

# Enterprise Java Beans (2)

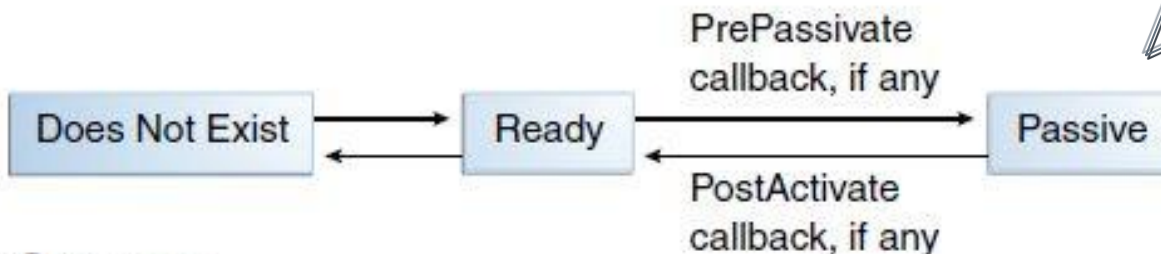
- O ciclo de vida de um EJB
- O serviço EJB timer

# Ciclo de vida de um EJB (1)

## ■ Stateful Session Bean

- ① Create
- ② Dependency injection, if any
- ③ PostConstruct callback, if any
- ④ Init method, or ejbCreate<METHOD>, if any

*O cliente obtém a referência para o Bean*



*O Bean é removido da memória para o disco*

- ① Remove
- ② PreDestroy callback, if any

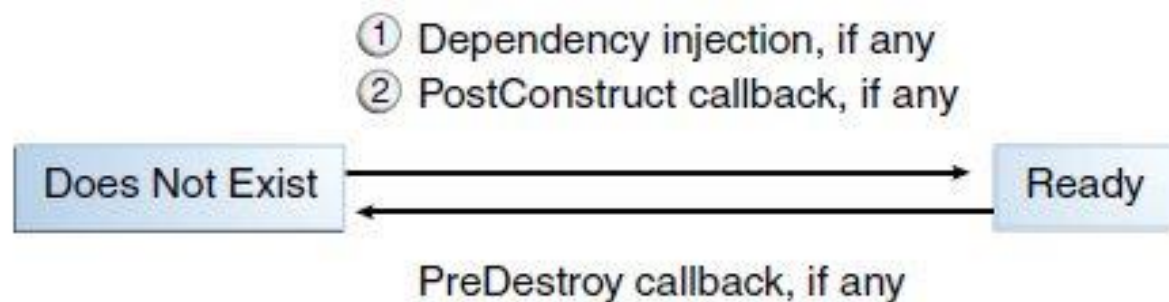
*Único método invocado pelo utilizador*

## Ciclo de vida de um EJB (2)

### ■ Stateless Session Bean

(O EJB *container* cria uma pool de stateless beans)

- Para cada instância:



### ■ Singleton Session Bean

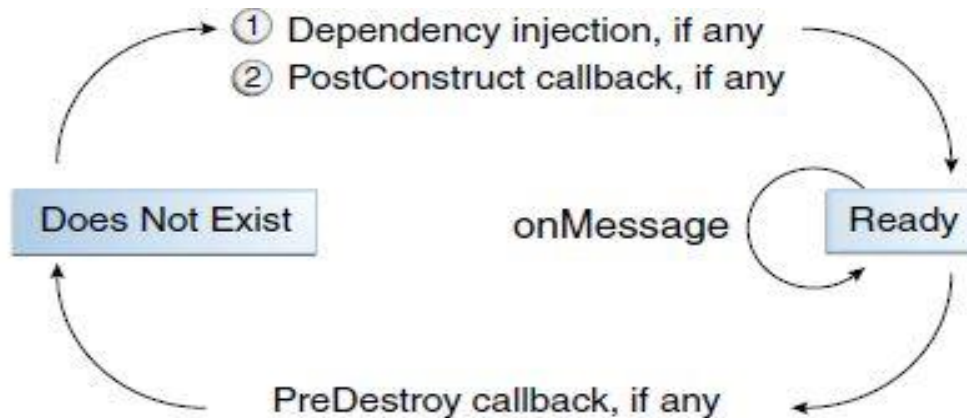
- Se o Singleton bean tem a anotação @Startup, a única instância do Bean é iniciada pelo *container* quando é feito o deploy da aplicação.

- Possui os mesmos estados que um Stateless bean.

## Ciclo de vida de um EJB (3)

### ■ Message Driven Bean

- O EJB *container* cria uma pool de MDBs



# Timers

- O serviço de Timer do contentor de JEBs permite:
  - Programar no tempo notificações para todos os tipos de EJB com **exceção de Stateful Session Beans**
- São possíveis notificações que ocorrem:
  - De acordo com um determinado calendário
  - Num tempo específico (e.g., *12 de Setembro, 9:00 a.m.*)
  - Após um certo período de tempo (e.g., *dentro de 4 dias*)
  - Em intervalos de tempo (e.g., *cada 3 minutos*)

# Timers

- Os Timers podem ser programados ou automáticos
- Timers programados são criados por invocação de um dos métodos da interface `TimerService`
- Quando expira o tempo de um timer programado é executado o método anotado com `@Timeout`
- e.g.,

```
@Timeout  
public void timeout(Timer timer) {  
    System.out.println("TimerBean: timeout occurred");  
}
```

# Timers

## ■ Criar um Timer programado (1)

■ `long duration = 60000;`

`Timer timer =`

```
    timerService.createSingleActionTimer(duration,  
                                         new TimerConfig());
```

*Expira em 1 minuto  
(60000 milissegundos)*

■ `SimpleDateFormat formatter =`

```
    new SimpleDateFormat("MM/dd/yyyy 'at' HH:mm");
```

```
Date date = formatter.parse("13/09/2013 at 18:00");
```

```
Timer timer = timerService.createSingleActionTimer(date,  
                                                    new TimerConfig());
```

*Expira numa data específica*

# Timers

- Criar um Timer programado (2)



*Timer baseado num  
calendário*

```
ScheduleExpression schedule = new ScheduleExpression();
```

```
schedule.dayOfWeek("Mon");
```

```
schedule.hour("12-17, 23");
```

```
Timer timer = timerService.createCalendarTimer(schedule);
```



# Timers

- Criar um Timer programado (3)

Por omissão os Timers são persistentes.

- Se o servidor falha, o timer fica guardado em memória persistente e reativado quando o servidor recupera.
  - Se o timer expirar enquanto o servidor estiver inativo, o método `@Timeout` é invocado, após o restart do servidor.
- Para um Timer não persistente: `TimerConfig.setPersistent(false)` ;

# EJB Timers

- Criar um Timer automático (1)
  - Timers automáticos são criados após o deploy de um Bean que contém um método anotado com
    - `java.ejb.Schedule` ou `java.ejb.Schedules`
  - Um Bean pode ter vários timers automáticos (os timers programados são únicos por bean)
  - Um método anotado com `@Schedule` funciona como um método de timeout para o calendário especificado nos atributos da anotação.

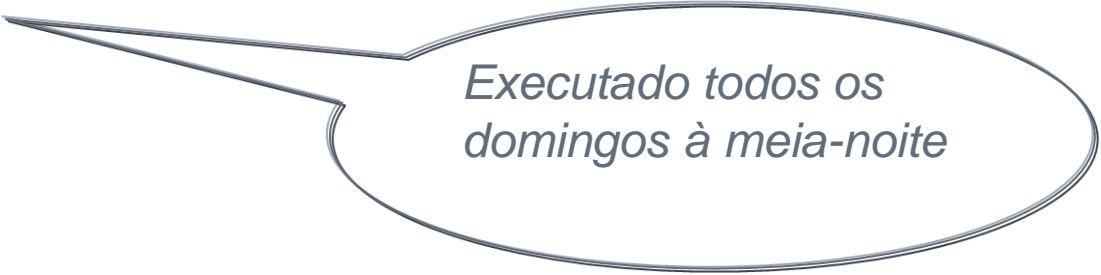
# EJB Timers

- Criar um Timer automático (2)

- e.g.

```
@Schedule(dayOfWeek="Sun", hour="0")
```

```
public void cleanupWeekData() { ... }
```



*Executado todos os domingos à meia-noite*

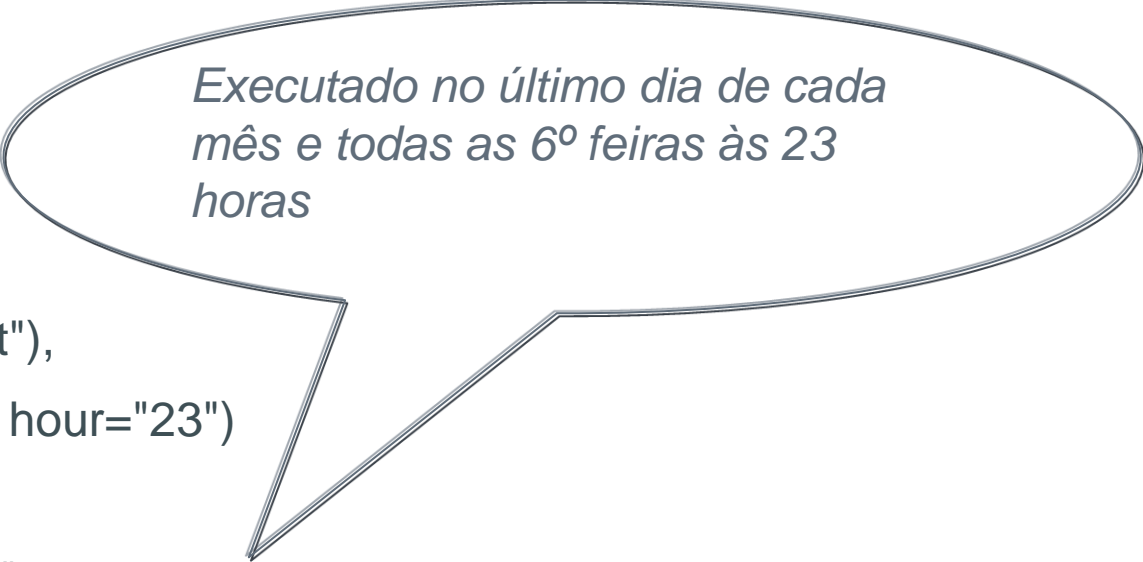
# EJB Timers

- Criar um Timer automático (2)

- Usar Schedules para especificar vários calendários para o mesmo método.

- e.g.

```
@Schedules ({  
  @Schedule(dayOfMonth="Last"),  
  @Schedule(dayOfWeek="Fri", hour="23")  
})  
public void doPeriodicCleanup() { ... }
```



*Executado no último dia de cada mês e todas as 6<sup>o</sup> feiras às 23 horas*

# EJB Timers

## ■ Especificação de Intervalos

- Numa expressão da forma  $x/y$ ,  
x o ponto de partida e y o intervalo.

O wildcard (\*) pode ser usado na posição x e equivale a  $x=0$ .

e.g.

minute="\*/10" significa ,todos os 10 minutos começando às 0 horas.

hour="12/2" significa todas as duas horas a começar ao meio-dia.