FIAP GRADUAÇÃO

DIGITAL BUSINESS ENABLEMENT

Prof. MSc. Rafael Matsuyama

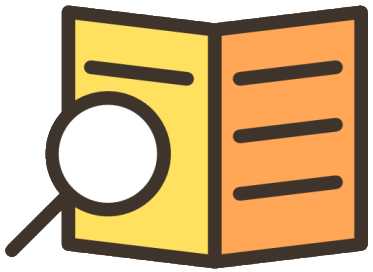
#03 – PADRÕES DE PROJETOS E FRAMEWORKS

 *profrafael.matsuyama@fiap.com.br*  *rafaelmatsuyama*

PERCURSO

- Java Application
- Padrões de Projetos e Frameworks

#03 - AGENDA



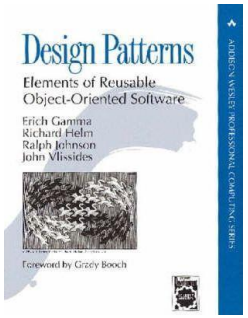
- Padrões de Projetos
  - Categoria dos padrões - GoF
  - Camadas de um sistema
  - Transfer Object
  - Singleton
  - Façade
- Frameworks
  - Frameworks Java
  - Framework de Log

PADRÕES DE PROJETOS

PADRÕES DE PROJETOS

- **Design Patterns** são soluções para problemas **comuns** que encontramos no desenvolvimento ou manutenção de um sistema orientado a objetos.

- O primeiro grande trabalho sobre o assunto foi o livro "**Design Patterns: Elements of Reusable Object-Oriented Software",** escrito por *Erich Gamma*, *Richard Helm*, *Ralph Johnson* e *John Vlissides.* escrito por *Erich Gamma*, *Richard Helm*, *Ralph Johnson* e *John Vlissides* ***(GoF, Gang of Four).*** O livro discutiu 23 padrões de projetos.



CATEGORIA DOS PADRÕES - GOF

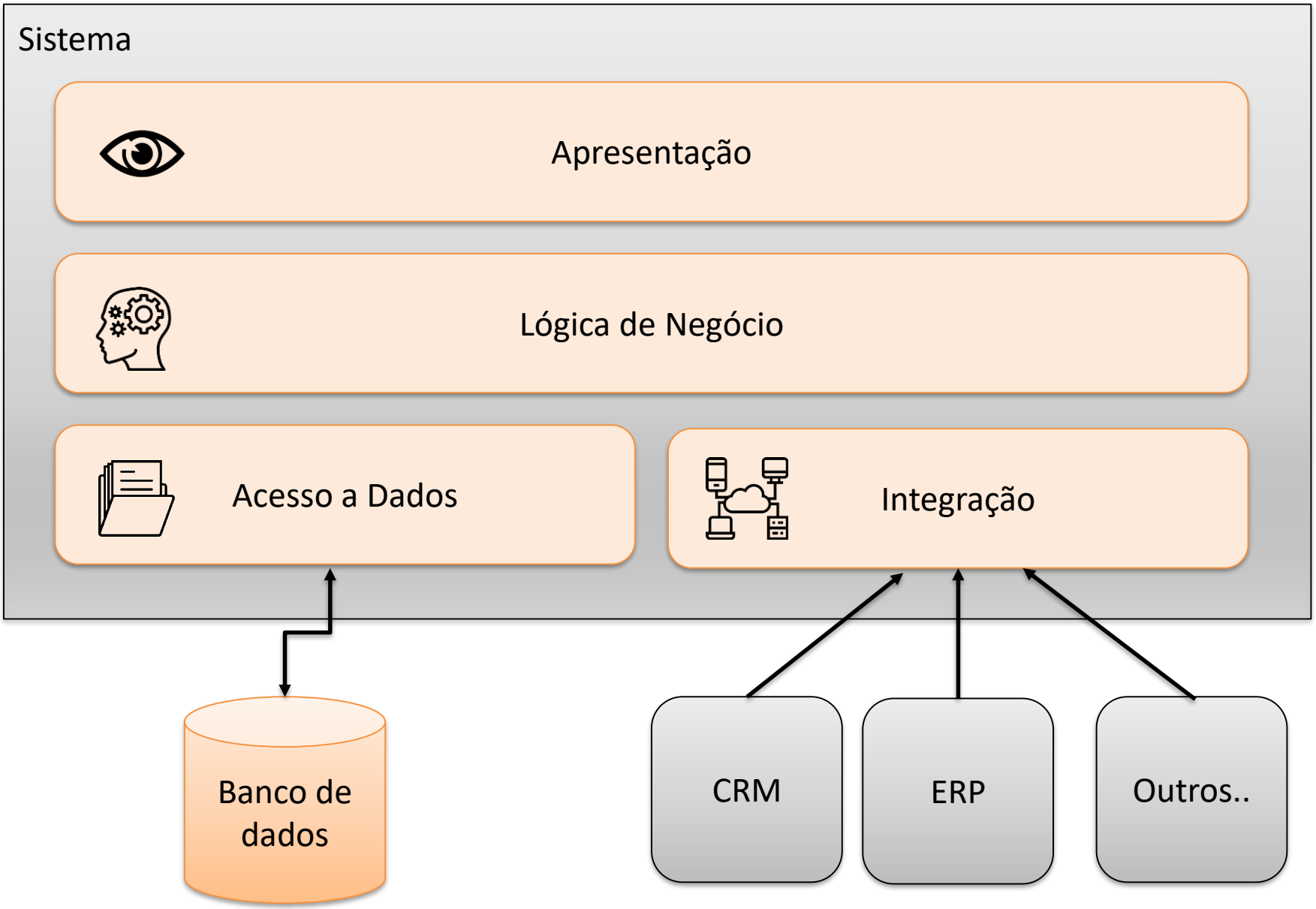
Os **padrões GoF** foram divididos e categorizados de acordo com a natureza do problema que eles resolvem:

- **Padrões de Criação:** Tem como objetivo abstrair a instanciação de objetos.
  - Abstract Factory
  - Singleton
- **Padrões estruturais:** Se preocupam em organizar a estrutura das classes e os relacionamentos entre classes e objetos.
  - Adapter
  - Facade
- **Padrões comportamentais:** Atuam diretamente na delegação de responsabilidades, definindo como os objetos devem se comportar e se comunicar.
  - Iterator
  - Strategy

PADRÕES DE PROJETOS

- Atualmente **existem vários de padrões de projetos** sendo utilizados pelos desenvolvedores.
- Um padrão de projeto deve, **obrigatoriamente**, ter as seguintes características:
  - Ter um **nome**;
  - Definir o **problema** que será resolvido;
  - **Solucionar** o problema;
  - Definir as **consequências** da adoção do padrão;

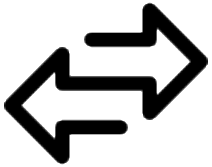
CAMADAS DE UMA APLICAÇÃO



TRANSFER OBJECT (TO)

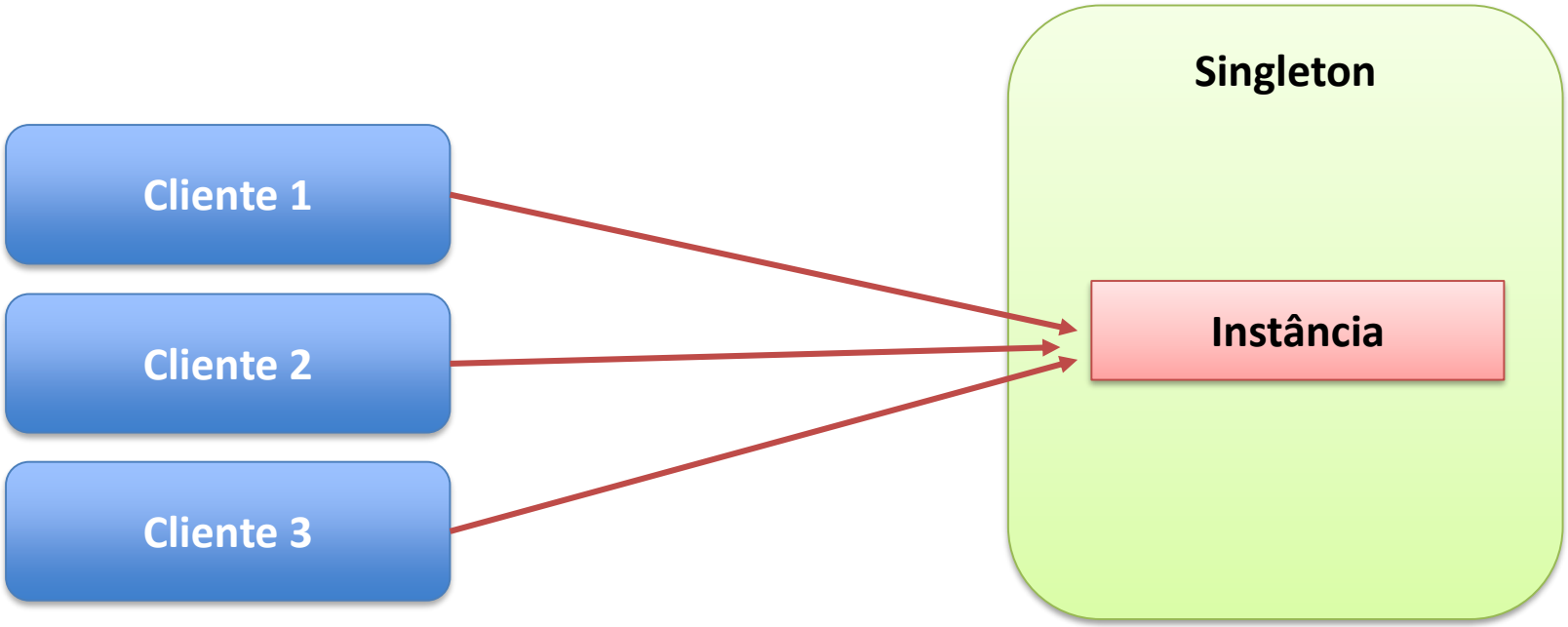
- **Transfere** os dados entre as camadas;
- Também pode ser encontrado com outros nomes como **Value Object (VO) e Data Transfer Object (DTO);**

```
public class VooTO implements Serializable {

        private String origem;
        private String destino;
        private String horario;

        public String getOrigem() {
                return origem;
        }

        public void setOrigem(String origem) {
                this.origem = origem;
        }

        //...
}
```



SINGLETON

- É aplicado quando se deseja que exista apenas **uma instância da classe**;
- Esse padrão é implementando de forma que **a própria classe fique responsável por instanciar** e oferecer a única instância dela mesma.



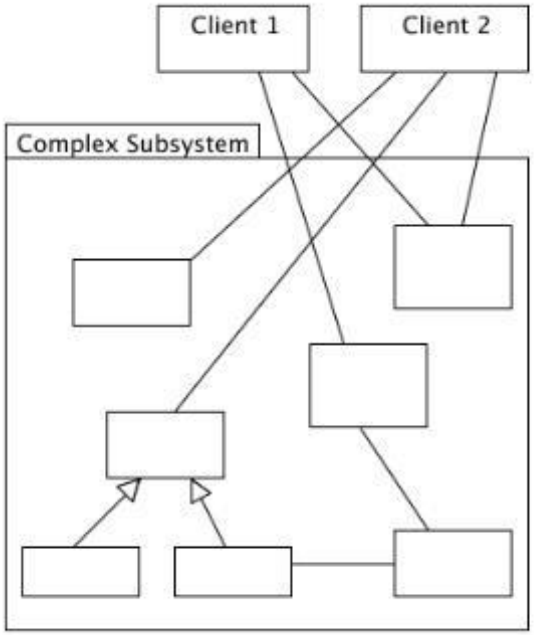
SINGLETON - EXEMPLO

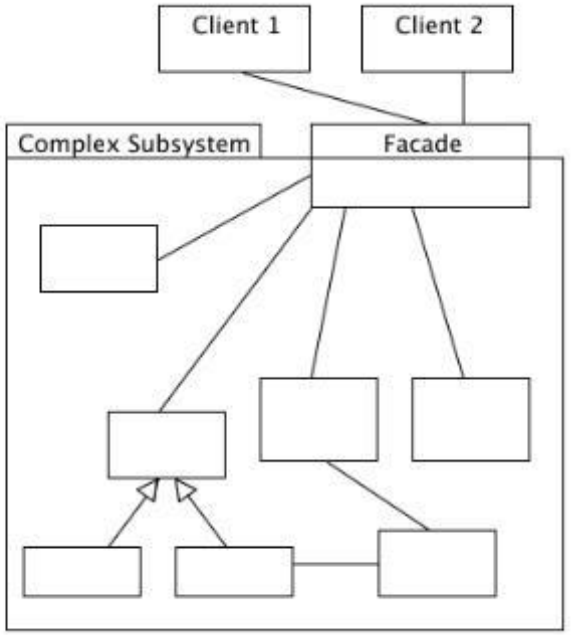
- Sempre recomendável utilizar **arquivos de propriedades** para armazenar variáveis que representam parâmetros do sistema;
- O objeto **Properties** é ideal para armazenar este tipo de valor;

```
public class PropertySingleton {
  private static Properties p;
  private static final String ARQ = "/arquivo.properties";

  public static Properties getInstance() {
            if (p == null){
              try {
                        p = new Properties();
                        p.load(PropertySingleton.class.getResourceAsStream(ARQ));
              } catch (IOException e) {
                        e.printStackTrace();
              }
            }
            return p;
  }
}
```

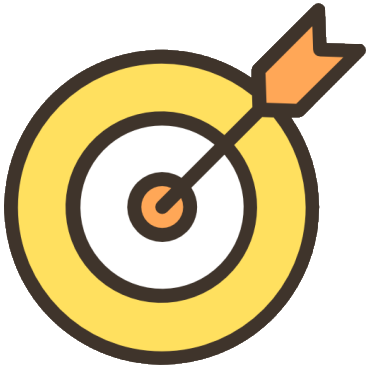
FAÇADE (FACHADA)

- **Simplifica a utilização** de um subsistema complexo apenas implementando uma classe que fornece uma interface única para acessar as funcionalidades;
- **Oculta toda a complexidade** de uma ou mais classes através de uma Facade (Fachada);





PRÁTICA! REFACTORING



Utilize o exercício anterior (Apostila 02, **01-Loja-App**);

1. Separar a camada de apresentação (Presentation Layer) da camada de negócio (Business Layer) . A classe de negócio será **br.com.fiap.loja.bo.EstoqueBO** com o método **consultarProduto(int codProduto)**;

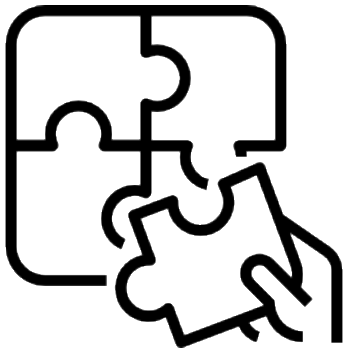
2. Para transferir as informações entre as camadas, crie a estrutura de **produtos em formato de TO** com os atributos: código, preço, quantidade e descrição;

3. Ajuste a tela, exiba o nome da loja filial da empresa. **Esta informação deverá ser obtida por meio de um arquivo de propriedades.**

FRAMEWORKS

FRAMEWORKS

- Pode ser visto como um software incompleto, que **deve ser preenchido com partes específicas da aplicação**;
- Provê a **reutilização de código, estrutura e design**. Muitos **já implementam** alguns padrões de projetos;
- **Benefícios do uso de um Framework no desenvolvimento de Software:**
  - Redução do esforço de programação = maior produtividade;
  - Interoperabilidade;
  - Redução do esforço de aprendizado;
  - Redução do esforço de projetar e implementar;
  - Promover o reuso de software;
  - Estar de acordo com boas práticas de mercado, entre outros ...

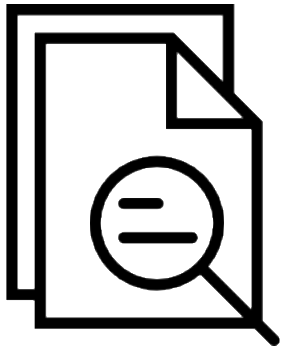


FRAMEWORKS - JAVA

- **Existem milhares de frameworks** em praticamente todas as plataformas/linguagens de programação;
- Na **plataforma Java**, existem vários frameworks para cada tipo de necessidade:
  - **Web:**
    - JSF, Spring MVC, Struts, VRaptor;
  - **Persistência:**
    - JPA, Hibernate, Spring Data;
  - **Gerenciador de transação:**
    - JTA, Spring AOP Transaction;
  - **Container IoC:**
    - Spring, CDI, EJB;

FRAMEWORKS - LOGGING

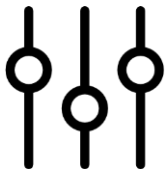
- Muito utilizado em **aplicações corporativas**;
- Elemento importante para coletar de informações em **ambiente de produção;**
- Na **plataforma Java**, existem várias implementações, as mais conhecidas são: **Log4J**, **Apache Commons Log** ou **Simple Logging Facade for Java** (SLF4J);
- Todas se baseiam no conceito de um **arquivo de propriedade** onde os logs podem ser configurados;



FRAMEWORKS - LOGGING

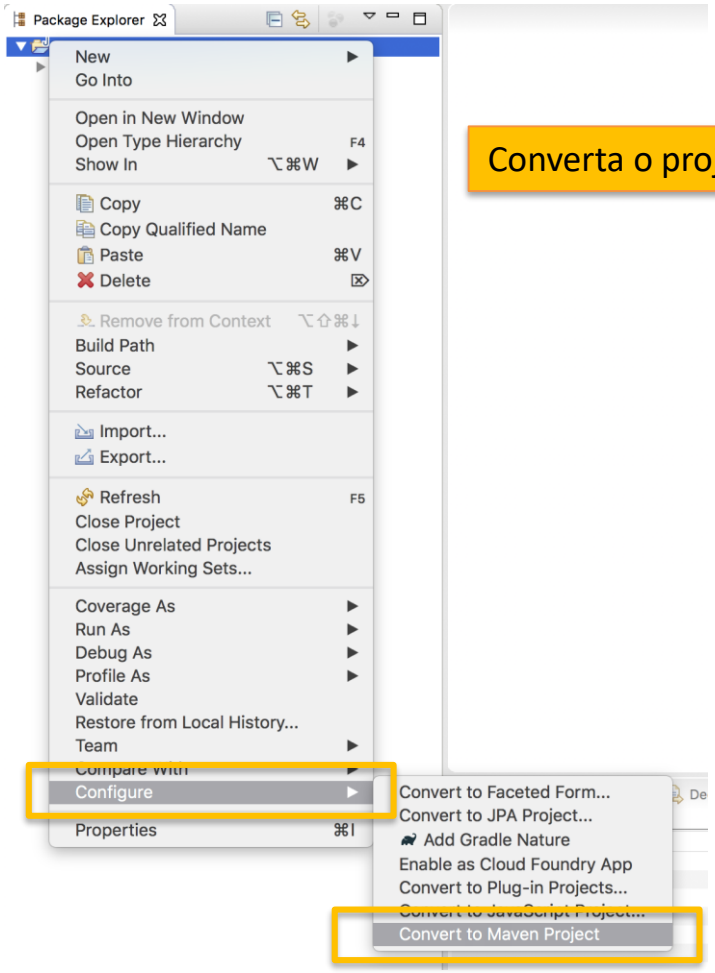
- As mensagens de log podem apresentar diferentes níveis:
  - **TRACE** (Informação para identificar um comportamento da aplicação. Imprime todos os níveis de log);
  - **DEBUG** (Informação para depurar uma aplicação. Apresenta todos os logs exceto trace);
  - **INFO** (Mensagens informativas da aplicação. Mostra todos os logs exceto trace e debug);
  - **WARN** (Mostra avisos de comportamentos inesperados ou suspeitos da aplicação. Mostra somente logs de WARN e ERROR);
  - **ERROR** (Mostra Mensagens de erro da aplicação. Mostra apenas logs de ERROR).

Níveis de Apresentação ↓



LOG4J

- Para utilizar o **Log4J** é preciso adicionar a **dependência** (biblioteca) e criar um **arquivo de configuração** (properties ou xml);



Converta o projeto para **Maven**

```
<dependencies>
 <dependency>
        <groupId>log4j</groupId>
        <artifactId>log4j</artifactId>
        <version>1.2.17</version>
 </dependency>
</dependencies>
```

Adicione a dependência no pom.xml

LOG4J - PROPERTIES

- Na raíz do diretório **SRC**, crie o arquivo **log4j.properties**;
- Esse arquivo possui as **configurações** para definir o formato de apresentação dos dados dos logs (TRACE, DEBUG, INFO, WARN e ERROR).

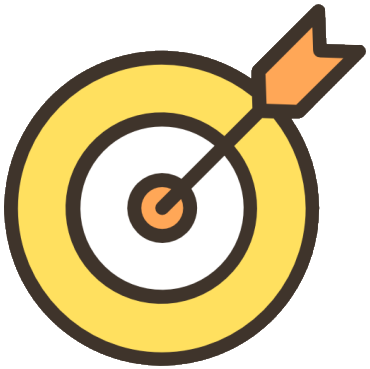
```
# Root logger option
log4j.rootLogger=DEBUG, stdout, file


# Redirect log messages to console
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.Target=System.out
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %-5p %c{1}:%L - %m%n


# Redirect log messages to a log file, support file rolling.
log4j.appender.file=org.apache.log4j.RollingFileAppender
log4j.appender.file.File=C:\\log4j-application.log
log4j.appender.file.MaxFileSize=5MB
log4j.appender.file.MaxBackupIndex=10
log4j.appender.file.layout=org.apache.log4j.PatternLayout
log4j.appender.file.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %-5p %c{1}:%L - %m%n
```

LOG4J - IMPLEMENTAÇÃO

- Crie o **log** para a classe, o parâmetro do método **getLogger** é a própria classe, para ser possível identificar qual classe gerou o log;
- O objeto log possui **métodos** para cada nível de log (trace, info, debug, warn, error);

```
import org.apache.log4j.Logger;

public class App {
    private static Logger log = Logger.getLogger(App.class);

    public static void main( String[] args ){
        log.debug("Log de debug");
    }
}
```

PRÁTICA! LOG



Ainda no exercício anterior, implemente:

1. Diferentes níveis de logs no sistema:
    - **DEBUG**: Indicando a condição lógica por onde a aplicação passou. Neste caso indicando o caminho do produto em função da escolha;
    - **WARN**: Indicando início e fim da aplicação;
    - **ERROR**: Caso ocorra algum erro não esperado em virtude do usuário ter digitado um código de produto não existente.
2. Fazer deployment da aplicação para Windows (formato .exe).

VOCÊ APRENDEU...

- O que são **padrões de projetos** e o motivo de ser muito utilizado;
- **Tipos** de padrões de projetos e os padrões: **transfer object** e **singleton**;
- O que são **frameworks**;
- Quais são os principais frameworks de mercado **Java**;
- Utilizar um framework de **log**;

"Aquele que não luta pelo futuro que quer, deve aceitar o futuro que vier"