FIAP GRADUAÇÃO

FIAP

ENTERPRISE APPLICATION DEVELOPMENT

Prof. Me. Thiago T. I. Yamamoto

#10 – ENTERPRISE JAVA BEANS

 thiagoyama@gmail.com  thiagoyama

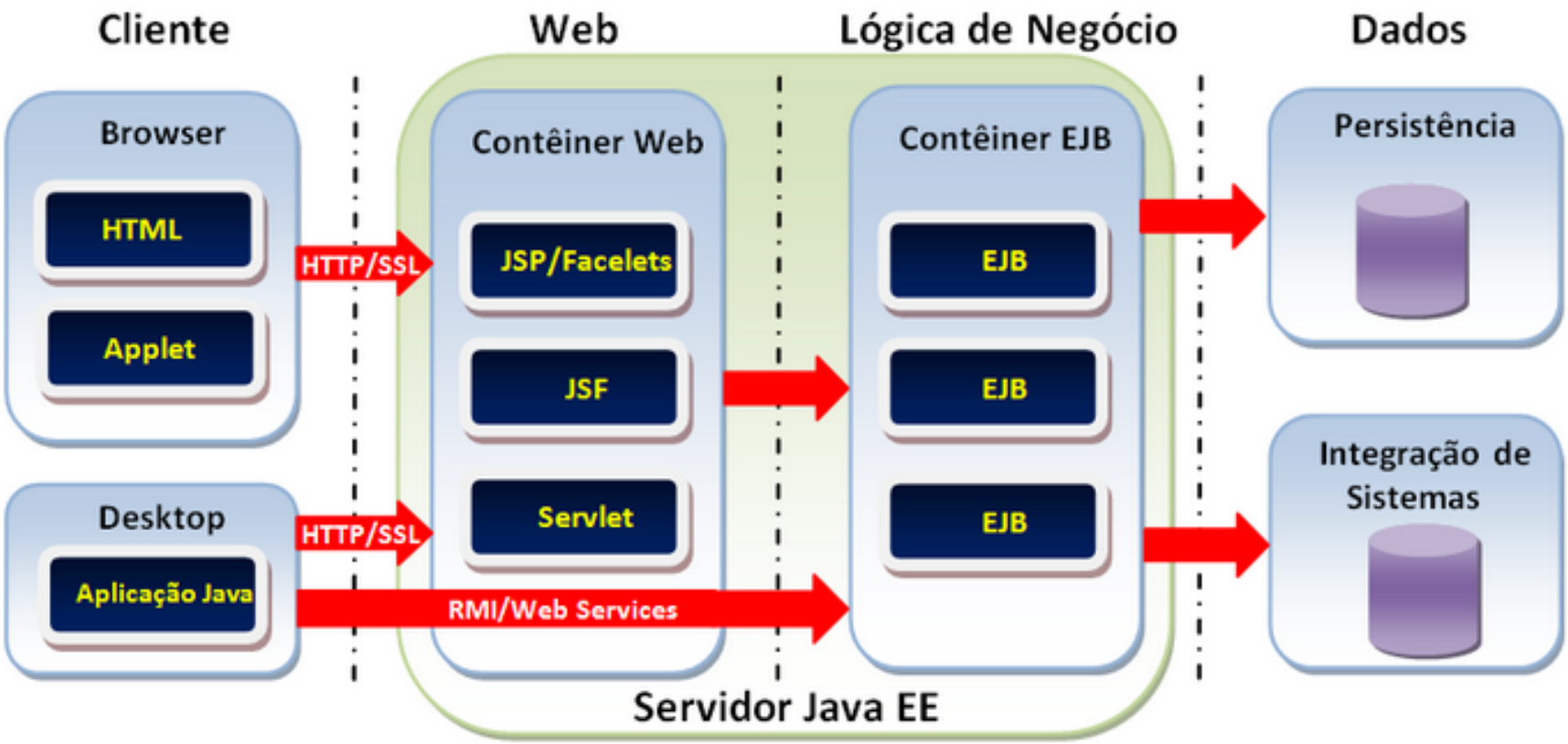
TRAJETÓRIA

- JPA Introdução
- JPA API
- Design Patterns e JUnit
- Relacionamentos
- JPQL
- Mapeamento Avançado
- Serialização de objetos e Sockets
- Remote Method Invocation
- Enterprise Java Beans

#10 - AGENDA

- Arquitetura de uma aplicação
- EJB Container
- Session Bean
- Interface Local e Remota
- EJB Stateless, Stateful e Singleton
- Acesso remoto

ARQUITETURA

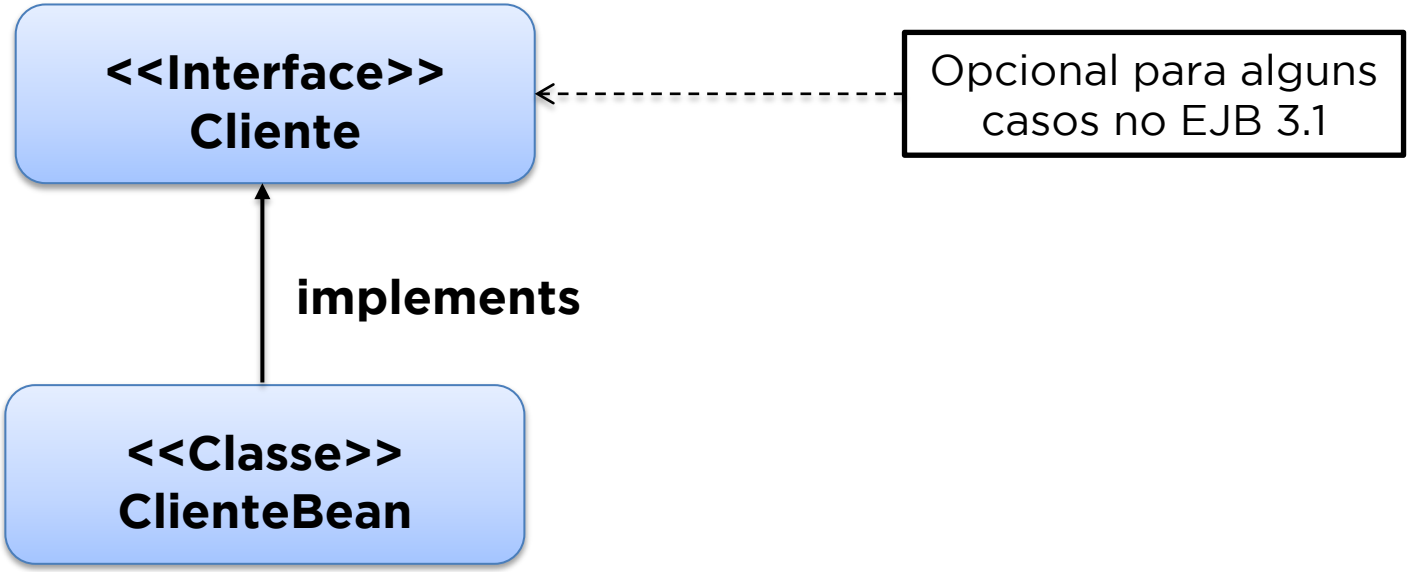


EJB CONTAINER

- Os **Session Beans** necessitam de um **EJB Container** para serem executados;
- Não é possível ter session beans no Tomcat, por exemplo;
- O **EJB Container** tem como atribuições:
  - Gerenciar o ciclo de vida dos EJBs
  - Controlar a concorrência de recursos;
  - Controlar as transações;
  - Persistência;
  - Segurança;
  - Garantir o desempenho da aplicação;

SESSION BEAN

- Um **Session Bean** é composto por dois elementos:
  - **Interface** (business interface) onde os métodos de negócio são declarados;
  - **Implementação** (bean class) onde os métodos declarados na interface são implementados;



BUSINESS INTERFACE

FIAP

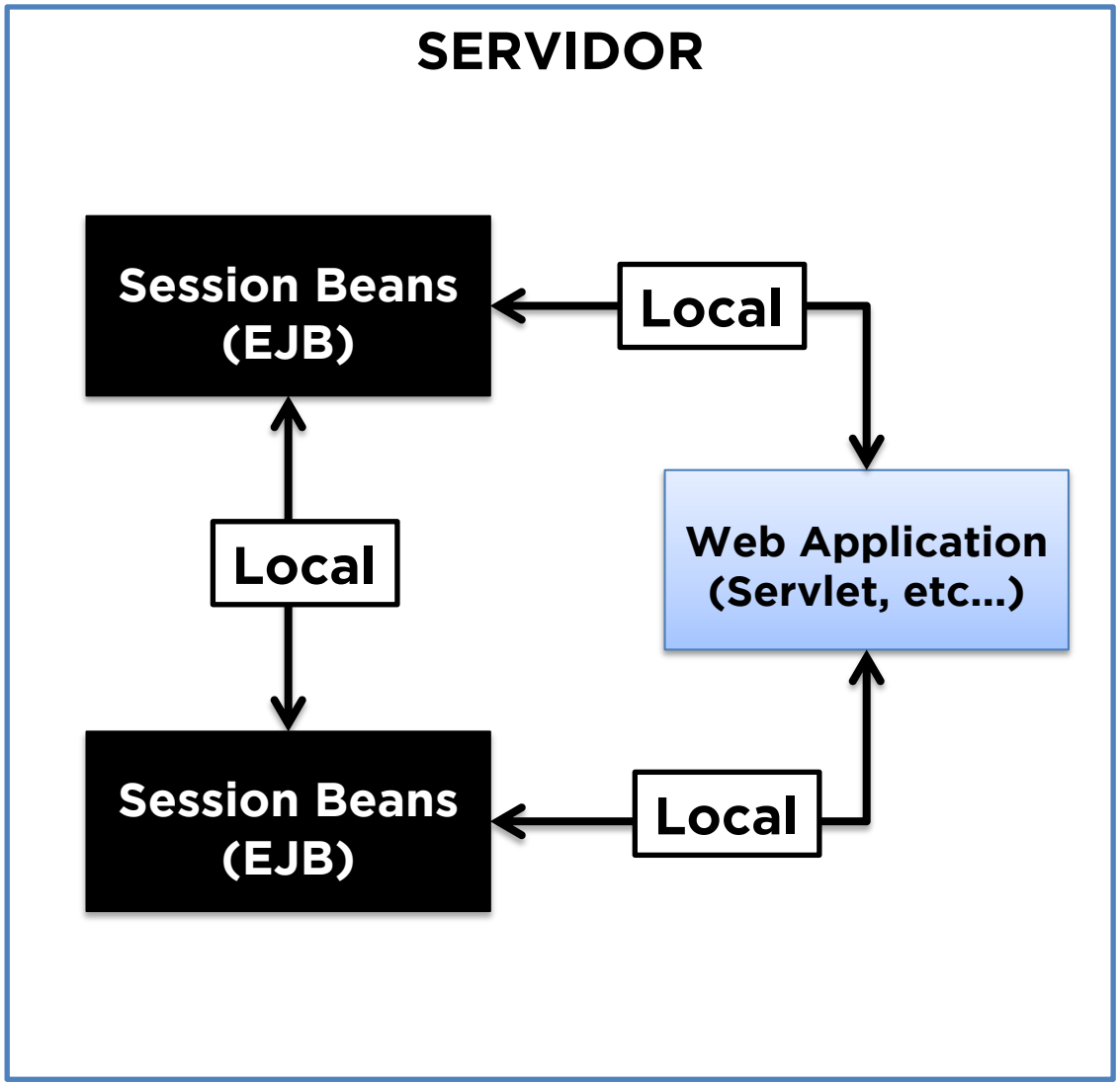
- Pode ser de um dos tipos abaixo:
  - **@Local** → somente clientes localizados na mesma instância do container (JVM) podem acessar o bean → Opcional para o EJB 3.1;
  - **@Remote** → tanto clientes localizados na mesma instância do container (JVM) quanto localizados fora podem acessar o bean;

ATENÇÃO!!!

- Uma mesma interface não pode ser ao mesmo tempo **@Remote** e **@Local**

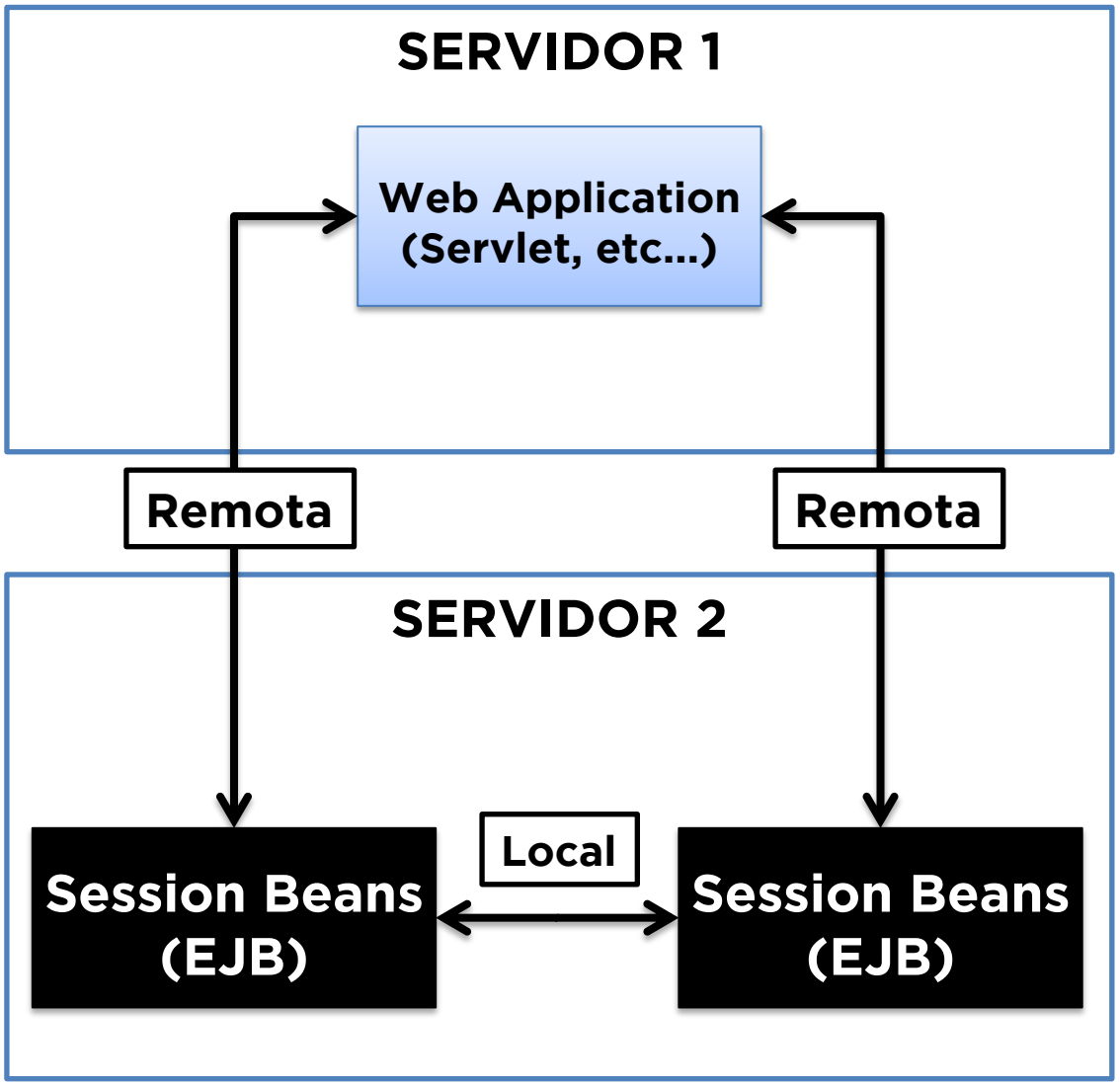
INTERFACE LOCAL

- Em um mesmo servidor é possível ter EJBs, JPA e aplicações Web;
- Os elementos dentro do mesmo servidor comunicam-se entre si por meio da interface **Local**;



INTERFACE REMOTA

- Os métodos declarados na interface Remota podem ser acessados por elementos em outros servidores;



BEAN (EJB) CLASS

FIAP

- Implementa a **business interface** onde os métodos de negócio são definidos;
- Pode ser de um dos tipos:
  - **@Stateless** → o mesmo EJB pode atender mais de uma requisição cliente (sem preservação do estado);
  - **@Stateful** → um único EJB para cada requisição cliente;
  - **@Singleton** → um único EJB para todos os clientes;

- Pode-se utilizar injeção de dependência para obter um **EntityManager**;
- Os métodos já são encapsulados em transações automaticamente.

PRIMEIRO EJB

- Abaixo temos um **EJB** simples:

```
import javax.ejb.Stateless;

@Stateless
public class Mensagem {
        public String getMensagem() {
                return "Teste";
        }
}
```

- A princípio, todo método declarado na implementação do EJB oferece automaticamente somente o acesso **local**;

- A anotação **@Stateless** indica que este é um EJB sem preservação do estado, isto é, uma mesma instância do EJB pode atender mais de uma requisição.

PRIMEIRO EJB

FIAP

- Componentes dentro do mesmo container do *EJB* local podem acessá-lo por meio de injeção de dependência;
- Para tanto, basta utilizar a anotação **@EJB**;
- Vejamos um exemplo abaixo de um *Servlet* acessando o **EJB Mensagem**:

```
@WebServlet("/EJBServlet")
public class EJBServlet extends HttpServlet {

   @EJB
   private Mensagem mensagem;

   protected void doGet(HttpServletRequest request,
   HttpServletResponse response) throws ServletException,
   IOException {
        response.getWriter().print(mensagem.getMensagem());
   }
}
```

STATELESS SESSION BEAN

FIAP

- Os ***Stateless Session Beans*** são reutilizados, isto é, uma mesma instância do EJB pode atender **mais de uma** requisição cliente;

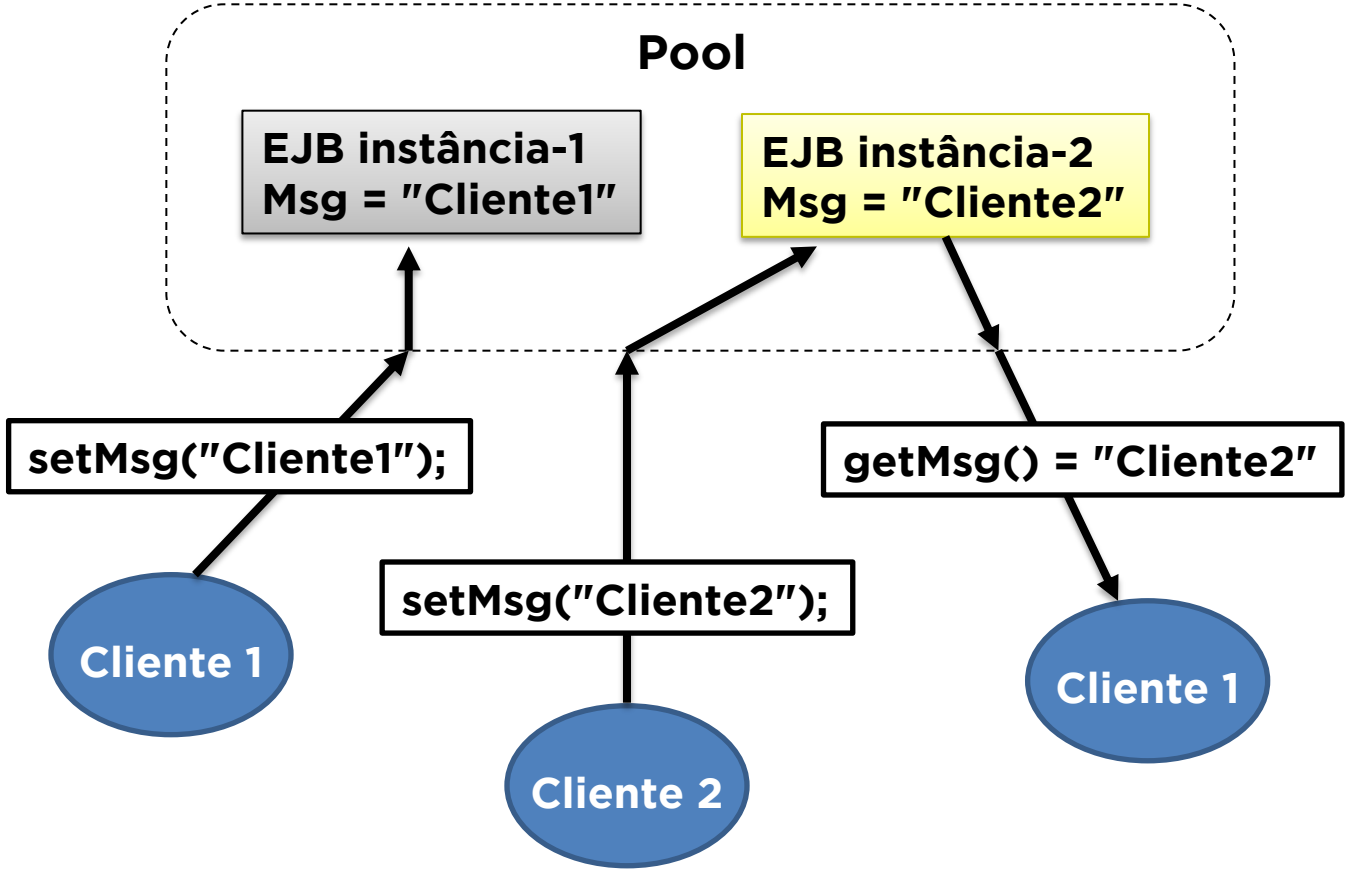
- Instâncias do mesmo EJB são mantidas em um ***pool*** implicando em um **baixo** uso de **memória** e **maior desempenho**;

- **Não** é garantido o estado (dados) entre duas ou mais requisições (chamadas de método);

- Basta anotar a *bean class* com **@Stateless**;

STATELESS SESSION BEAN

```
@Stateless
public class Mensagem {
        String msg = "";
}
```



STATEFUL SESSION BEAN

FIAP

- ***Stateful Session Beans*** **não** são reutilizados, isto é, o mesmo bean atende várias requisições porém do **mesmo** cliente, preservando o seu estado;
- Para cada requisição um *bean stateful* é **instanciado** e **dura** enquanto **durar a sessão com o cliente**, implicando no maior uso de memória e menor desempenho;
- Um *bean stateful* somente é **destruído** por ***time-out*** ou **explicitamente** pelo cliente;
- Basta anotar a *bean class* com **@Stateful**;
- É **possível preservar o estado** (dados) entre duas ou mais requisições (chamadas de método);

STATEFUL SESSION BEAN

FIAP

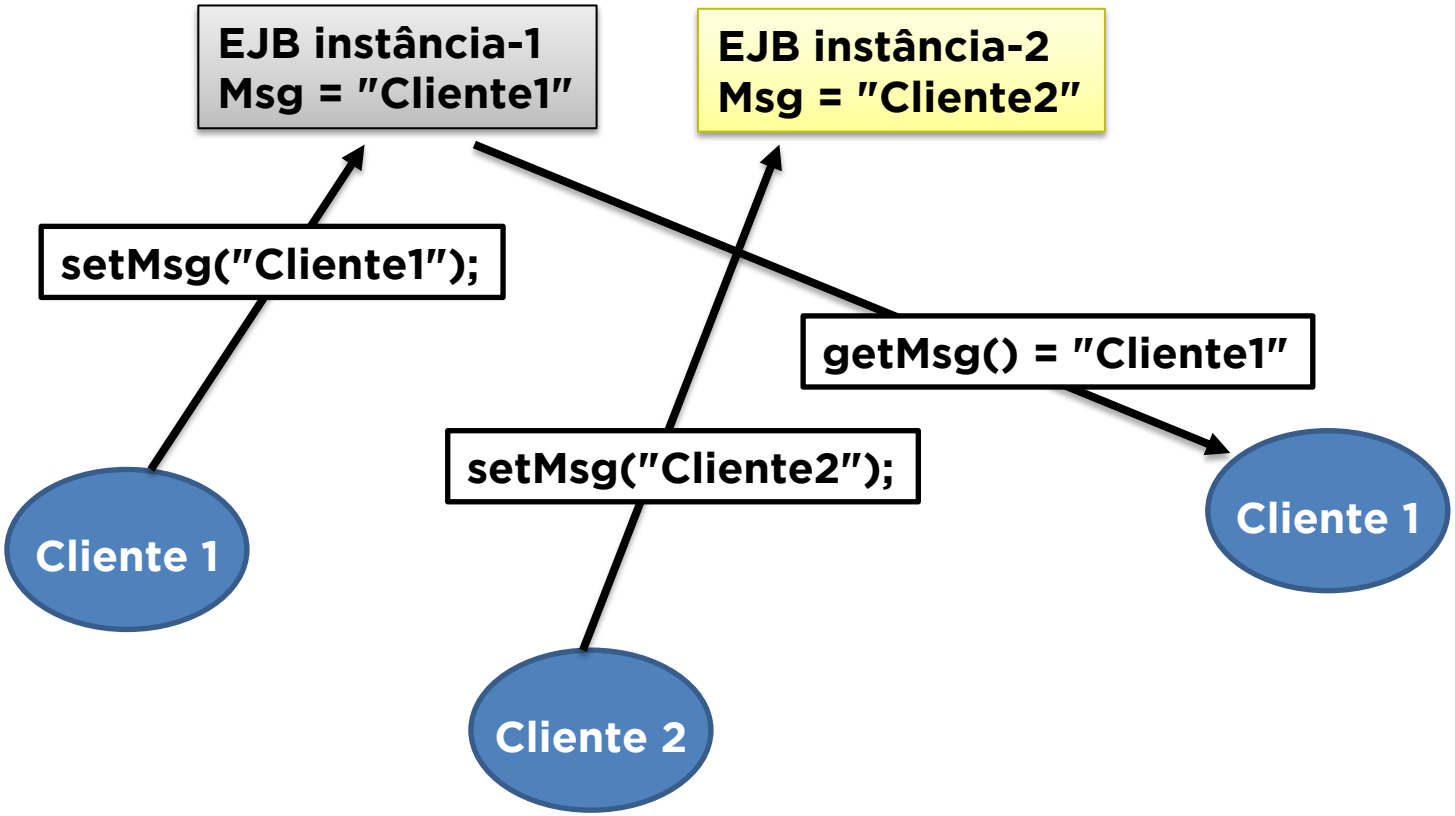
- Como lidar com a possibilidade de estouro de memória?

- Quando é necessário liberar memória, os beans não utilizados têm seu estado serializado (arquivo, banco) e são removidos da memória por meio de um processo denominado **Passivation**;

- Ao ser acionado novamente, o bean é novamente instanciado e tem seu estado desserializado por meio de um processo denominado **Activation;**

STATEFUL SESSION BEAN

```
@Stateful
public class Mensagem {
        String msg = "";
}
```



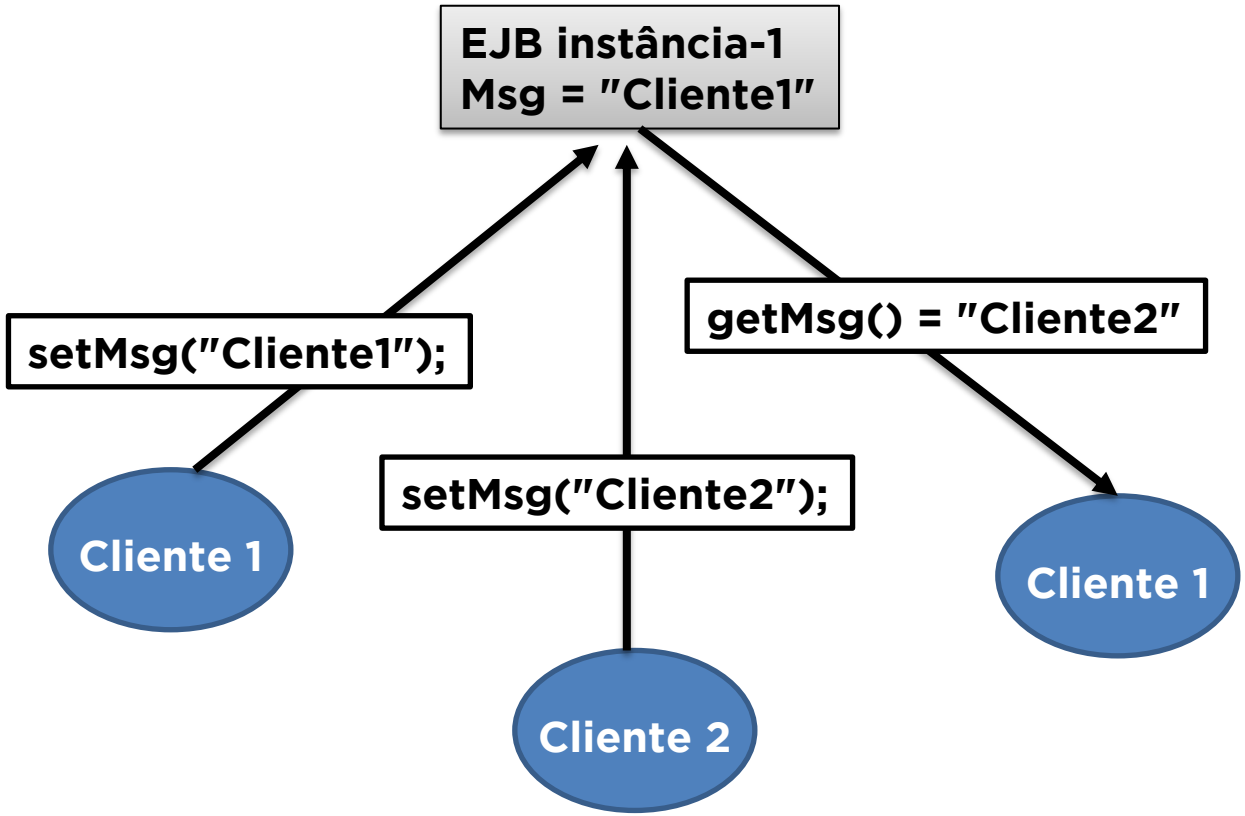
SINGLETON SESSION BEAN

FIAP

- Os ***Singleton Session Beans*** foram introduzidos na versão EJB 3.1;
- Implementam a ideia do *design pattern Singleton*;
- Uma única instância do *Singleton* EJB é compartilhada para todas as requisições cliente;
- Basta anotar a *bean class* com **@Singleton**;

SINGLETON SESSION BEAN

```
@Singleton
public class Mensagem {
        String msg = "";
}
```



ACESSO REMOTO

ACESSO REMOTO

FIAP

- Um EJB pode ser **acessado remotamente**, isto é, por elementos localizados em outros servidores;
- Para que isso seja possível devemos descrever os métodos remotos em uma interface com a anotação **@Remote**;
- Então a interface deve ser **implementada** pelo bean class do EJB;

```
@Remote
public interface MensagemRemota {
        String getMensagem();
}

@Stateless
public class Mensagem implements MensagemRemota { ... }
```

CLIENTE REMOTO

- Os ***session beans*** são localizados pelos clientes no servidor por meio de um nome;
- Para que aplicações cliente localizem determinado objeto é necessário o uso da **API *JNDI*** – *Java Name and Directory Interface*;
- É necessário primeiramente acessar o servidor e obter o chamado contexto inicial;
- Após obter o contexto inicial que contém o registro dos nomes dos objetos no servidor, basta efetuar localização do objeto desejado;
- **ATENÇÃO**: No lado cliente somente são necessárias as interfaces remotas dos componentes e objetos de serialização (ex: Entity Beans);

CLIENTE REMOTO

Para realizar o acesso remoto a um EJB será necessário, no cliente:

1. Definir algumas bibliotecas no *classpath* (específico do EJB container);
2. Obter as interfaces remotas dos componentes e também os objetos que serão serializados (Entity Beans, por exemplo);
3. No caso específico do Jboss 7.1 criar um arquivo de propriedades no src path chamado **jboss-ejb-client.properties**;
4. Acessar as interfaces remotas via JNDI. Para o JBoss:

```
ejb:<Nome EAR>/<Nome Projeto>/<Nome Alternativo>/<Nome Bean Class>!<Nome Completo da Interface Remota (com pacote)>
```

Obs: Nome Alternativo é opcional

CLIENTE REMOTO - EXEMPLO

```
public static void main(String[] args) {
        try {

            Properties p = new Properties();

            p.put(Context.URL_PKG_PREFIXES,
            "org.jboss.ejb.client.naming");

            final Context context = new InitialContext(p);

            MensagemRemote r = (MensagemRemote)
            context.lookup("ejb:EJBEar/EJBBasico//Mensagem
            !mensagem.MensagemRemote");

            System.out.println(r.getMensagem());

        } catch (NamingException e) {
             e.printStackTrace();
        }
}
```

VOCÊ APRENDEU..

- O que são EJB **Session Beans** e o **EJB Container**;
- Os tipos de interfaces de um EJB, **Remote** e **Local**;
- As classes EJB pode ser de três tipos: **Stateless**, **Stateful** e **Singleton**;
- **Utilizar** um EJB;
- Realizar uma chamada **Remota** de um EJB;

FIAP

"O sucesso nasce do querer, da determinação e persistência em se chegar a um objetivo. Mesmo não atingindo o alvo, quem busca e vence obstáculos, no mínimo fará coisas admiráveis"