EJB

Distributed application engineering

Short Introduction to EJB

- Introduction
- Sessions Beans
- Web Services
- Message Driven Beans
- Annotation/Interception

History

- First Specification1997 – 1.0, 1.1 en 99
  - Definition of rôles and views
  - XML configuration
  - Entity Beans support
- EJB 2.0 – 2001
  - New kind of beans
  - Portability
- EJB 3.0 – 2006
  - Simplification
  - Annotation
- EJB 3.1 - 2009
  - More simplification – Configuration by default
- EJB 3.2 – 2013
  - Small evolutions

What is an EJB

- A Server Side Component
  - Written in Java
  - Running in an EJB container
  - Configurable

Goal of EJB

- Simplify the writing of portable Software Components
- Make distribution more transparent
- Consider
  - Development
  - Deployment
  - Execution

Bean Types

- Session Beans
  - Used for the business logic of the application
  - Stateless
  - Stateful
- Message Driven Beans
  - Used for asynchronous conversation
- Web Service Beans

Les Sessions Beans

- Contain the business logic of the application
- Stateless ou Stateful
- A Session Bean has
  - An implementation
  - An interface (or not)

Interface Locale/Remote

- EJB can be accessed localy or remotely
  - Remote : access from a distant client (RMI/IIOP)
    - Follow the RMI rules
    - RemoteException
    - Serialisation of parameters
    - Parameter passing by value
  - Local : access from a local client (e.g. a servlet)

Un Stateless Session Bean

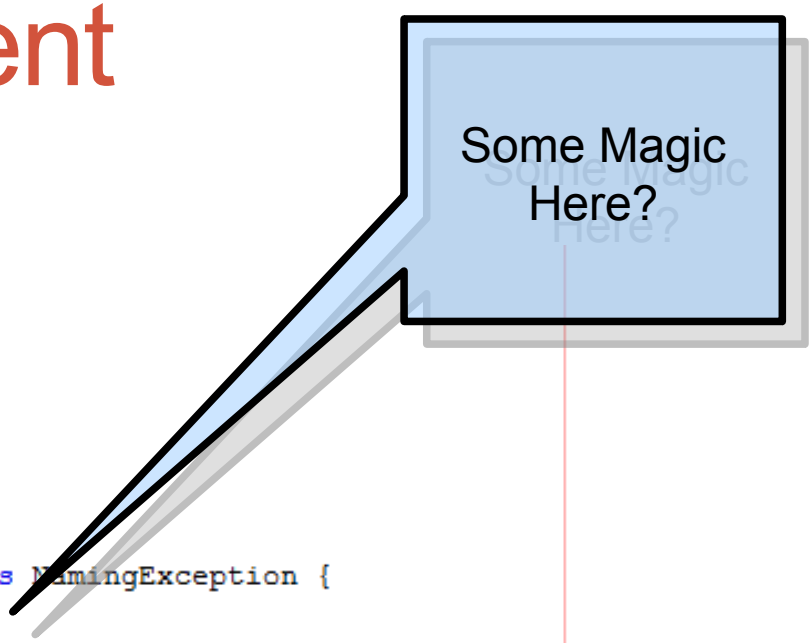
- Successive calls are independant
- No state is kept between two calls from the same client

```java
package esial.test;

import javax.ejb.Stateless;


@Stateless
public class HelloBean implements HelloBeanRemote {
    public String sayHello(String name) {
        System.out.println("Executing sayHello");
        return "coucou "+name;
    }
}
```

```java
package esial.test;

import javax.ejb.Remote;

@Remote
public interface HelloBeanRemote {

    String sayHello(String name);

}
```

Stateless Bean Client

```
package cours.esial;

import esial.test.HelloBeanRemote;
import javax.naming.InitialContext;
import javax.naming.NamingException;

public class Main {
    public static void main(String[] args) throws NamingException {
        InitialContext ic=new InitialContext();
        HelloBeanRemote hello=(HelloBeanRemote)ic.lookup("esial.test.HelloBeanRemote");
        System.out.println(hello.sayHello("guys"));
    }

}
```



JNDI Configuration

```
java.naming.factory.initial=com.sun.enterprise.naming.SerialInitContextFactory
java.naming.factory.url.pkgs=com.sun.enterprise.naming
# Required to add a javax.naming.spi.StateFactory for CosNaming that
# supports dynamic RMI-IIOP.
java.naming.factory.state=com.sun.corba.ee.impl.presentation.rmi.JNDIStateFactoryImpl
```

Allows to create a connection to the naming server

Execution (Glassfish 3)

```
run:
25 août 2010 09:42:14 com.sun.enterprise.transaction.Ja
INFO: Using com.sun.enterprise.transaction.jts.JavaEETr
coucou guys
BUILD SUCCESSFUL (total time: 9 seconds)
```

Client Side

```
INFO: JMXStartupService: Started JMXConnector, JMXService URL = service:jmx:rmi://fatal
INFO: Created HTTP listener http-listener-1 on port 8080
INFO: Grizzly Framework 1.9.18-k started in: 19ms listening on port 8080
INFO: Perform lazy SSL initialization for the listener 'http-listener-2'
INFO: Created HTTP listener http-listener-2 on port 8181
INFO: Grizzly Framework 1.9.18-k started in: 24ms listening on port 8181
INFO: Updating configuration from org.apache.felix.fileinstall-autodeploy-bundles.cfg
INFO: Installed C:\Program Files\sges-v3\glassfish\modules\autostart\org.apache.felix.f
INFO: {felix.fileinstall.poll (ms) = 5000, felix.fileinstall.dir = C:\Users\Charoy\.net
INFO: Executing sayHello
```

Server Side

If something goes wrong

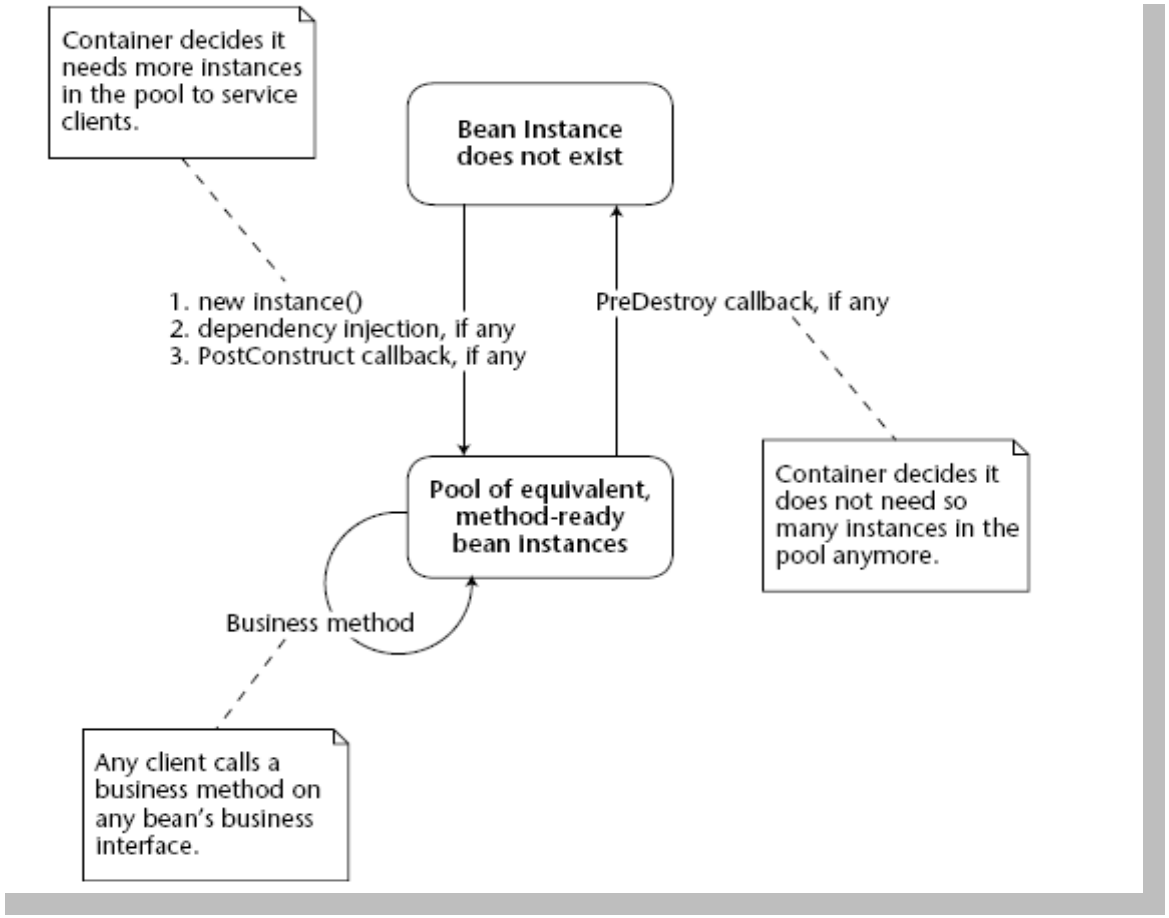
- Never forget the Multi-Tier Context
- It may goes wrong on
  - The Client Side
  - The Server Side

Session Bean Life Cycle

- The container decides
  - When to create a Bean
  - When to delete a Bean
  - When it is available

- Life Cycle Management cannot be controlled

SLSB Lifecycle



Stateful Session Bean

- Non persistant object
- The state is maintain through the session
- May have a Remote or a Local Interface
- @Stateful

Stateful Session Bean

- *memory* is the state

```
package esial.test;

import javax.ejb.Stateful;
@Stateful
public class Memory implements MemoryRemote {
    private int memory;
    public void add(int x) {
        memory=memory+x;
    }

    public void sub(int x) {
        memory=memory-x;
    }

    public int getMemory() {
        return memory;
    }
}
```

Using a SFSB

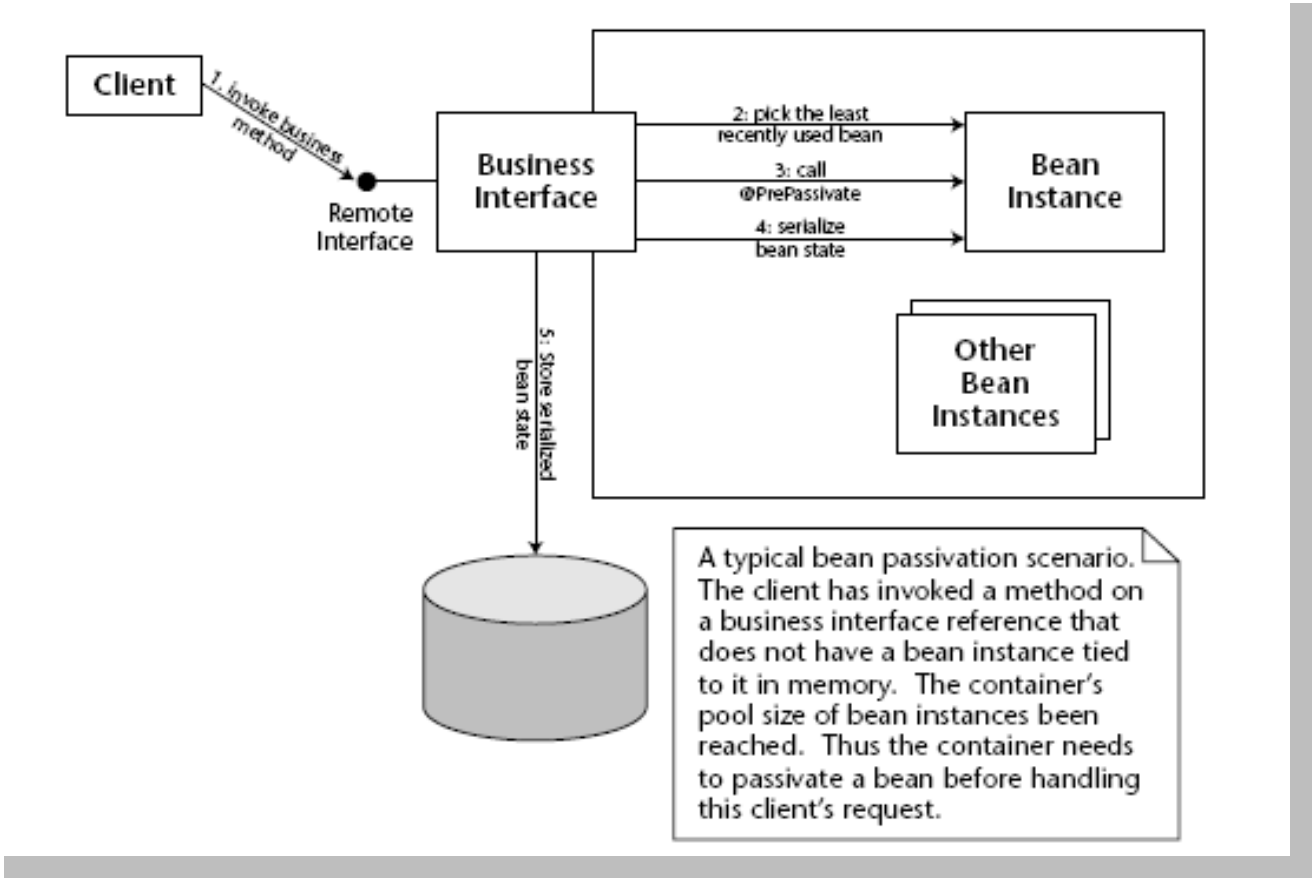
```
package cours.esial;

import esial.test.MemoryRemote;
import javax.naming.InitialContext;
import javax.naming.NamingException;

public class MainSF {
    public static void main(String[] args) throws NamingException {
        InitialContext ic=new InitialContext();
        MemoryRemote memory1=(MemoryRemote)ic.lookup("esial.test.MemoryRemote");
        memory1.add(5);
        memory1.add(3);
        System.out.println(memory1.getMemory());
        MemoryRemote memory2=(MemoryRemote)ic.lookup("esial.test.MemoryRemote");
        memory2.add(6);
        memory2.sub(2);
        System.out.println(memory2.getMemory());
    }
}
```

```
run:
25 août 2010 11:21:03 com.sun.enterprise.transaction.JavaEETrar
INFO: Using com.sun.enterprise.transaction.jts.JavaEETransactic
8
4
BUILD SUCCESSFUL (total time: 4 seconds)
```

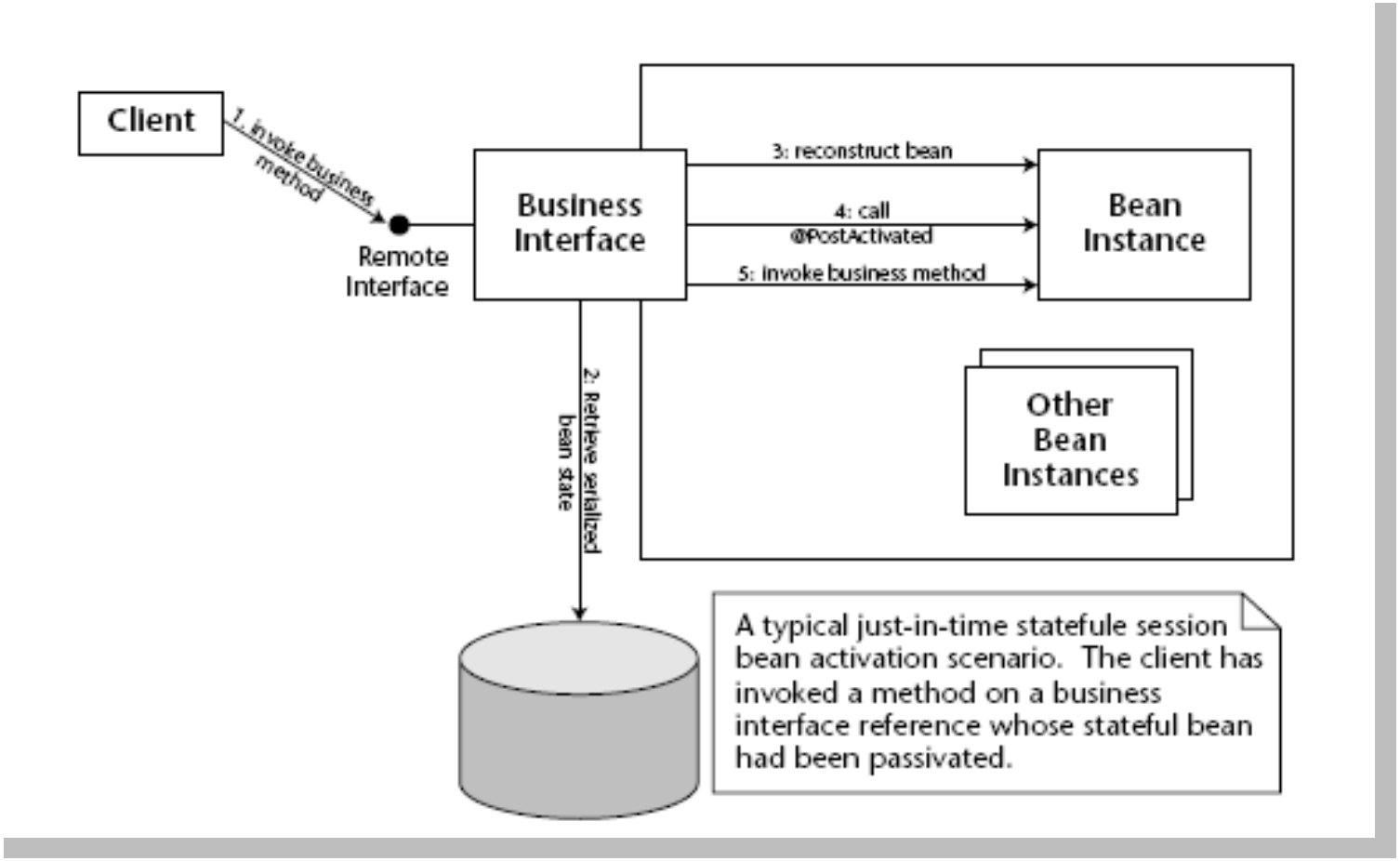
Activation and Passivation

- A SFSB has a memory footprint
- Activation/Passivation = System pagination
- Passivation when resource is missing
- Activation when the bean is needed

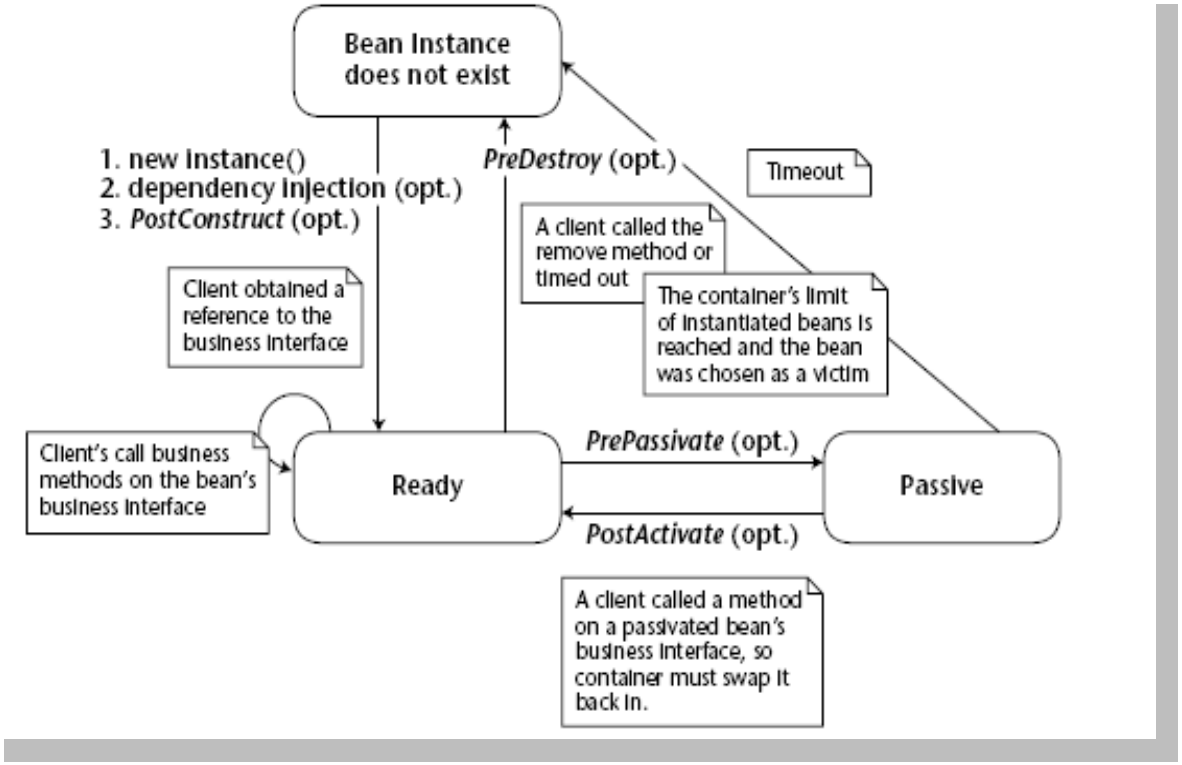
Passivation



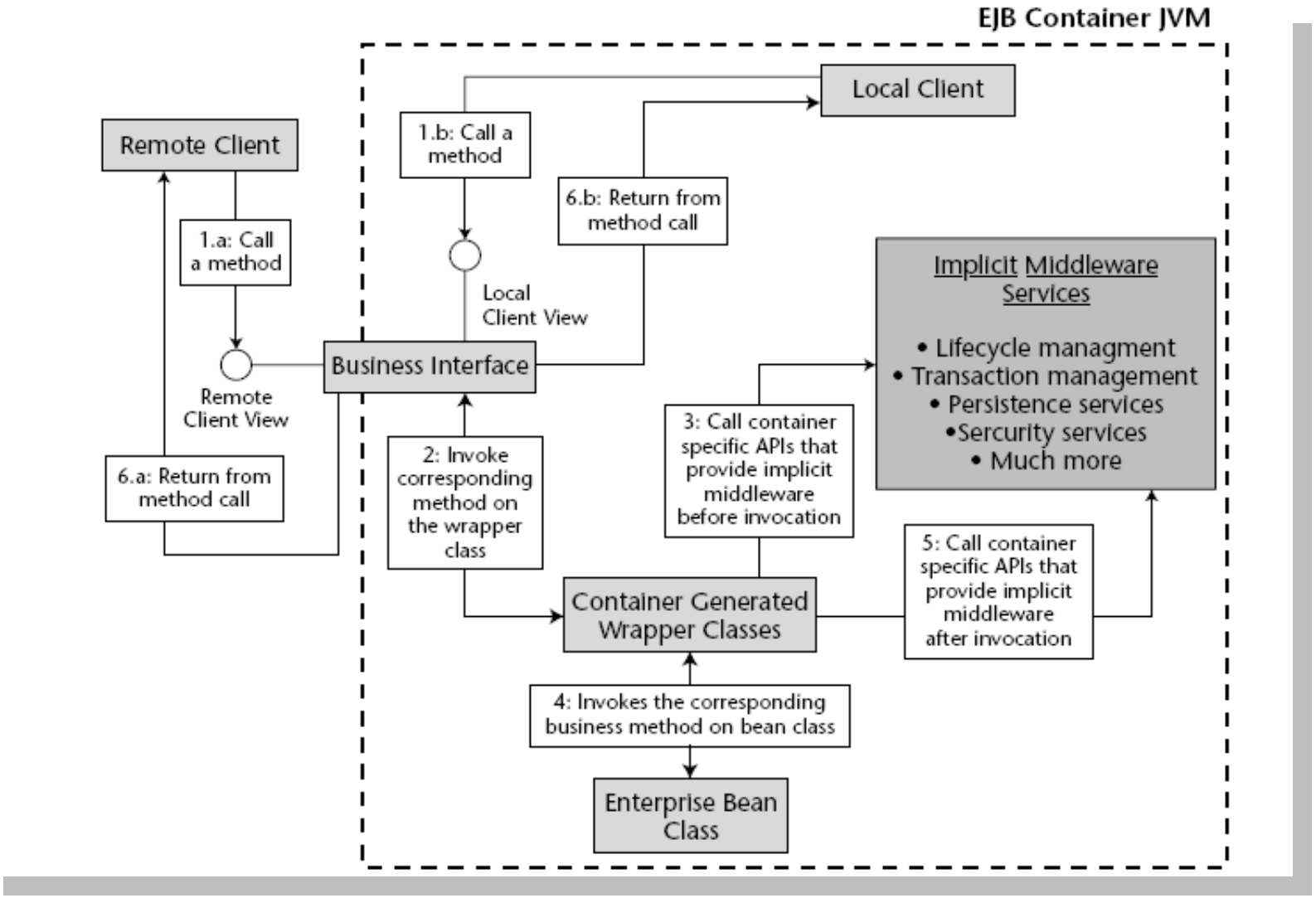
Activation


Client
1. invoke business method
Remote Interface
Business Interface
3: reconstruct bean
4: call @PostActivated
5: invoke business method
Bean Instance
Other Bean Instances
2: Retrieve serialized bean state
A typical just-in-time statefule session bean activation scenario. The client has invoked a method on a business interface reference whose stateful bean had been passivated.

SFSB Lifecycle



Session Bean Structure



Callbacks

- Allow to insert code at some point in the Bean Lifecycle
- @PostConstruct : called after EJB creation
- @PreDestroy : called before EJB descruction
- @PostActivate : called after activation
- @PrePassivate : called before activation

Basic use

```
@Stateful
public class MemoryBean implements MemoryRemote {
    private int memory=0;

    public int add(int a) {
        return memory+a;
    }

    public int minus(int a) {
        return memory-a;
    }

    @PostConstruct
    public void creation() {
        System.out.println("A new Session Bean has been created");
    }
}
```

Execution

```
MemoryRemote mr1=(MemoryRemote)ic.lookup("tp.MemoryRemote");
System.out.println("mr1="+mr1.add(10));
MemoryRemote mr2=(MemoryRemote)ic.lookup("tp.MemoryRemote");
System.out.println("mr2="+mr2.add(5));
```

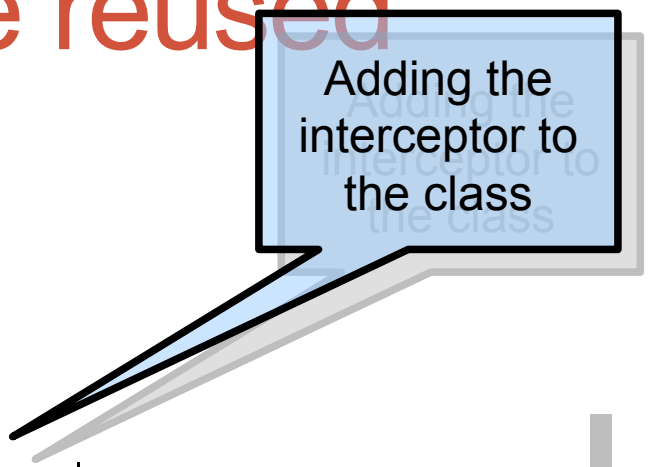
```
run:
10 + 5=15
mr1=10
mr2=5
```

```
**RemoteBusinessJndiName: tp.MemoryRemote; remoteBusIntf: tp.MemoryRemote
LDR5010 : Tous les ejb de [CalcEJB] chargés avec succès !
A new Session Bean has been created
A new Session Bean has been created
```

Callbacks and interceptors

```
public class CompteurCallback {
    @PostConstruct
    public void construct(InvocationContext ctx) {
        System.out.println("dans construct");
        System.out.println("Target name = "+ctx.getTarget());
    }
    @PreDestroy
    public void destroy(InvocationContext ctx) {
        System.out.println("dans destroy");
        System.out.println("Target name = "+ctx.getTarget());
    }
    @PrePassivate
    public void passivate(InvocationContext ctx) {
        System.out.println("dans passivate");
        System.out.println("Target name = "+ctx.getTarget());
    }
    @PostActivate
    public void activate(InvocationContext ctx) {
        System.out.println("dans activate");
        System.out.println("Target name = "+ctx.getTarget());
    }
}
```

Interceptors can be reused




```
@Stateful
@Interceptors(CompteurCallback.class)
public class CompteurBean implements CompteurRemote {
    int compteur;
```

```
LDR5010: All ejb(s) of [Exemple] loaded successfully!
dans construct
Target name = cours._CompteurBean_Serializable@1291453
Dans count
Suppression explicite du Bean
dans destroy
Target name = cours._CompteurBean_Serializable@1291453
```

Singleton (new in EJB 3.1)

- Ensure that there is only one instance of the Bean
- Concurrency can be managed by the Bean or by the Container

```
package esial.test;

import javax.ejb.Singleton;
@Singleton
public class Single implements SingleRemote {

    private int callCount;

    public void increment() {
        callCount++;
    }

    public int getCallCount() {
        return callCount;
    }
}
```

Singleton - exécution

```
package cours.esial;

import esial.test.SingleRemote;
import javax.naming.InitialContext;
import javax.naming.NamingException;
public class MainSingleton {
    public static void main(String[] args) throws NamingException {
        InitialContext ic=new InitialContext();
        SingleRemote sr1=(SingleRemote)ic.lookup("esial.test.SingleRemote");
        sr1.increment();
        System.out.println(sr1.getCallCount());
        SingleRemote sr2=(SingleRemote)ic.lookup("esial.test.SingleRemote");
        sr2.increment();
        System.out.println(sr2.getCallCount());
    }

}
```

```
run:
25 août 2010 14:48:00 com.sun.enterprise.transaction.JavaEE
INFO: Using com.sun.enterprise.transaction.jts.JavaEETransa
1
2
BUILD SUCCESSFUL (total time: 5 seconds)
```

```
run:
25 août 2010 14:48:39 com.sun.enterprise.transaction.JavaEE
INFO: Using com.sun.enterprise.transaction.jts.JavaEETransa
3
4
BUILD SUCCESSFUL (total time: 4 seconds)
```

Startup Callback

- Eager initialisation

```
@Singleton
@Startup
public class StartupBean {

  @PostConstruct
  private void onStartup() { … }

  @PreDestroy
  private void onShutdown() { … }

}
```

EJB injection

- EJB Convert – Interface locale
- EJB Bank
  - Interface Remote
  - A reference to Convert is injected in Bank

```java
package esial.test;

import javax.ejb.EJB;
import javax.ejb.Stateless;
@Stateless
public class Bank implements BankRemote, BankLocal {
    @EJB
    private ConvertLocal convert;

    public double convertAndTax(double amount) {
        System.out.println("In Bank.convertAndTax");
        return (convert.convert(amount)*1.05) +1;
    }
}
```

```java
package esial.test;

import javax.ejb.Stateless;

@Stateless
public class Convert implements ConvertRemote, ConvertLocal {
    public double convert(double amount) {
        System.out.println("In Convert.convert");
        return amount*0.78;
    }
}
```

Execution

```
package cours.esial;

import esial.test.BankRemote;
import javax.naming.InitialContext;
import javax.naming.NamingException;

public class MainBank {
    public static void main(String[] args) throws NamingException {
        InitialContext ic=new InitialContext();
        BankRemote br=(BankRemote)ic.lookup("esial.test.BankRemote");
        System.out.println("100 $ = "+br.convertAndTax(100)+" €");
     }
}
```

```
25 août 2010 15:32:26 com.sun.enterprise.transaction.JavaEE
INFO: Using com.sun.enterprise.transaction.jts.JavaEETransa
100 $ = 82.9 €
BUILD SUCCESSFUL (total time: 4 seconds)
```

```
INFO: TestJEE was successfully deployed in 1 084 milliseconds.
INFO: In Bank.convertAndTax
INFO: In Convert.convert
```

Configuration

- Configuration can be done with a configuration file
- XML representation
- External configuration takes precedence on Annotation
- See Chapter 19 of the specification

Session Bean without interface

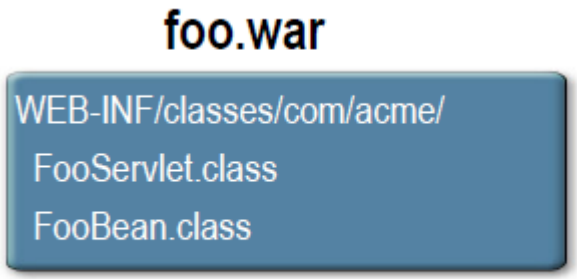
- No implements
- All public methods are available

```
@Stateless
public class HelloBean {

    public String sayHello(String msg) {
        return "Hello " + msg;
    }

}
```

```
@EJB HelloBean h;

...

h.sayHello("bob");
```

Packaging des applications

- Before/after (Java ee5 / Java ee6)



Lite Version

- Allows to use EJB in web application

Lite

- Local Session Beans
- CMT / BMT
- Declarative Security
- Interceptors

** Web Profile also includes Java Persistence API

Full = Lite + :

- Message-Driven Beans
- Web Service Endpoints
- 2.x / 3.x Remote view
- RMI-IIOP Interoperability
- Timer Service
- Async method calls
- 2.x Local view
- CMP / BMP Entity Beans

Portable naming

- Globally unique name

```
java:global[/<app-name>]/<module-name>/<ejb-name>
```

- Unique name within same application

```
java:app/<module-name>/<ejb-name>
```

- Unique name within defining module

```
java:module/<ejb-name>
```

Example

```
@Stateless
public class HelloBean implements Hello {
    public String sayHello(String msg) {
      return "Hello " + msg;
   }
}
```

If deployed as hello.jar, JNDI entries are:

java:global/hello/HelloBean
java:app/hello/HelloBean
java:module/HelloBean

Testing EJB

- Hard to test
  - Deployment is required
  - Client/Server testing
  - No standard

```
@Stateless
@Local(Bank.class)
public class BankBean implements Bank {

  @PersistenceContext EntityManager accountDB;

  public String createAccount(AcctDetails d)
   { … }

  public void removeAccount(String acctID)
   { … }
```

Embedded EJB Container

No need for a server

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

    EJBContainer container =
       EJBContainer.createEJBContainer();

    // Acquire Local EJB reference
    Bank bank = (Bank) container.getContext().
      lookup("java:global/bank/BankBean");

    testAccountCreation(bank);
    container.close();
```

```
java -classpath bankClient.jar :
                bank.jar :
                javaee.jar :
                <vendor_rt>.jar

                com.acme.BankTester
```

On Session Bean

- Distributed Component (or not)
  - Executed in an EJB container
- Provide Business Services
  - Stateful or not
- Can be configured
  - Security
  - Transaction
  - Resources

More on EJB

- Calling EJB
- Annotation
- Interception
- Injection

Annotations

- @Stateless, @Stateful
  - + variations
- Meta-Programming
- Information about the code behavior
- Can be overidden in configuration files

Injection

- Reference injection
  - Initialisation at instatiation time

```
@Stateless
public class FacadeBean implements FacadeRemote {

    @EJB
    private RandomLocal randomBean;

    public FacadeBean() {
    }

    public long getDice() {
        return Math.round(randomBean.random()*6.0)+1;
    }
```

Un RandomBean injected in the FacadeBean

No NullPointerException

```
@Stateless
public class RandomBean implements RandomLocal {
    private Random r;
    /** Creates a new instance of RandomBean */
    public RandomBean() {
        r=new Random();
    }

    public double random() {
        return r.nextDouble();
    }
```

Ressource réference

```
@Resource(name="jdbc/__default",type=Datasource.class)
DataSource dataSource;
```

```
package com.temp;

public class MyClass {
   ...
   @Resource
   private void setMyDataSource(DataSource ds) {
     myDataSource = ds;
   }
   private myDataSource;
   ...
}
```

Setter Injection

```
@Resources({
   @Resource(name="datasource",type="javax.sql.DataSource.class),
   @Resource(name="queue",type="javax.jms.Queue")
})
public class X {
   ...
}
```

Injection du PersistenceContext

```
@PersistenceContext EntityManager em;
```

```
@PersistenceContext(unitName="pu1") EntityManager em;
```

Interceptors

- Invoked in replacement or around business methods
- Basic Aspect Oriented Programming
- Facilitate Separation of concerns

Example 1

```
public class PerformanceInterceptor {
    @AroundInvoke
    public Object timer(InvocationContext ctx) throws Exception {
        long startTime=System.nanoTime();
        Object o=ctx.proceed();
        long endTime=System.nanoTime();
        long totalTime=endTime-startTime;
        System.out.println("Time spent in "+ctx.getMethod().getName()+" = "
                    +(totalTime/1000)+" microseconds");
        return o;
    }
}
```

Example 2

```
public class LoggingInterceptor {
    @AroundInvoke
    public Object logger(InvocationContext ctx) throws Exception {
        System.out.println("Intercepted call via "+
                "external class to: "+ctx.getMethod().getName());
        return ctx.proceed();
    }
}
```

Utilisation des intercepteurs

```
@Stateless
@Interceptors({LoggingInterceptor.class,PerformanceInterceptor.class})
public class RandomBean implements RandomLocal {
    private Random r;
    /** Creates a new instance of RandomBean */
    public RandomBean() {
        r=new Random();
    }

    public double random() {
        return r.nextDouble();
    }
```

```
Intercepted call via external class to: getDice
Intercepted call via external class to: random
Time spent in random = 18 microseconds
Time spent in getDice = 872 microseconds
Intercepted call via external class to: getDice
Intercepted call via external class to: random
Time spent in random = 15 microseconds
Time spent in getDice = 349 microseconds
Intercepted call via external class to: getDice
Intercepted call via external class to: random
Time spent in random = 13 microseconds
Time spent in getDice = 282 microseconds
```

Conclusion

- Simplify access to resources
- Injecting is possible for all named resources
- Interceptors simplify separation of concerns
- Injection and Interception available in most modern framework

Where We are

- At the end of the part on distributed components
- What is missing now ?
  - Advanced concepts
  - Relationships to the data management tier