Em resumo… por agora!

- Aplicações multi-tier:
  - Presentation Layer
    - Colecção de componentes que implementam a UI. Utilizam-se servlets, JSPs, CSS, etc.
  - Business Layer
    - EJBs e Managed Beans
  - Persistence Layer
    - Serialized Java beans, ORM, Hibernate, etc.

Arquitectura do EJB

- Componentes EJB server side
  - **Session Beans**: parte da aplicação que faz a gestão de processos ou de tarefas. Implementam a lógica de negócio que estabelece os relacionamentos entre as entidades.
    - Use Cases correspondem (de alguma forma) a métodos dos Session Beans.
  - **Message Driven Beans**: são necessários para desenvolver a parte de coordenação de diálogo entre outros Session Beans e as entidades. Processam mensagens assíncronamente de JMS, sistemas legados ou mesmo Web Services.

- A actividade descrita num Session Bean ou num Message Driven Beam é transiente
  - Inicia-se, realiza-se e termina (é uma tarefa!)

EJB: Classes e Interfaces

- Para a criação de Session ou Message Driven beans é necessário definir:
  - A interface remota
    - Conjunto de métodos (de negócio) que podem ser acedidos por aplicações fora do container de EJB. É uma interface Java anotada com a tag `@javax.ejb.Remote`
  - A interface local
    - Define os métodos que podem ser invocados por outros beans existentes no mesmo container. Tem a anotação `@javax.ejb.Local`
  - Endpoint interface
    - Define os métodos de negócio que podem ser acedidos fora do contentor via tecnologia SOAP. Trata-se de JAVA XML-RPC e tem como objectivo ser compatível com os standard SOAP e WSDL.
    - Anotada com `@javax.ejb.WebService`

EJB: Classes e Interfaces

- Interface de Mensagem
  - Define os métodos através dos quais pode ter ligação a frameworks de messaging como o JMS

- Bean Class
  - Classe com a lógica que o bean representa
  - A classe implementa a lógica de negócio associada e tem pelo menos um dos interfaces atrás referidos.
  - O bean deve ser anotado com as tags `@javax.ejb.Stateful` ou `@javax.ejb.Stateless` consoante a sua natureza.

Beans: Exemplo Simples

- Definição de interface remota

```
import javax.ejb.Remote;

@Remote
public interface CalculatorRemote {
    public int add(int x, int y);
    public int subtract(int x, int y);
}
```

- Classe que implementa a interface

```
import javax.ejb.*;

@Stateless
public class CalculatorBean implements CalculatorRemote {
    public int add(int x, int y) {
        return x + y;
    }
    public int subtract(int x, int y) {
        return x - y;
    }
}
```

EJB Container

- Os session bean declaram interfaces que os clientes invocam
  - As aplicações clientes utilizam objectos do tipo da interface pretendida
- Os clientes fora do mesmo container invocam a interface remota
  - ou utilizam Web Services
- Clientes dentro do mesmo sistema Java EE podem utilizar a interface local, desde que estejam a correr na mesma máquina virtual.

- A arquitectura do EJB tem três componentes importantes
  - o container de beans
  - o proxy stub
  - as instâncias de bean

EJB Container

- Proxy stub:
  - quando um cliente invoca um método num session bean, não o faz directamente na instância
  - A invocação é feita ao interface remoto ou local do bean
  - Os pedidos são respondidos por um proxy stub
    - que encaminha os pedidos remotos para um contentor de beans remoto
    - encaminha as invocações da interface local para um container de beans local à virtual machine
  - Por exemplo, no caso da família de servidores JBOSS, este proxy é gerado dinamicamente em tempo de deployment. Utiliza os serviços de `java.lang.reflect.Proxy`
- EJB container
  - gere as instâncias de bean que estão contidas
  - Fornece serviços de segurança, implementação de transações, cache, etc.

EJB Container

- O container agrega a informação fornecida
  - Nas anotações existentes em cada ficheiro Java
  - Nos descritores XML
- Baseado nessa informação efectua a gestão necessária para
  - Efectuar autenticação
  - Invocar transacções
  - Gerir o ciclo de vida de um bean
  - Funcionar como mecanismo de middleware, no encaminhamento de pedidos às interfaces locais e remotas

Session e Entity Beans

- Exemplo: um TravelAgentBean cria uma reserva para um Cliente (implementa a funcionalidade expressa no Use Case)

- Aplicação cliente (excerto)

```
// Get the credit card number from the text field.
String creditCard = textField1.getText( );
int cabinID = Integer.parseInt(textField2.getText( ));
int cruiseID = Integer.parseInt(textField3.getText( ));

Customer customer = new Customer(name, address, phone);

// Create a new TravelAgent session, passing in a reference to a
// customer entity bean.
TravelAgentRemote travelAgent = ...; // Use JNDI to get a reference
travelAgent.setCustomer(customer);
// Set cabin and cruise IDs.
travelAgent.setCabinID(cabinID);
travelAgent.setCruiseID(cruiseID);

// Using the card number and price, book passage.
// This method returns a Reservation object.
Reservation res = travelAgent.bookPassage(creditCard, price);
```

Session e Entity Beans

- TravelAgent Bean
  - A implementação da funcionalidade de camada de negócio

```
@Stateful
public class TravelAgentBean implements TravelAgentRemote {
    @PersistenceContext private EntityManager entityManager;
    @EJB private ProcessPaymentRemote process;

    private Customer customer;
    private Cruise cruise;
    private Cabin cabin;

    public void setCustomer(Customer cust) {
        entityManager.create(cust);
        customer = cust;
    }
    public void setCabinID(int id) {
       cabin = entityManager.find(Cabin.class, id);
    }
    public void setCruiseID(int id) {
       cruise = entityManager.find(Cruise.class, id);
    }

    public Reservation bookPassage(String card, double price)
        throws IncompleteConversationalState {
        if (customer == null ||cruise == null ||cabin == null){
            throw new IncompleteConversationalState( );
        }
        try {

            Reservation reservation =
                new Reservation(customer,cruise,cabin,price,new Date( ));

            entityManager.persist(reservation);

            process.byCredit(customer,card,price);


            return reservation;
        }catch(Exception e){
            throw new EJBException(e);
        }
    }

}
```

Em resumo

- Beans são componentes de lógica de negócio ou entidades
- Session beans tem interfaces remotas ou locais, para efectuar serviço às aplicações cliente
  - Message driven beans e entity beans não tem interface de serviço locais ou remotas
- Três tipos de beans:
  - Entity: são persistentes e correspondem às entidades do modelo de domínio
  - Session: são pontos de contacto das aplicações cliente e implementam as tarefas decorrentes da lógica de negócio
  - Message: são pontos de integração com recurso a envio de mensagens
- O proxy stub é um construtor conceptual que gere invocações em session beans
  - As aplicações cliente não falam com os beans, mas com o proxy

EJB: serviços primários

- Cenário típico de aplicações baseadas em JEE:
  - Sistema com muitos utilizadores, isto é, muitas instâncias de aplicações cliente
  - Milhares de objectos criados e em utilização
  - Muitas interacções entre os objectos (por forma a descrever a lógica de negócio)
  - Concorrência e operações com requisitos transaccionais

- O servidor de EJB tem de lidar com esta complexidade e
  - Regular, sincronizando, as interacções entre objectos
  - Partilhar recursos entre os diversos componentes
    - Exemplo típico: acesso a base de dados (pool de connections)