FIAP GRADUAÇÃO

FIAP

DIGITAL BUSINESS ENABLEMENT

Prof. THIAGO T. I. YAMAMOTO

#03 – DESIGN PATTERNS E FRAMEWORKS

 thiagoyama  thiagoyama@gmail.com

AGENDA

- Padrões de Projetos (Design Patterns)
- Frameworks
- Framework de Log

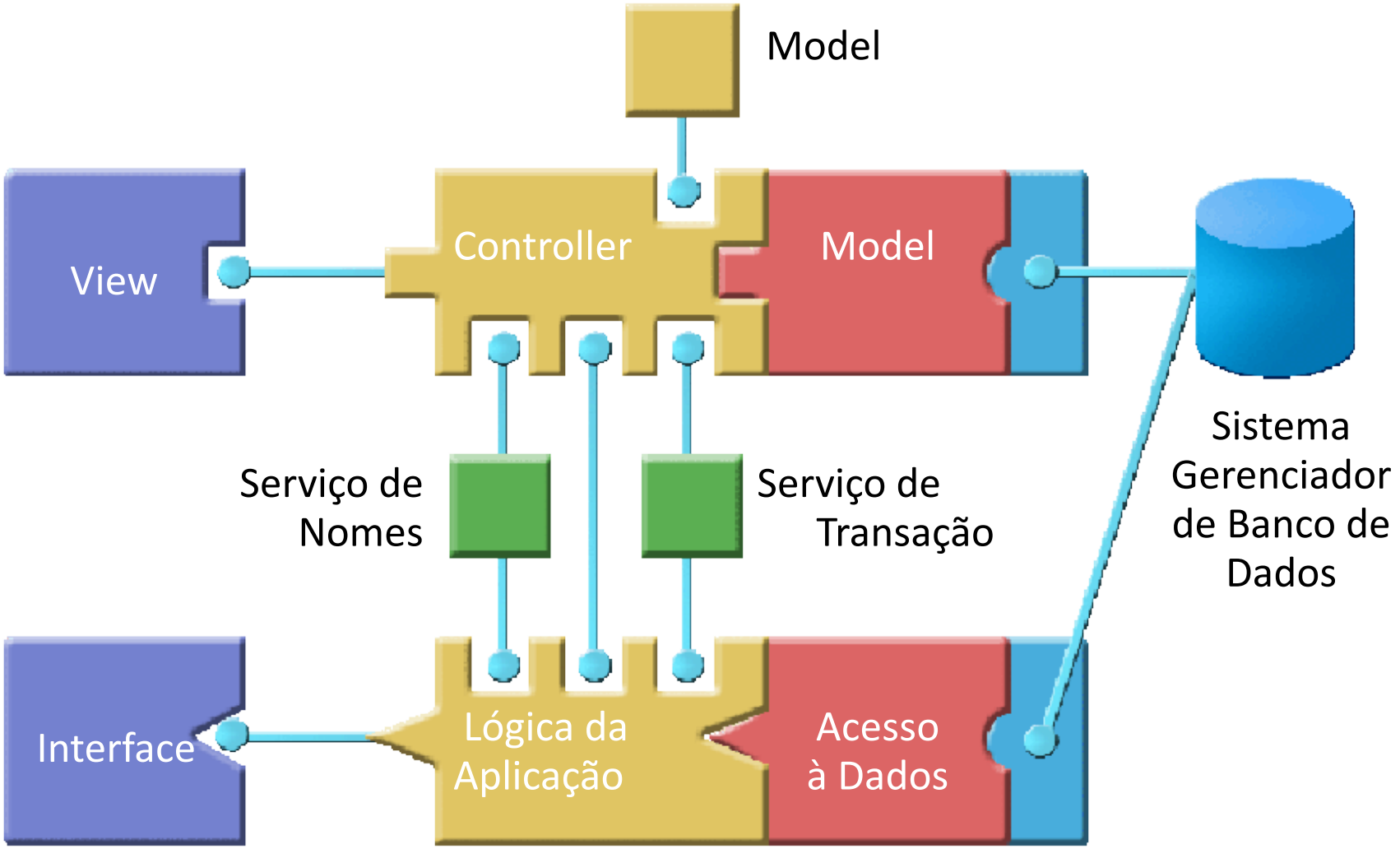
FIAP

DESIGN PATTERN

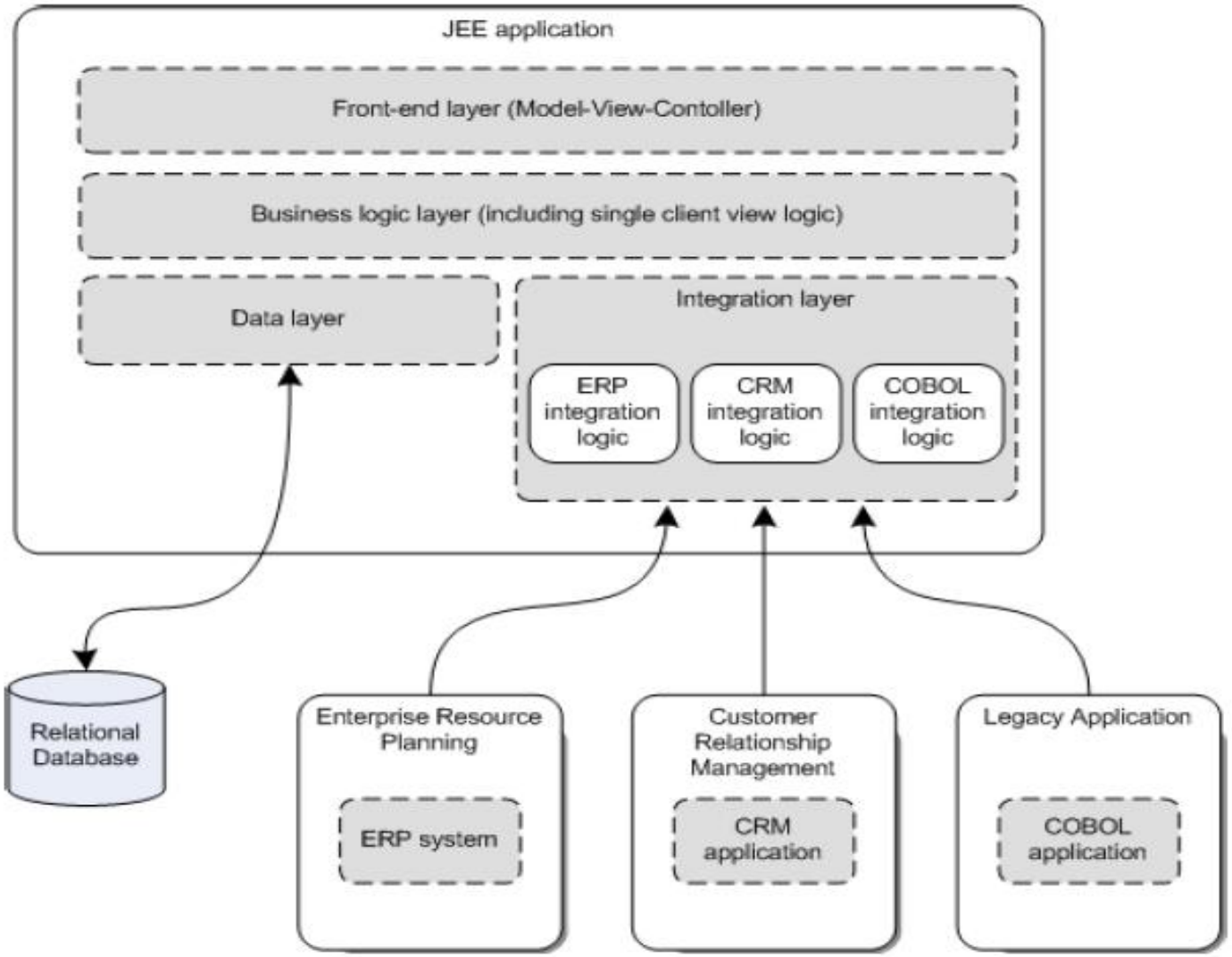
DESIGN PATTERN

- Padrões de Projeto (Design Patterns) são soluções para problemas de engenharia de software, desenvolvidas pela comunidade de orientação a objetos. A meta da comunidade de padrões é construir uma base de conhecimento que suporte soluções de design e desenvolvimento em geral;
- Os Padrões de Projeto tornam mais fácil reutilizar soluções e arquiteturas bem sucedidas para construir softwares orientados a objetos de forma flexível e fácil de manter;
- Estas soluções vão desde a solução de problemas de domínio de negócio até a arquitetura de software;
- A utilização de padrões de projeto pode ajudar de forma significativa a geração de modelos de componentes;
- Podemos ter patterns nos mais diversos níveis de granularidade.

DESIGN PATTERN : MVC (MODEL-VIEW-CONTROLLER)

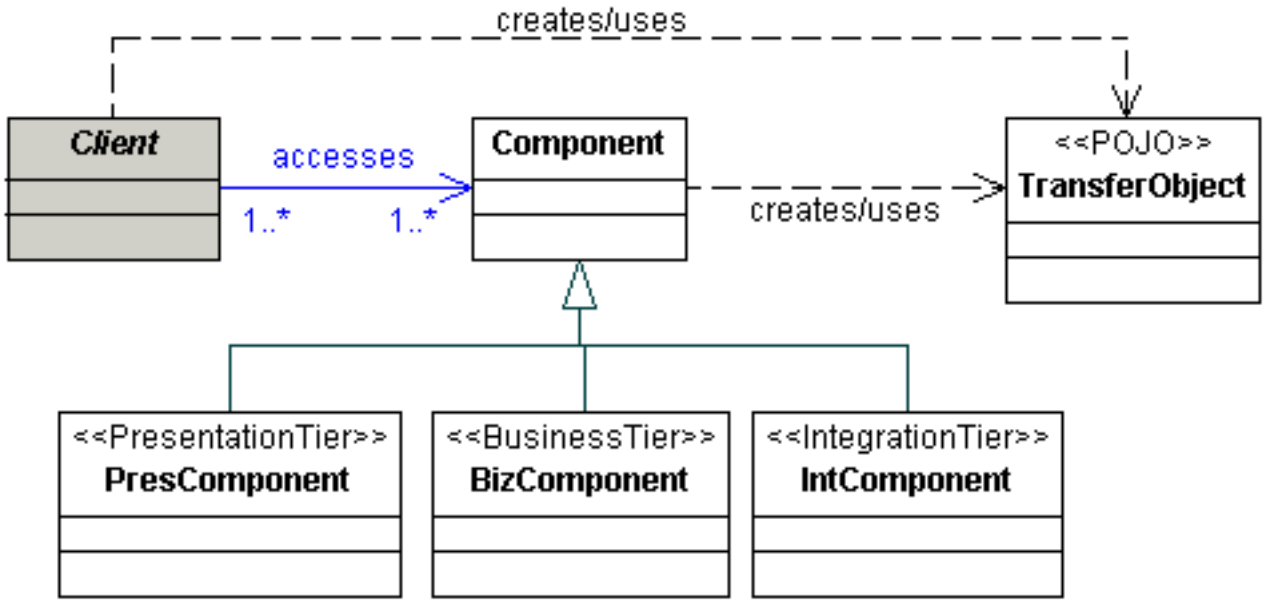


PATTERN DE DISTRIBUIÇÃO DE CAMADAS DE APLICAÇÃO



DESIGN PATTERN : TRANSFER OBJECT (TO)

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



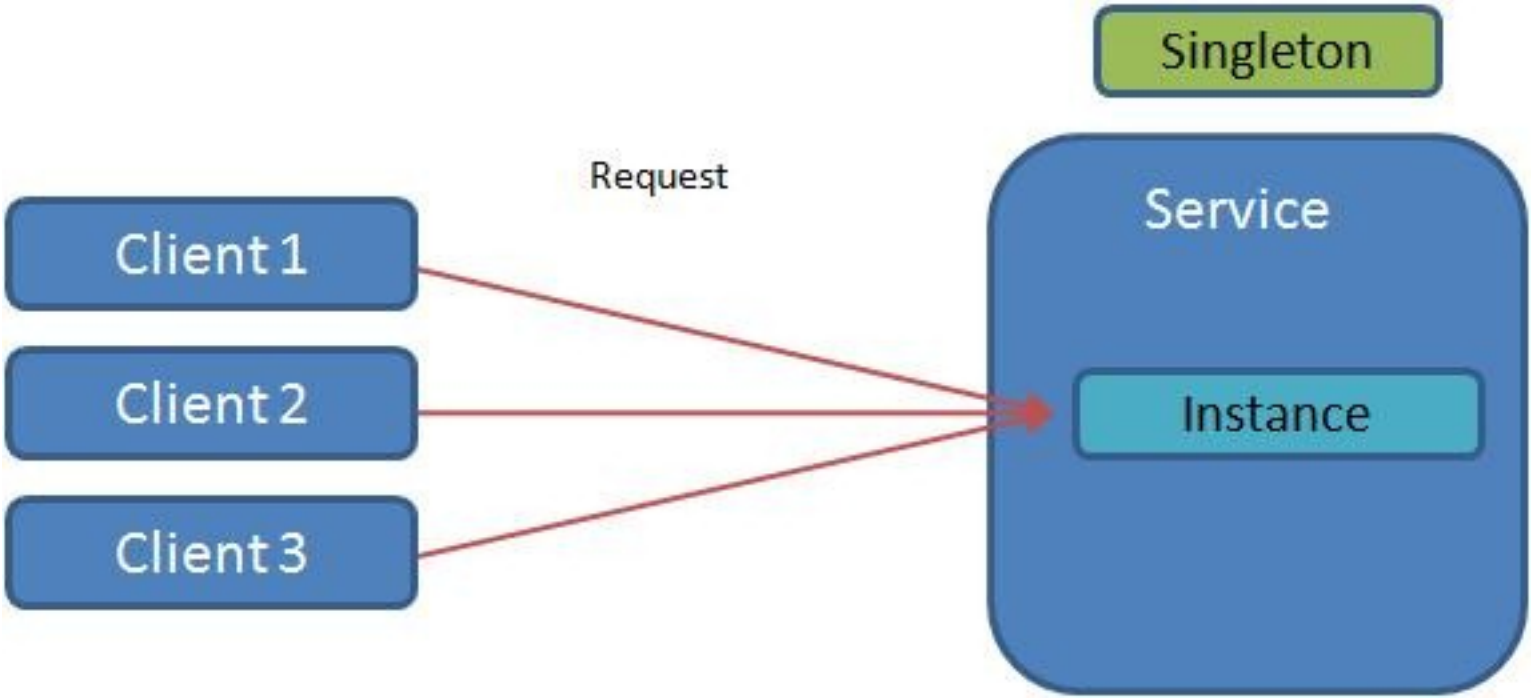
Link: http://www.oracle.com/technetwork/java/transferobject-139757.html

TRANSFER OBJECT – ESTRUTURA DE DADOS

```
public class VooTO implements Serializable {

    private static final long serialVersionUID = 1L;

    //Opcional
    public VooTO() {}

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

    public String getOrigem() {
    return origem;
    }

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

    ...
}
```

DESIGN PATTERN : SINGLETON

- O padrão de projeto Singleton existe para ser 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

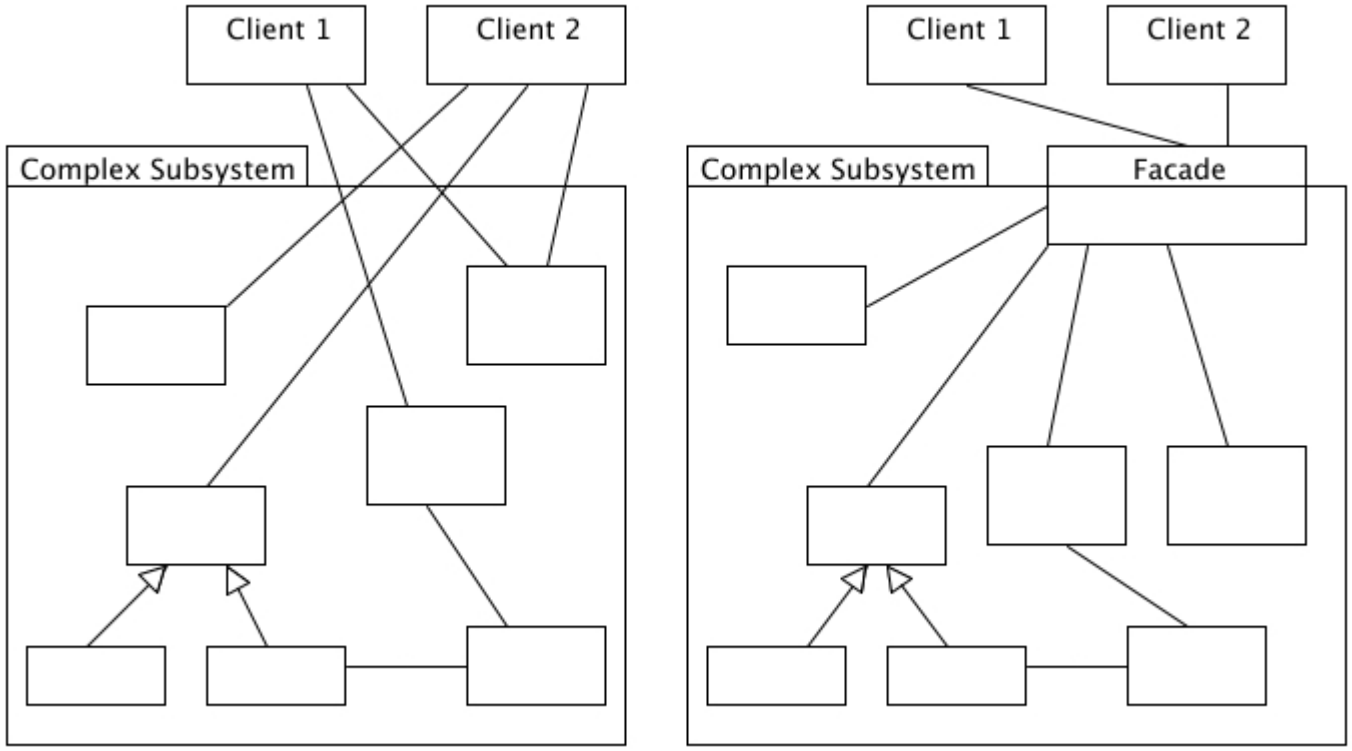


APLICANDO SINGLETON – PROPRIEDADES DE SISTEMA

- Sempre recomendável utilizar arquivos de propriedades para armazenar variáveis que representam parâmetros do seu sistema
- O objeto **Properties** é ideal para armazenar este tipo de valor
- Classes estáticas não precisam de instância na classe. O objeto pode ser chamado diretamente como **PropertySingleton.getInstance().getProperty("propriedade")**

```
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;
   }
}
```

DESIGN PATTERN FACADE



Fonte: http://best-practice-software-engineering.ifs.tuwien.ac.at/patterns/facade.html

EXERCÍCIO: APPLICATION REFACTORING

FIAP

- Separar camada de apresentação (Presentation Layer) da camada de negócio (Business Layer) . A classe de negócio será **com.fiap.loja.EstoqueBO** com o método **consultarProduto(int codProduto)**;
- Alterar aplicação do exercício anterior para armazenar a estrutura de produtos em formato de TO criando os atributos preço unitário (Double) e quantidade em estoque (int);
- Utilize geração automática de set's e get's para o TO;
- Torne o TO um objeto serializável (implements Serializable);
- Formatar a apresentação do preço em formato monetário:
  - DecimalFormat df = new DecimalFormat("R$ #,###.00");
  - System.*out.println(df.format(valor));*
- Incluir o nome da loja filial da empresa na tela. **Esta informação deverá ser obtida por meio de um arquivo de propriedades.**

FRAMEWORKS

FRAMEWORKS - CONCEITOS

- (Estrutura) um framework é uma coleção de classes abstratas e concretas e a interface entre elas, representando o projeto de um subsistema (Pree, 1995)

- (Propósito) esqueleto de uma aplicação, que pode ser instanciado por um desenvolvedor de aplicações (Jonhson, 1997). Trata-se de uma aplicação semi-completa, reutilizável que, quando especializada, produz aplicações personalizadas (Jonhson and Foote, 1998).

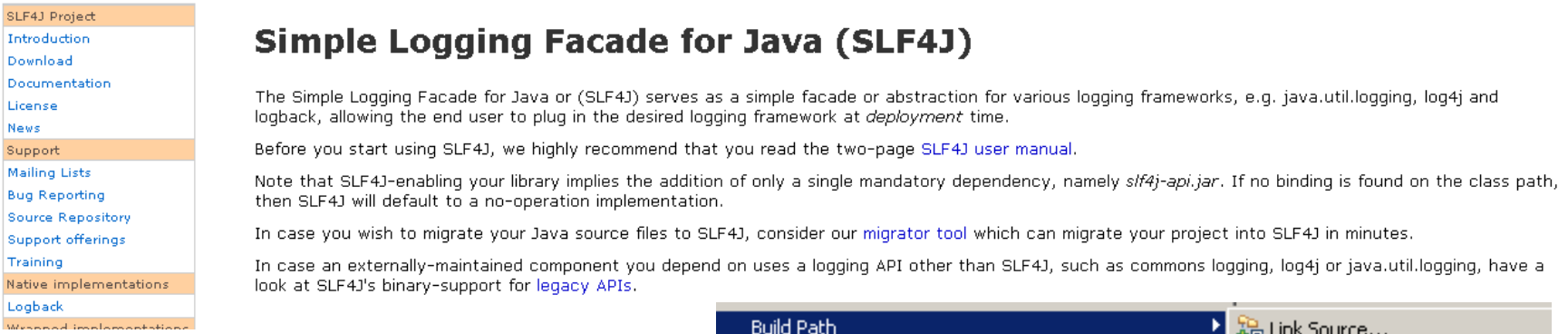
- Framework é um programa incompleto que servirá de base para diversos outros programas. Isso quer dizer que ele deve ser geral o suficiente para abranger diversos programas e também específico o suficiente para atender às necessidades de cada um.

- **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 ...

FRAMEWORK PARA LOGGING

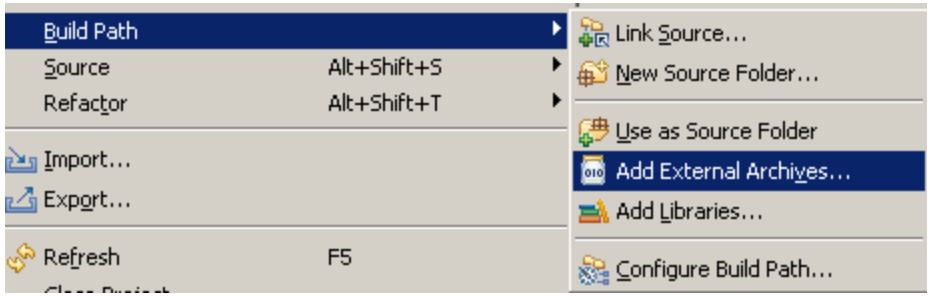
- Uso extensivo em aplicações corporativas;
- Elemento importante para ser utilizado para coleta de informações em ambiente de produção;
- Pode ser implementado de várias formas. Implementações mais conhecidas são o Log4J, Apache Commons Log ou Simple Logging Facade for Java (SLF4J);
- Todas elas baseiam-se no conceito de um arquivo de propriedade onde os logs podem ser configurados;
- 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).

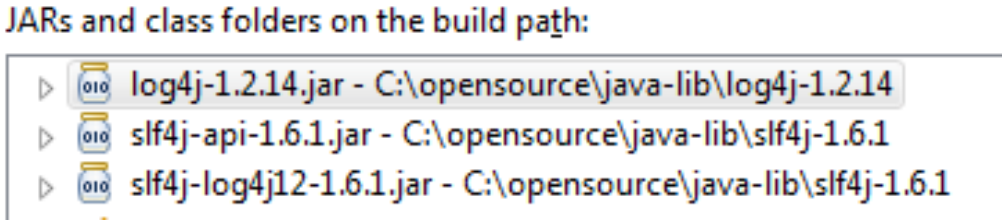
SLF4J (HTTP://WWW.SLF4J.ORG/)



Bibliotecas (JAR's):

Botão direito no projeto ->





Observação: Biblioteca do log4j deve ser baixada através do link:

http://logging.apache.org/

IMPLEMENTAÇÃO NA CLASSE

- Faça os imports do SLF4J:

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
```

- Implemente uma propriedade para classe para saber quem foi o gerador do log (similar ao Log4J):

```
private static final Logger log = LoggerFactory.getLogger(Cadastro.class);
```

- Implemente as condições de Log:

```
if(expressaoValida) {
    log.debug("Estou registrando no log para debug");

} else {
    log.error("Aconteceu um Erro");

}
```

ARQUIVO DE PROPRIEDADES (LOG4J.PROPERTIES)

- Deve ser armazenado na raíz do diretório SRC da aplicação;
- Possui as propriedades utilizadas para definir o formato de apresentação dos dados dos logs (TRACE, DEBUG, INFO, WARN e ERROR).

```
log4j.rootLogger=DEBUG,stdout

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{ABSOLUTE} %5p %c{1}:%L -
%m%n
```

Resultado do Log:
14:48:59,734 DEBUG TerminalConsulta:20 – Mensagem no Log debug

EXEMPLOS DE FORMATO DE MENSAGENS DE LOG

log4j.appender.stdout.layout.ConversionPattern=%d{ABSOLUTE} %5p %c{1}:%L - %m%n
13:00:54,654 ERROR ReceberCarga:29 - Esta e uma mensagem.

log4j.appender.stdout.layout.ConversionPattern=%-5p [%t]: %m%n
ERROR [main]: Esta e uma mensagem.

log4j.appender.stdout.layout.ConversionPattern=%d [%c{1}] %p - %m%n
2011-03-26 13:00:54,654 [ReceberCarga] ERROR - Esta e uma mensagem.

log4j.appender.stdout.layout.ConversionPattern=%d{dd MMM yyyy HH:mm:ss} %-5p [%c] - %m%n
26 Ago 2011 13:06:36 ERROR [com.fiap.RecebeApp] - Esta e uma mensagem.

log4j.appender.stdout.layout.ConversionPattern=%5p [%C:%M] (%F:%L) - %m%n
ERROR [org.jboss.s4lf.JBossLogger:info] (JBossLogger.java:68) - Esta e uma mensagem.

log4j.appender.stdout.layout.ConversionPattern=%d{ABSOLUTE} %r %5p (%l) - %m%n
13:38:42,976 10 ERROR (org.jboss.s4lf.JBossLogger:info(JBossLogger.java:68)) - Esta e uma mensagem.

- **Para maiores informações, acesse:**

http://www.mobilefish.com/developer/log4j/log4j_quickguide_layouts.html

EXERCÍCIO – IMPLEMENTAR FUNCIONALIDADE DE LOG

Ainda no exercício anterior, implemente:

- Defina diferentes níveis de mensagens para visualizar as diferenças:
  - 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.
- Altere o arquivo de propriedade do log4j.properties para ver as diferenças de mensagens no log;
- Documentar as classes utilizando notação de javadoc;
- Fazer deployment da aplicação para Windows (formato .exe).

FIAP

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