Enterprise Java Beans

Java Platform, Enterprise Edition. The Java EE Tutorial
https://docs.oracle.com/javaee/7/JEETT.pdf

Java EE Annotations
http://www.physics.usyd.edu.au/~rennie/javaEEReferenceSheet.pdf

Creating and Running an Application Client on the GlassFish Server
https://netbeans.org/kb/docs/javaee/entappclient.html

Introduction to Developing Web Applications
https://netbeans.org/kb/docs/web/quickstart-webapps.html

Java Bean: Standard para definição de classes reutilizáveis

- Classe Java que:
  - tem todas as propriedades privadas
  - métodos públicos (getters e setters) acedem às propriedades
  - tem um construtor sem parâmetros
  - implementa a interface Serializable
- POJO (Plain Old Java Object) serializável.

- Arquitectura Enterprise Java Beans (EJB)
- Session Beans
  - Stateless
  - Stateful
  - Singleton
- Tipos de acesso a um EJB
- Message Driven Beans

Enterprise Java Beans (EJB)

Conceito

- Componente executado do lado do servidor que implementa a lógica da aplicação / negócio

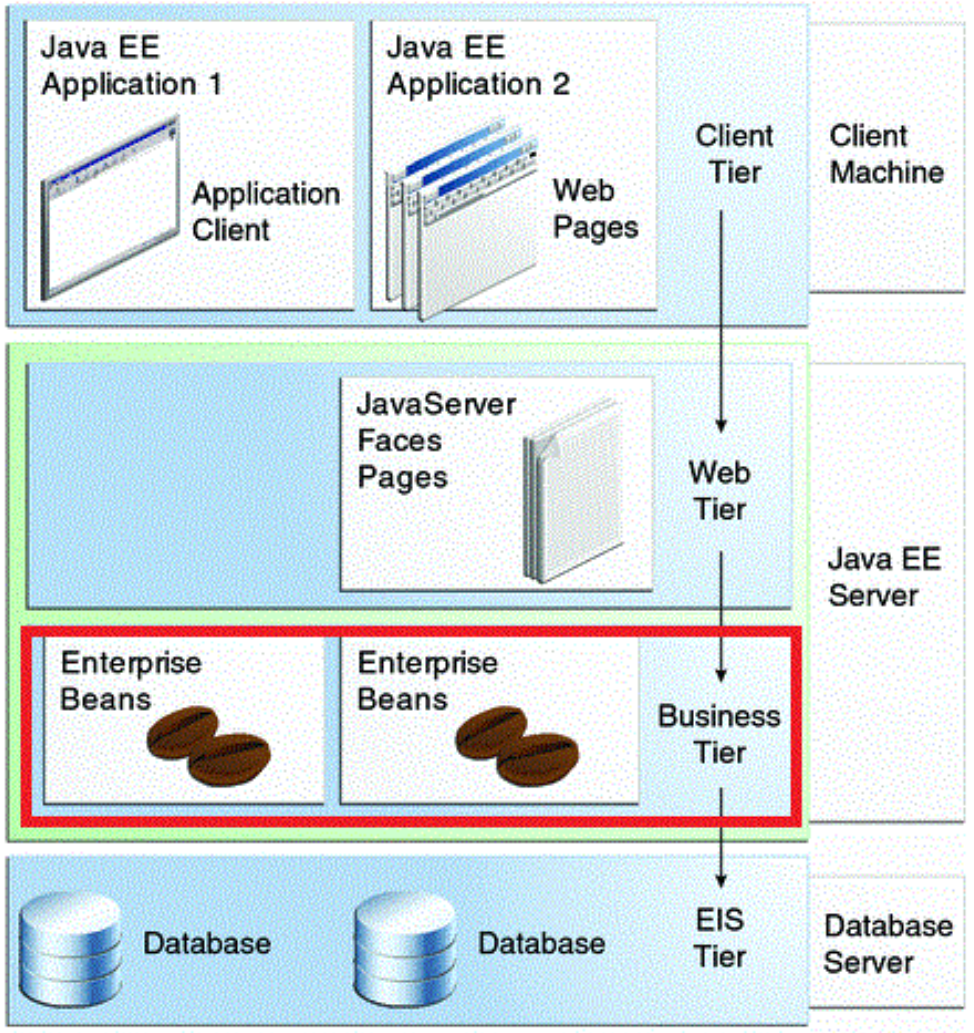
- O EJB *container* oferece os serviços do sistema para:
  - Gestão de transações
  - Segurança
  - Concorrência
  - Escalabilidade

Enterprise Java Beans (EJB)

- O código da aplicação deve poder executar em qualquer plataforma que cumpra as especificações EJB

(e.g. GlassFish, Jboss (WildFly), WebSphere, etc.)

Arquitetura JEE – Camada EJB



Tipos de EJB

- Session Beans

- Implementam um tarefa para a aplicação cliente.
- Opcionalmente podem implementar um *Web service*.

  - Stateful
  - Stateless
  - Singleton

Tipos de EJB

- Message-Driven Beans

- Atuam como *Listeners* para um determinado tipo de mensagens

(e.g. mensagens da Java Message Service API)

Acesso a *session* Beans (1)

- O cliente de um *session* Bean obtém a referência para uma instância do Bean por:

  - *Dependency injection (usando anotações da linguagem)*

  ou

  - *JNDI* (Java Naming and Directory Interface) *lookup*

- O cliente tem acesso a um *session* Bean através dos métodos de uma interface ou através dos métodos públicos do Bean.

Acesso a *session* Beans (1)

- 3 tipos de acesso:

  - *Local*
  - *Remoto*
  - *Como web service*

Acesso a *session* Beans (2)

- **Local** – contexto local (JVM) do servidor

```
@Local
public interface ExampleLocal { …  }
```

- **Remote** – interface remota (dentro ou fora da JVM)

```
@Remote
public interface ExampleRemote { …  }
```

Acesso a *session* Beans (2)

- **Web Service** – serviço sobre HTTP

```
@WebService
public interface InterfaceName { …  }
```

Acesso a *session* Beans (3)

- Abordagem clássica através do serviço JNDI:

```
ExampleLocal example = (ExampleLocal)
        InitialContext.lookup
("java:module/ExampleLocal");
```

```
ExampleRemote example = (ExampleRemote)
        InitialContext.lookup
("java:global/ExampleRemote");
```

Acesso a *session* Beans (3)

- Por *dependency injection*

```
@EJB
Example example
```

O habitual *lookup* pode ser substituído pela anotação @EJB em que o servidor JEE implicitamente “injecta” o código para obter o EJB referenciado

Stateful Session Beans

Características

- Cada instância de um Stateful Bean está associada a um único cliente.
- O Bean mantém o estado de sessão (não partilhado) com um dado cliente (estado = valores das variáveis de instância)
- Permitem guardar informação do cliente entre múltiplas invocações
- Possuem um tempo de vida (configurável) até serem removidos

Stateful Session Beans – Exemplo (1)

- Interface remota

```
@Remote
```

Acesso remoto

```
public interface Cart {
 public void initialize(String person) throws BookException;
 public void initialize(String person, String id)throws BookException;

 public void addBook(String title);
 public void removeBook (String title) throws BookException;
 public List<String> getContents();

 public void remove();
}
```

Método que permite ao cliente remover a instância do Bean. Na implementação o método terá a anotação @Remove

Stateful Session Beans – Exemplo (2)

- Stateful Bean


```
@Stateful
public class CartBean implements Cart {
   private List<String> contents;
   private String customerId; private String customerName;

    public void initialize(String person) throws BookException {
        if (person == null) {
            throw new BookException("Null person not allowed.");
        } else {
            customerName = person;
        }
        customerId = "0";
        contents = new ArrayList<String>();
    }  …
```

Stateful Session Beans – Exemplo (3)

- Cliente

```
public class CartClient {
    @EJB
    private static Cart cart;

    public CartClient(String[] args) {
    }

     public static void main(String[] args) {
        CartClient client = new CartClient(args);
        client.doTest();
    }

    public void doTest() {                          …
```

O cliente obtém a referência para o Bean por dependency injection

Stateful Session Beans – Exemplo (4)

- Invocação do métodos do Bean

```
public class CartClient {
    …
 cart.initialize("Duke d'Url", "123");
 cart.addBook("Infinite Jest");
 List<String> bookList = cart.getContents();
 Iterator<String> iterator = bookList.iterator();
 while (iterator.hasNext()) {
        String title = iterator.next();
        System.out.println("Retrieving book title from cart: " + title);
 }
 System.out.println("Removing \"Gravity's Rainbow\" from cart.");
 cart.removeBook("Gravity's Rainbow");
 cart.remove();
 …
```

Remoção do Bean após execução do método

Stateless Session Beans

Características

- Não mantêm informação especifica de um cliente;
- O estado existe apenas durante a invocação de um método;
- O servidor gere uma *pool* de instâncias que servem pedidos de vários clientes;
- Interface pode ser exposta como *web service*;

Stateless Session Beans – Exemplo (1)

- Stateless Bean

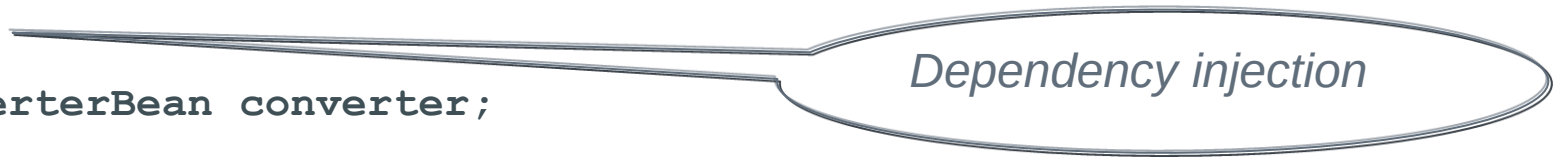

```
@Stateless
public class ConverterBean {
    private BigDecimal euroRate = new BigDecimal("0.0100169");
    private BigDecimal yenRate = new BigDecimal("79.3916");
    public BigDecimal dollarToYen(BigDecimal dollars) {
        BigDecimal result = dollars.multiply(yenRate);
        return result.setScale(2, BigDecimal.ROUND_UP);
    }
    public BigDecimal yenToEuro(BigDecimal yen) {
        BigDecimal result = yen.multiply(euroRate);
        return result.setScale(2, BigDecimal.ROUND_UP);
    }
}
```

Stateless Session Beans – Exemplo (2)

- Cliente Web

```
@WebServlet
public class ConverterServlet extends HttpServlet {
    @EJB
    ConverterBean converter;
    …
     BigDecimal yenAmount = converter.dollarToYen(d);

    BigDecimal euroAmount = converter.yenToEuro(yenAmount);
    out.println("<p>" + amount + " dollars are "
                        + yenAmount.toPlainString() + " yen.</p>");
    out.println("<p>" + yenAmount.toPlainString() + " yen are "
                        + euroAmount.toPlainString() + "Euro.</p>");
}
```



Singleton Session Beans

Características

- Instanciado apenas uma vez por aplicação
- Estado partilhado entre todos os clientes
- Estado preservado entre invocações
- Anotação *@Startup* indica que deve ser instanciado no arranque da aplicação

Exemplo:

```
@Singleton
public class CounterBean {
    private int hits = 1;
    // Increment and return the number of hits
    public int getHits() {
    return hits++;
}
```

Message Driven Beans

Características

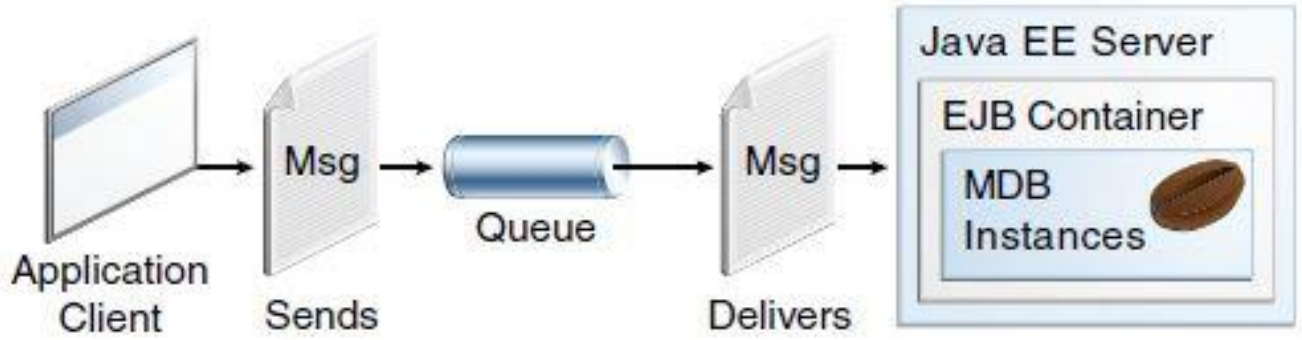
- Consiste num receptor **assíncrono** de mensagens Java
- Não mantém estado
- Um MDB pode processar mensagens de múltiplos clientes

Message Driven Beans

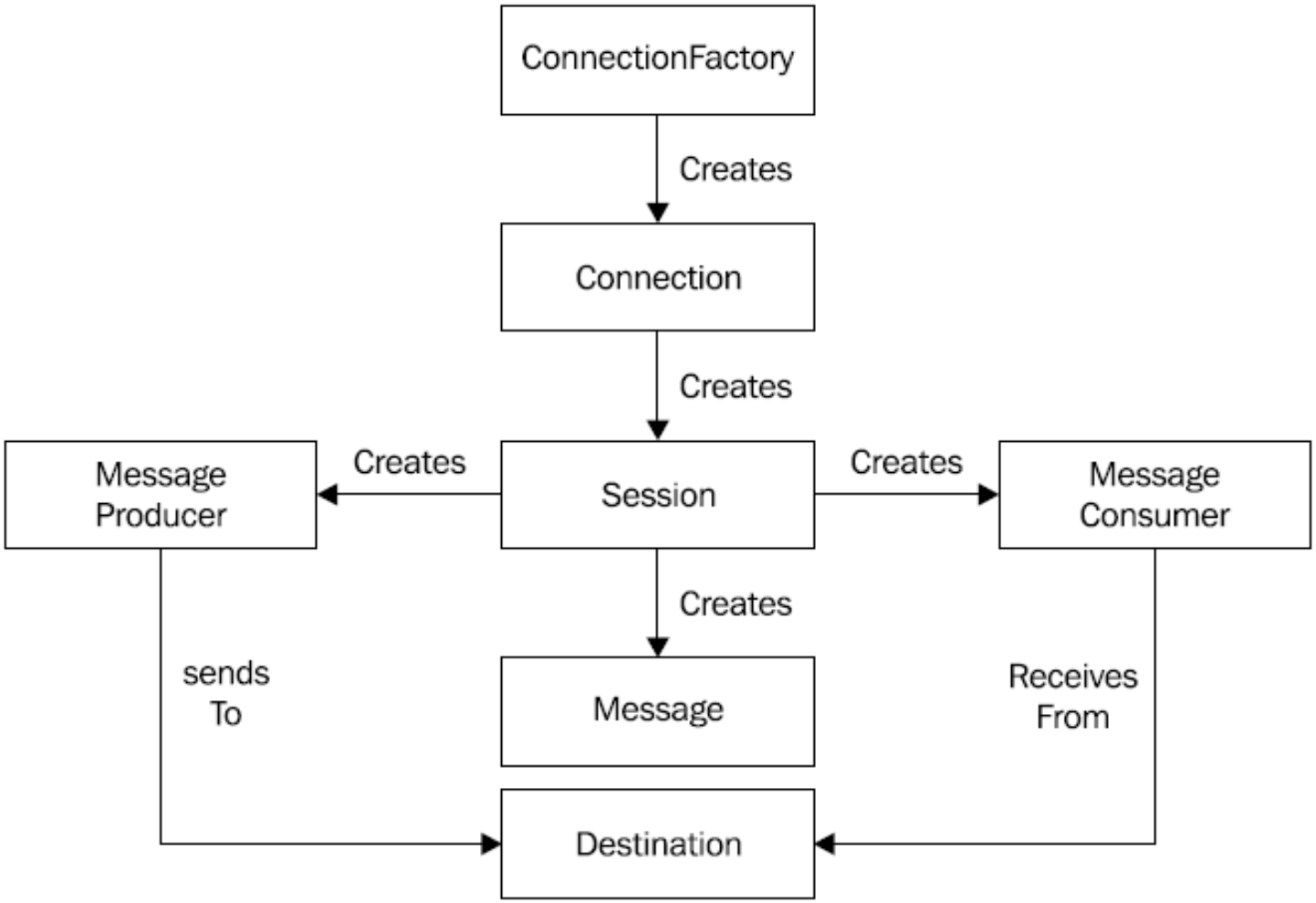
Características

- O cliente não acede diretamente à interface do MDB, usa um serviço de mensagens, e.g.,

  JMS – Java Messaging Service;



JMS - Arquitectura



Message Driven Beans – Exemplo (1)

- As mensagens do cliente são enviadas para uma *queue* JAVA e posteriormente entregues ao MDB;
- O nome da *queue* é indicado na anotação *@MessageDriven* do MDB que se torna automaticamente (sem código adicional) num consumidor;

Ex.Io:

```
@MessageDriven(mappedName = "jms/Queue", activationConfig =  {
      @ActivationConfigProperty(propertyName = "acknowledgeMode",
                                propertyValue = "Auto-acknowledge")
    , @ActivationConfigProperty(propertyName = "destinationType",
                                propertyValue = "javax.jms.Queue")
}  )

public class SimpleMessageBean implements MessageListener {

  static final Logger logger Logger.getLogger("SimpleMessageBean");
...
```

Message Driven Beans – Exemplo (2)

- O método *onMessage do MDB* é invocado quando uma mensagem é recebida;
- A sua implementação contém o processamento da mensagem recebida;

```
public void onMessage(Message inMessage) {
      TextMessage msg = null;
      try {
        if (inMessage instanceof TextMessage) {
          msg = (TextMessage) inMessage;
          logger.info("MESSAGE BEAN: Message received: " + msg.getText());
        } else {
          logger.warning(  "Message of wrong type: " +
                           inMessage.getClass().getName());
        }
      } catch (JMSException e) {
    …
```

Message Driven Beans – Cliente (1)

- O cliente precisa dos recursos:
  - JMS connection factory
  - JMS destination

```
public class SimpleMessageClient {
    static final Logger logger = Logger.getLogger("SimpleMessageClient");
    @Resource(mappedName = "jms/ConnectionFactory")
    private static ConnectionFactory connectionFactory;
    @Resource(mappedName = "jms/Queue")
    private static Queue queue;

    public static void main(String[] args) {
        Connection connection = null;
        Session session;
        MessageProducer messageProducer;
…
```

Message Driven Beans – Cliente (2)

```
...
  try {
     connection = connectionFactory.createConnection();
     session = connection.createSession(false,
                          Session.AUTO_ACKNOWLEDGE);
     messageProducer = session.createProducer(queue);
     message = session.createTextMessage();

     for (int i = 0; i < NUM_MSGS; i++) {
         message.setText("This is message " + (i + 1));
         System.out.println("Sending message: " +
         message.getText());
         messageProducer.send(message);
     }
```