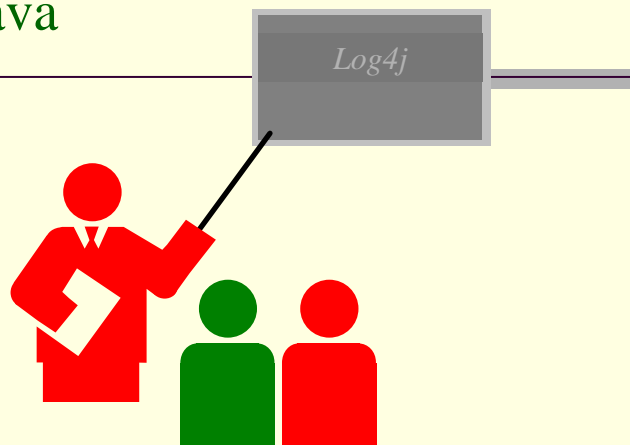
- Quando utilizar logging?
  - Na fase de desenvolvimento: para depurar a aplicação.
    - O que houver de errado? Onde ocorreu? Por que ocorreu?
  - Na fase de produção: ajuda a resolver problemas de execução.
- (Desvantagem) instruções de log têm o potencial de umentar o tamanho do executável e de reduzir a velocidade da aplicação.

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

7

# POO-Java



April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

8

## Log4j

---

- Log4j é um projeto de código aberto (*open source*) que permite ao desenvolvedor incluir logging em sua aplicação.
  - <http://logging.apache.org>
- Fornece serviços de logging para auditoria e depuração de aplicações.
- Com o Log4j, podemos ativar e desativar o logging sem a necessidade de modificar os binários da aplicação.
  - Isso pode ser controlado apenas editando um arquivo de configuração (detalhes mais adiante).

## Log4j

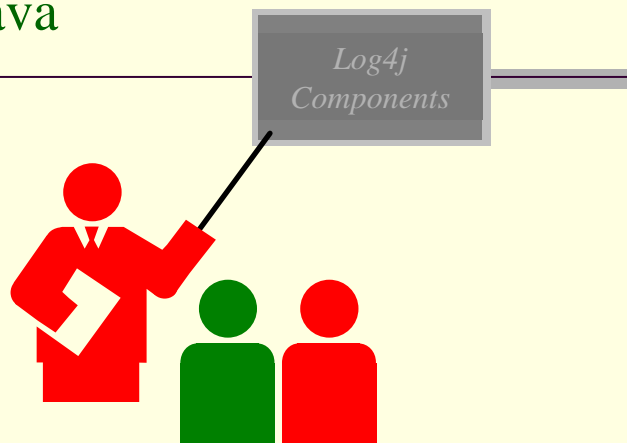
---

- **Permite o controle dinâmico de:**
  - Destino da saída de log (email, console, SGBD, arquivo do SO, servidores de log, etc.)
  - Que informação deve ser registrada
  - Formatação da saída (texto, HTML, XML, etc)
- **Foi implementado para outras linguagens além de Java:**
  - C, C++, C#, Perl, Python, Ruby e Eiffel

## Instalação do Log4j

- A versão atual do Log4J é a 1.2.8
  - <http://logging.apache.org/log4j/docs/download.html>
  - disponíveis o código fonte e as bibliotecas (.jar)
- Faça a descompactação dos arquivos para alguma pasta.
- Adicione a biblioteca (.jar) do log4J no classpath da aplicação.
  - Nome do arquivo da biblioteca: log4j-xxx.jar (onde xxx é a versão).

## POO-Java



## Componentes do Log4J

- A API do Log4j tem os seguintes componentes principais:
  - Logger (classe)
  - Appender (interface)
  - Layout (classe)
- Há também componentes acessórios (auxiliares):
  - Level
  - PropertyConfigurator, DOMConfigurator
- Vamos agora detalhar esses componentes...

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

13

## O componente Logger

- O ponto de entrada para o log4j é um objeto Logger.
- Criamos um objeto dessa classe e requisitamos a ele que faça o log de mensagens.
- Há diversos **métodos de log** (i.e., métodos que aceitam como argumento uma mensagem de log) existentes em Logger.
- Além disso, todo objeto da classe Logger possui duas propriedades importantes:
  - Seu **nível**
  - Seu **nome**

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

14

## Nível de um Logger

---

- O nível de um objeto logger determina que mensagens de log são realmente consideradas por este objeto.
  - Mensagens com nível igual ou mais alto que o nível definido para o logger são enviadas para a saída.
- Há 8 níveis, definidos como constantes (inteiras) na classe Level (também do log4j):
  - DEBUG < INFO < WARN < ERROR < FATAL
  - ALL (nível mais baixo) e OFF (nível mais alto)

## Nível de um Logger

---

- Um logger pode ser definido em um desses níveis
  - O método **setLevel** permite definir o nível do logger.



## Método setLevel - exemplo

- O Log4j registra uma mensagem sse o método de log utilizado corresponde a um nível que é igual ou maior que o nível do logger.

- Exemplo:

```
Logger logger = Logger.getLogger(MinhaClasse.class.getName());  
...  
logger.setLevel(Level.ERROR);  
...
```

## Métodos de log

- Método de log = método chamado quando desejamos registrar uma mensagem no log na aplicação.
- São os métodos de log existentes na classes Logger são: debug, info, warn, error e fatal.
- Cada um desses métodos está associado a um nível, correspondente ao seu nome.
  - e.g., **logger.info("Servidor Levantado")** é uma requisição de log no nível INFO.

## Métodos de log

- **Habilitação versus desabilitação**
  - Diz-se que uma mensagem de log está habilitada se o nível do método de log utilizado for maior ou igual ao nível do logger.
  - Do contrário, diz-se que a mensagem está desabilitada.

## Métodos de log

```
public void debug(Object message);  
public void debug(Object message , Throwable t);  
  
public void error(Object message);  
public void error(Object message , Throwable t);  
  
public void fatal(Object message);  
public void fatal(Object message , Throwable t);  
  
public void info(Object message);  
public void info(Object message , Throwable t);  
  
public void warn(Object message);  
public void warn(Object message, Throwable t);
```

## Métodos de log - exemplo

```
Logger logger = Logger.getLogger(MinhaClasse.class.getName());
...
logger.setLevel(Level.ERROR);
...
logger.info("Início...");           // Mensagem desabilitada
...
logger.error("Entrada inválida!"); // Mensagem habilitada
...
logger.fatal("Erro fatal. Abortando!"); // Mensagem habilitada
...
```

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

21

## Métodos de log - exemplo

```
...
catch (Exception ex) {
logger.error("Exceção não esperada.", ex);
return false;
}
...
```

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

22

## Uso dos métodos de log - Melhores Práticas

- Use **debug** para mensagens de depuração, que não devem ser gravadas quando a aplicação estiver em produção.
- Use **info** para mensagens similares ao modo "verbose".
- Use **warn** para avisos, que são gravados em algum lugar. Avisos não devem corresponder a situações em que a aplicação deve parar de executar.
- Use **error** para mensagens que são registradas por conta de algum erro recuperável da aplicação.
- Use **fatal** para mensagens críticas; após a gravação das mesmas, a aplicação deve ser abortada.

April 05


Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

23

## Nome de um Logger

- O padrão de nomenclatura para objetos da classe Logger é semelhante ao encontrado para classes em Java.
  - (i.e. com.pdxinc é "pai" de com.pdxinc.nhinar)
- Prática normalmente utilizada: aproveitar o nome de uma classe como o nome do seu logger.

```
public class MinhaClasse {  
    private static Logger log = Logger.getLogger(MinhaClasse.class.getName());  
    ...  
}
```



April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

24

## Appenders

---

- No log4j, Appender é uma interface que representa o destinatário da saída do log.
- Um logger pode ter vários appenders associados a ele.
  - Ou seja, o Log4j permite que uma requisição de log seja enviada para mais de um objeto que implemente Appender.

## Appenders

---

- O método **addAppender** da classe Logger adiciona um objeto Appender a um certo logger.
- Há diversas classes que implementam a interface Appender...

## Classes que implementam Appender

---

- **ConsoleAppender**: envia requisições de log para a saída padrão (System.out ou System.err).
- **FileAppender**: envia requisições de log para um arquivo
- **RollingFileAppender**
  - subclasse de FileAppender,
  - pode fazer um backup do arquivo sempre que ele atingir um determinado tamanho.

## Classes que implementam Appender (cont)

---

- **DailyRollingFileAppender**
  - subclasse de FileAppender,
  - pode fazer um backup de tempos em tempos (e.g, a cada semana).
- **SMTPAppender** - appender para enviar o log para um destinatário de e-mail
- **SocketAppender** - envia os eventos de log para um servidor de logs remoto através do protocolo TCP.

## Classes que implementam Appender (cont)

- **NTEventLogAppender** - envia para o sistema de log de uma máquina com plataforma Windows.
- **SyslogAppender** - envia os logs para um daemon (monitor de logs) remoto
- **JMSAppender** - envia os logs como uma mensagem JMS
- **AsyncAppender** - possibilita que os logs sejam coletados em um buffer e depois enviados para um ou mais appender anexados a ele.

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

29

## ConsoleAppender - exemplo

```
public class ExemploConsoleAppender {
    static Logger logger =
        Logger.getLogger(ExemploConsoleAppender.class.getName());
    public static void main(String args[]) {
        SimpleLayout layout = new SimpleLayout();
        ConsoleAppender appender = new ConsoleAppender(layout);
        logger.addAppender(appender);
        logger.setLevel(Level.DEBUG);
        logger.debug("Mensagem de depuração.");
    }
}
```

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

30

## FileAppender - exemplo

```
public class ExemploFileAppender {
    static Logger logger = Logger.getLogger(simpandfile.class);
    public static void main(String args[]) {
        SimpleLayout layout = new SimpleLayout();
        FileAppender appender = null;
        try {
            appender = new FileAppender(layout, "output1.txt", false);
        } catch (Exception e) {
            ...
        }
        logger.addAppender(appender);
        logger.setLevel((Level) Level.DEBUG);
        logger.debug("Mensagem de DEBUG");
        logger.info("Mensagem de INFO");
        logger.warn("Mensagem de WARN");
        logger.error("Mensagem de ERROR");
        logger.fatal("Mensagem de FATAL");
    }
}
```

## Layouts

- Além de registrar mensagens de log, o Log4j pode também registrar:
  - Instante (data e hora) em que o log foi requisitado,
  - Prioridade da mensagem (DEBUG, WARN, FATAL etc.),
  - Nome da classe, nome do método, número da linha no código fonte, onde o log foi requisitado,
  - etc
- O formato do que é registrado em log é especificado no **layout** ao qual o appender é associado.



## Layouts

- Um appender deve ter um Layout associado.
- Layout é a classe base do Log4j responsável pela formatação da saída para um certo objeto Appender.
  - Classes do Log4j que especificam a formação estendem Layout

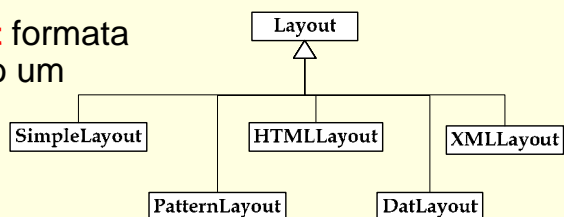
April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

33

## Subclasses de Layout

- Há no log4j diversas subclasses de Layout predefinidas:
  - **SimpleLayout**: imprime o nível, o símbolo '-' e a mensagem de log.
  - **HTMLLayout** formata a saída como HTML.
  - **XMLLayout** formata a saída como XML.
  - **PatternLayout** formata a saída usando um padrão de conversão.
  - **DateLayout**



April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

34

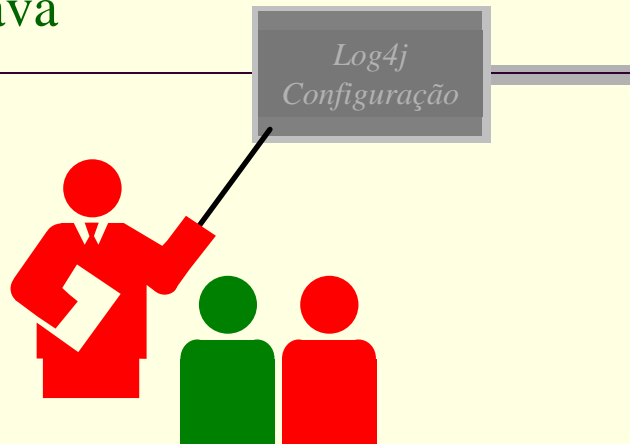
## SimpleLayout - exemplo

```
public class ExemploConsoleAppender {
    static Logger logger =
        Logger.getLogger(ExemploConsoleAppender.class.getName());
    public static void main(String args[]) {
        SimpleLayout layout = new SimpleLayout();
        ConsoleAppender appender = new ConsoleAppender(layout);
        logger.addAppender(appender);
        logger.setLevel(Level.DEBUG);
        logger.debug("Mensagem de depuração.");
    }
}
```

## PatternLayout - exemplo

```
public class ExemploPatternLayout {
    static Logger logger =
        Logger.getLogger(ExemploPatternLayout.class.getName());
    public static void main(String args[]) {
        String pattern = "Duração do programa em milisegundos: %r %n";
        pattern += "Nome da classe: %C %n";
        pattern += "Data no formato ISO8601: %d{ISO8601} %n";
        pattern += "Local do evento de log: %l %n";
        pattern += "Mensagem: %m %n %n";
        PatternLayout layout = new PatternLayout(pattern);
        ConsoleAppender appender = new ConsoleAppender(layout);
        logger.addAppender(appender);
        logger.setLevel(Level.DEBUG);
        logger.debug("Mensagem de depuração.");
    }
}
```

## POO-Java



April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

37

## Configuração do log4j por arquivo

- As configurações do log4j **não** devem ser feitas no código fonte.
  - Isso porque é desejável mudar as opções de logging sem ter que recompilar a aplicação.
- Modo adequado: através de um arquivo de configuração.
- Há dois modos de especificar o arquivo de configuração:
  - Arquivo de propriedades (PropertyConfigurator)
  - Arquivo XML (DOMConfigurator)

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

38

## PropertyConfigurator - exemplo

- Considere o seguinte arquivo de propriedades:

```
#define o nível do logger raiz e o nome de seu appender (htmlAppender)
log4j.rootLogger=DEBUG, htmlAppender

# define a classe do appender
log4j.appender.htmlAppender=org.apache.log4j.RollingFileAppender

#nome do arquivo de log
log4j.appender.htmlAppender.File=logging/HTMLLayout.html

# define a classe para formação (layout)
log4j.appender.htmlAppender.layout=org.apache.log4j.HTMLLayout

# define que deve ser registrada informação de contexto
log4j.appender.htmlAppender.layout.LocationInfo=true

# título do arquivo de log
log4j.appender.htmlAppender.layout.Title=Log gerado pelo Log4j
```

## PropertyConfigurator – exemplo (cont.)

- Considere que o arquivo de propriedades anterior foi passado como argumento para o programa a seguir:

```
public class ExemploPropertyConfigurator {
    static Logger logger =
        Logger.getLogger(ExemploPropertyConfigurator.class.getName());
    public static void main(String[] args) {
        PropertyConfigurator.configure(args[0]);
        logger.info("Aplicação iniciada.");
        doSomething();
        logger.info("Aplicação terminada.");
    }
}
```

## PropertyConfigurator – exemplo (cont.)

- O arquivo de log criado a partir do programa anterior:
  - Terá o nome HTMLLayout.html,
  - Terá como conteúdo uma tabela HTML com as mensagens de log em cada linha da tabela.

## PropertyConfigurator – exemplo (cont.)

- Para cada mensagem, terá:
  - o tempo (em milisegundos) decorrido desde quando o programa foi iniciado,
  - a thread que disparou o log,
  - o nível de prioridade,
  - (se LocationInfo = true) a classe, o nome do arquivo \*.java desta classe e o número da linha,
  - a mensagem de log propriamente dita.

## Customizações com PatternLayout

**Pattern:** "%r [%t] %-5p %c{2} - %m%n"

176 [main] INFO examples.Sort - Populating an array of 2 elements in reverse order

- r - número de milissegundos transcorridos desde o início do programa
- t - nome da thread que gerou o evento de log
- p - prioridade (o -5 indica que deve alinhar à direita se o número de caracteres for menor que cinco)

## Customizações com PatternLayout

**Pattern:** "%r [%t] %-5p %c{2} - %m%n"

176 [main] INFO examples.Sort - Populating an array of 2 elements in reverse order

- c - nome da classe (2 indica que se o nome completo da classe for "a.b.c" por exemplo, deverá ser mostrado apenas "b.c")
- m - é a mensagem (não pode faltar !)
- n - é o separador de linhas do SO ("\n" ou "\r\n")

## POO-Java

Log4j Resumo



April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

45

## Log4j - Resumo

- Principais componentes:
  - Logger: permite realizar requisições de log.
  - Appender: corresponde ao destinatário de uma mensagem de log.
  - Layout: especifica o formato de uma mensagem de log

April 05

Prof. Ismael H. F. Santos - ismael@tecgraf.puc-rio.br

46

## Log4j - Resumo

---

- Propriedades de um objeto Logger
  - nome
  - nível
- Hierarquia de loggers oferece um maior controle;
- Diversas saídas: Console, Arquivo, Banco de Dados, XML, HTML, e-mail, ...
- API de código aberto;

## Log4j - Resumo

---

- Elimina a necessidade de uso de "System.out" para depuração do código fonte;
- Permite controlar de maneira flexível as saídas de log;
- Configuração do log em tempo de execução sem alteração na codificação e sim em um arquivo de configuração;
- Com o uso de logging, há a possibilidade de diminuição do desempenho da aplicação.



## Exercício

- Experimente utilizar os arquivos de configuração a seguir:

```
log4j.rootLogger=DEBUG, htmlAppender
# htmlAppender is set to be a ConsoleAppender which outputs to System.out.
log4j.appender.htmlAppender=org.apache.log4j.ConsoleAppender
# htmlAppender uses PatternLayout.
log4j.appender.htmlAppender.layout=org.apache.log4j.PatternLayout
log4j.appender.htmlAppender.layout.ConversionPattern=%-5p [%t]: %m%n
#log4j.logger.package.MyApp =WARN
```

```
log4j.rootLogger=DEBUG, htmlAppender
log4j.appender.htmlAppender=org.apache.log4j.ConsoleAppender
log4j.appender.htmlAppender.layout=org.apache.log4j.PattenLayout
log4j.appender.htmlAppender.layout. ConversionPattern=%-4r [%t] %-5p %c - %m%n
```

## Informações Adicionais

- Site do Log4j
  - <http://jakarta.apache.org/log4j/docs/index.html>