Taller de Sistemas de Información 2

Componentes de negocio

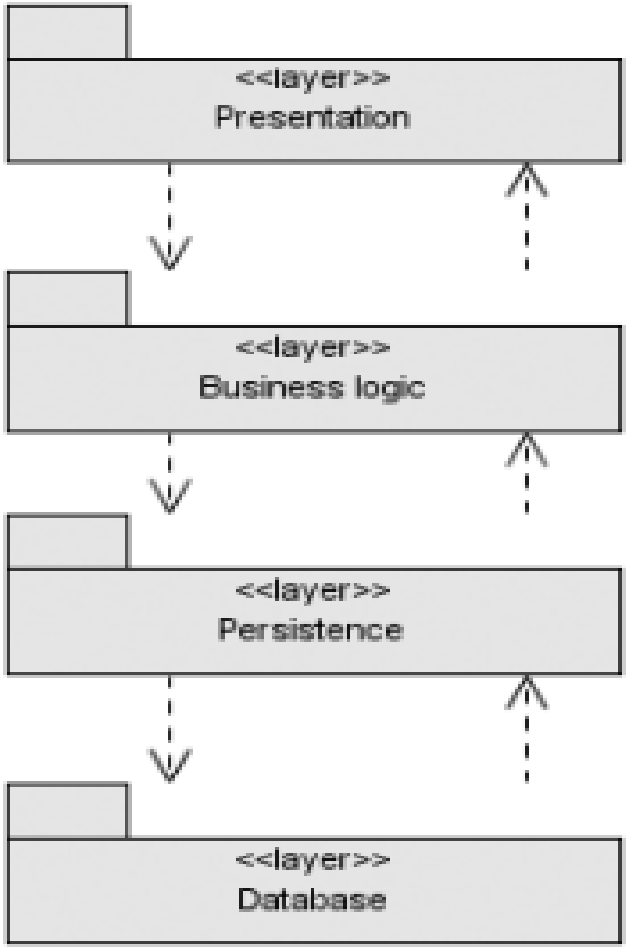
26 de Agosto de 2014

Instituto de Computación
Facultad de Ingeniería
Universidad de la República de Uruguay

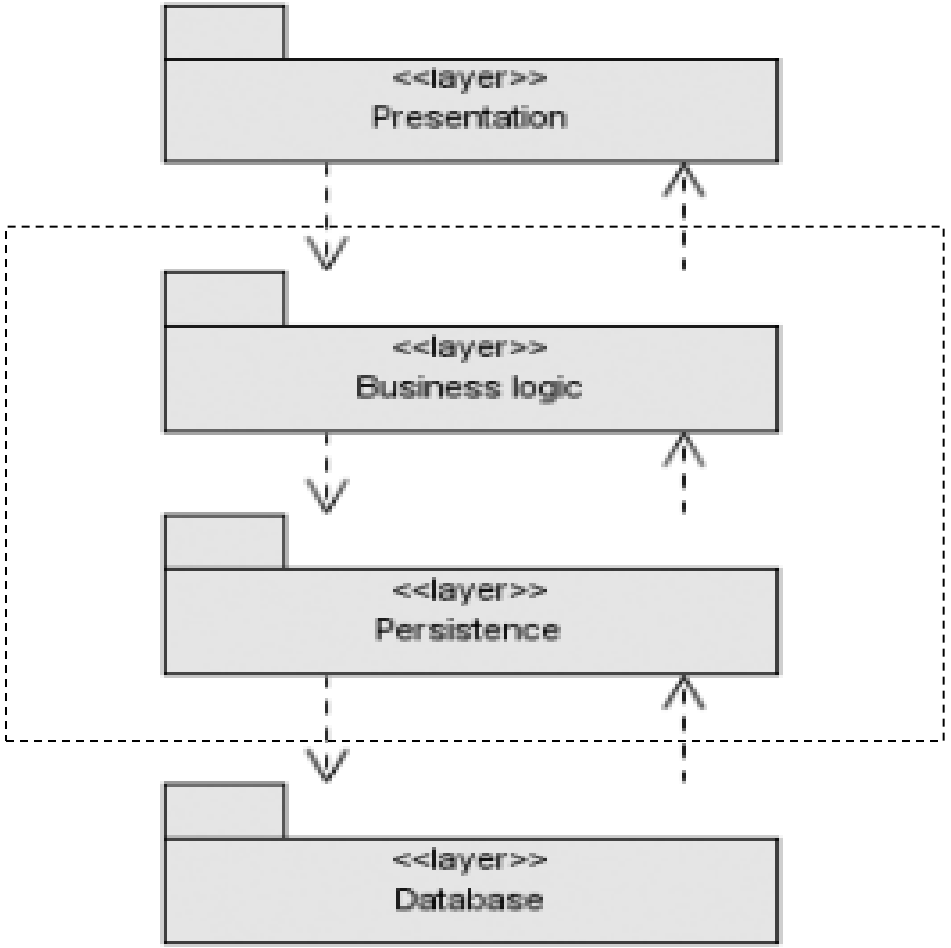
Introducción

- Los EJBs son componentes server-side que encapsulan lógica de negocio, y se encargan del manejo transaccional y de seguridad
- También poseen un stack integrado para el manejo de mensajería, scheduling, acceso remoto, web services (REST / SOAP), inyección de dependencias, ciclo de vida y AOP

Introducción



Introducción



Ámbito de los componentes de negocio

Tipos de EJB

- La especificación define tipos de EJB, cada uno de ellos adaptado a resolver una problemática especifica
- Esto tipos son:
  - Session Beans
    - Stateful
    - Stateless
    - Singleton
  - Message Driven Beans

Tipos de EJB

- Un session bean tiene diferentes variantes, según como maneja el estado el mismo
- Stateless
  - No posee estado conversacional entre métodos
  - Cualquier instancia puede ser usada por cualquier cliente
- Stateful
  - Posee estado conversacional, el cual debe ser retenido entre métodos para el mismo usuario

Tipos de EJB

- Los Message Driven Beans son usados para resolver integración con sistemas externos, recibiendo mensajes a través de JMS
  - Java Message Service API
- En general se utilizan cuando se quiere modelar asincronismo
- Usualmente delegan su lógica de negocio a los session beans

Anatomía de un EJB

- Para crear un EJB, solo se necesita una clase y una anotación (en su forma mas reducida)

```
@Stateless
public class BookEJB {
        @PersistenceContext(unitName = "LIBRARY")
        private EntityManager em;
        public Book findBookById(Long id) {
                return em.find(Book.class, id);
        }
        public Book createBook(Book book) {
                em.persist(book);
                return book;
        }
}
```

Ejecución de un EJB

- Para que los componentes funcionen, deben ejecutar en un ambiente llamado container
- Este provee los siguientes servicios
  - Comunicación remota
  - Inyección de dependencias
  - Manejo de estado
  - Pooling
  - Ciclo de vida del componente

Ejecución de un EJB

- Este provee los siguientes servicios
  - Mensajería
  - Manejo de transacciones
  - Seguridad
  - Soporte de concurrencia
  - Interceptores
  - Invocación asincronica de métodos (sin usar mensajería)

Packaging

- Como los demas componentes de la plataforma, los EJBs deben ser empaquetados para poder ser colocados en el container
- Una vez que se arma un JAR con el contenido de los artefactos que implementan el EJB, asi como las clases necesarias, entonces podemos colocarlo en el container

Session Beans

- Los session beans son excelentes para implementar lógica de negocio, workflows, procesos, etc.
- Debemos elegir el tipo de session bean según lo que queramos implementar
- Tenemos tres tipos
  - Stateless, Stateful y Singleton

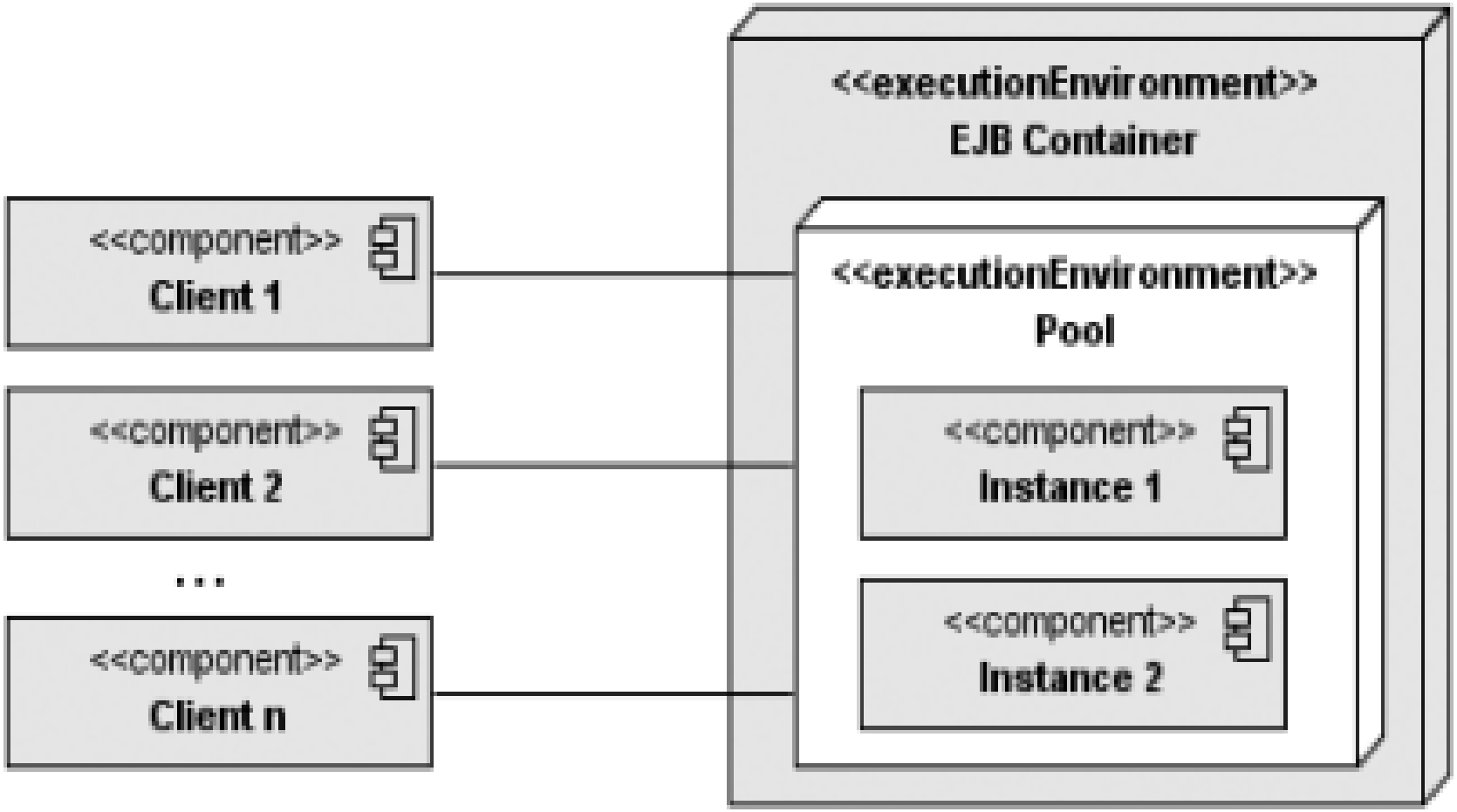
Session Beans

- Cualquiera de los tres tipos de session bean comparten el mismo modelo de programación
- Pueden tener interfaces locales, remotas, web service o ninguna
- Deben ser empaquetados en un ensamblado apropiado (jar, war o ear)
- Es responsabilidad del container, manejar el ciclo de vida del componente

Stateless Beans

- Son el tipo de componente mas popular
- Son simples, eficientes y potentes
- Cumplen con la tarea común de implementar lógica de negocio sin estado
- Pueden ser reutilizados por múltiples clientes, con lo que aumenta la escalabilidad de la aplicación
- El container mantiene un pool con las instancias del componente

Stateless Beans



Stateless Beans

- Para cada EJB, el container mantiene una cierta cantidad de instancias en memoria
- Estas instancias son compartidas por los clientes
- Como los stateless beans no mantienen estado, las instancias son equivalentes entre si
- La instancia es quitada y devuelta al pool, cuando comienza y termina la atención del cliente, respectivamente

ItemEJB

```
@Stateless
public class ItemEJB {
        @PersistenceContext(unitName = "LIBRARY")
        private EntityManager em;
        public List<Book> findBooks() {
                Query query = em.createNamedQuery("findAllBooks");
                return query.getResultList();
        }
        public List<CD> findCDs() {
                Query query = em.createNamedQuery("findAllCDs");
                return query.getResultList();
        }
        public Book createBook(Book book) {
                em.persist(book);
                return book;
        }
        public CD createCD(CD cd) {
                em.persist(cd);
                return cd;
        }
}
```

Stateless Beans

- En general los session beans contienen métodos de negocio fuertemente relacionados entre si
- Por ejemplo, el ItemEJB contiene métodos que están relacionados con la venta de Cds
- @Stateless es la anotación que caracteriza un POJO como un componente EJB stateless

@Stateless

```
@Target({TYPE}) @Retention(RUNTIME)
public @interface Stateless {
     String name() default "";
     String mappedName() default "";
     String description() default "";
}
```

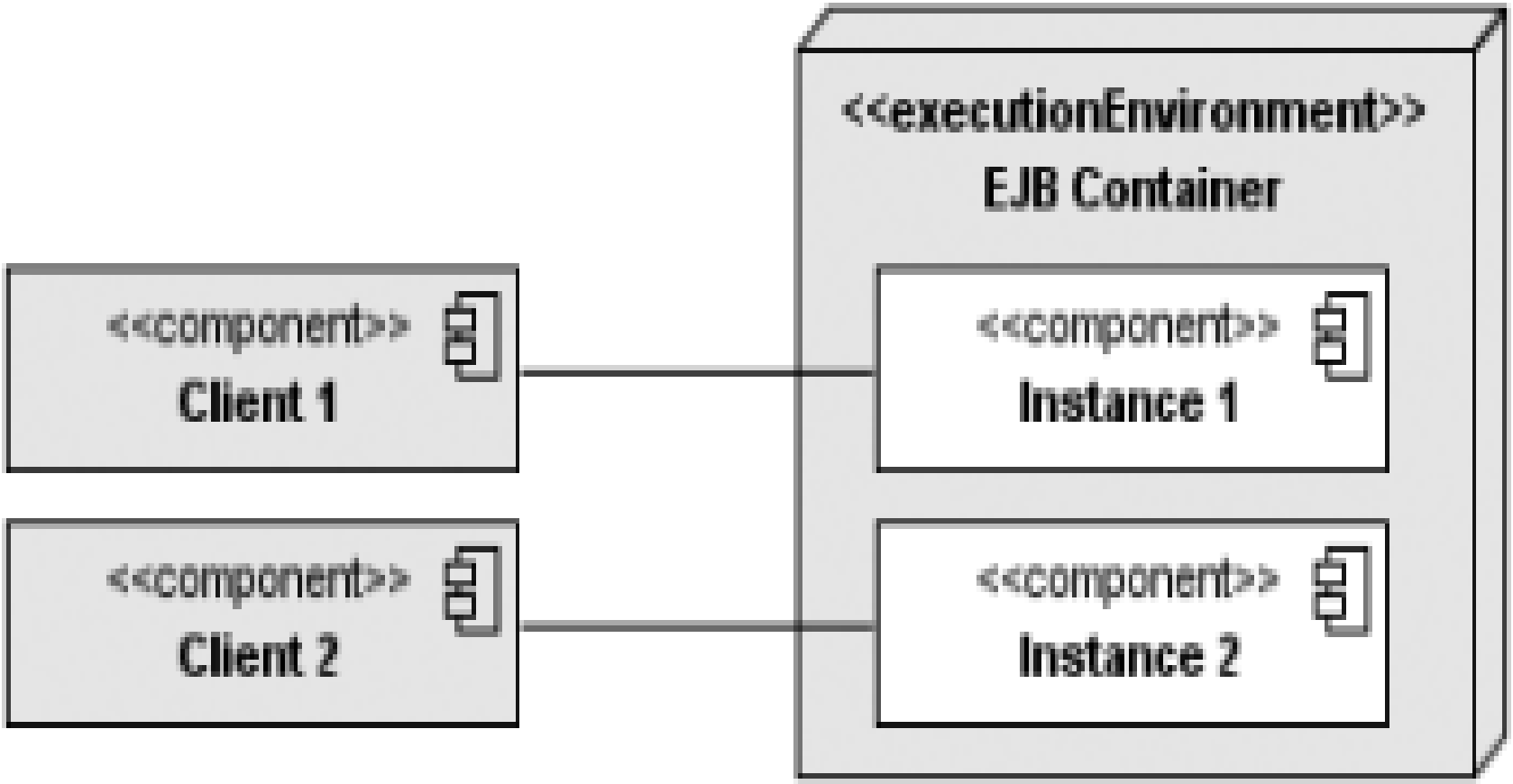
Stateful Beans

- A diferencia de los Stateless, este tipo de beans mantienen estado conversacional con su cliente
- Son muy útiles para procesos que deben realizarse en varios pasos, donde cada paso depende del estado dejado por el anterior
- Por ejemplo...

Stateful Beans

```
   Book book = new Book();
   book.setTitle("The Hitchhiker's Guide to the Galaxy");
   book.setPrice(12.5F);
   book.setDescription("Science fiction comedy series created by Douglas Adams.");
   book.setIsbn("1-84023-742-2");
   book.setNbOfPage(354);
   book.setIllustrations(false);
>> statefullComponent.addBookToShoppingCart(book);
   book.setTitle("The Robots of Dawn");
   book.setPrice(18.25F);
   book.setDescription("Isaac Asimov's Robot Series");
   book.setIsbn("0-553-29949-2");
   book.setNbOfPage(276);
   book.setIllustrations(false);
>> statefullComponent.addBookToShoppingCart(book);
>> statefullComponent.checkOutShoppingCart();
```

Stateful Beans



```
@Stateful
public class ShoppingCartEJB {
        private List<Item> cartItems = new ArrayList<Item>();
        public void addItem(Item item) {
                if (!cartItems.contains(item))
                        cartItems.add(item);
        }
        public void removeItem(Item item) {
                if (cartItems.contains(item))
                        cartItems.remove(item);
        }
        public Float getTotal() {
                Float total = 0f;
                for (Item cartItem : cartItems) {
                        total += (cartItem.getPrice());
                }
                return total;
        }
        @Remove
        public void checkout() {
                cartItems.clear();
        }
        @Remove
        public void empty() {
                cartItems.clear();
        }
}
```

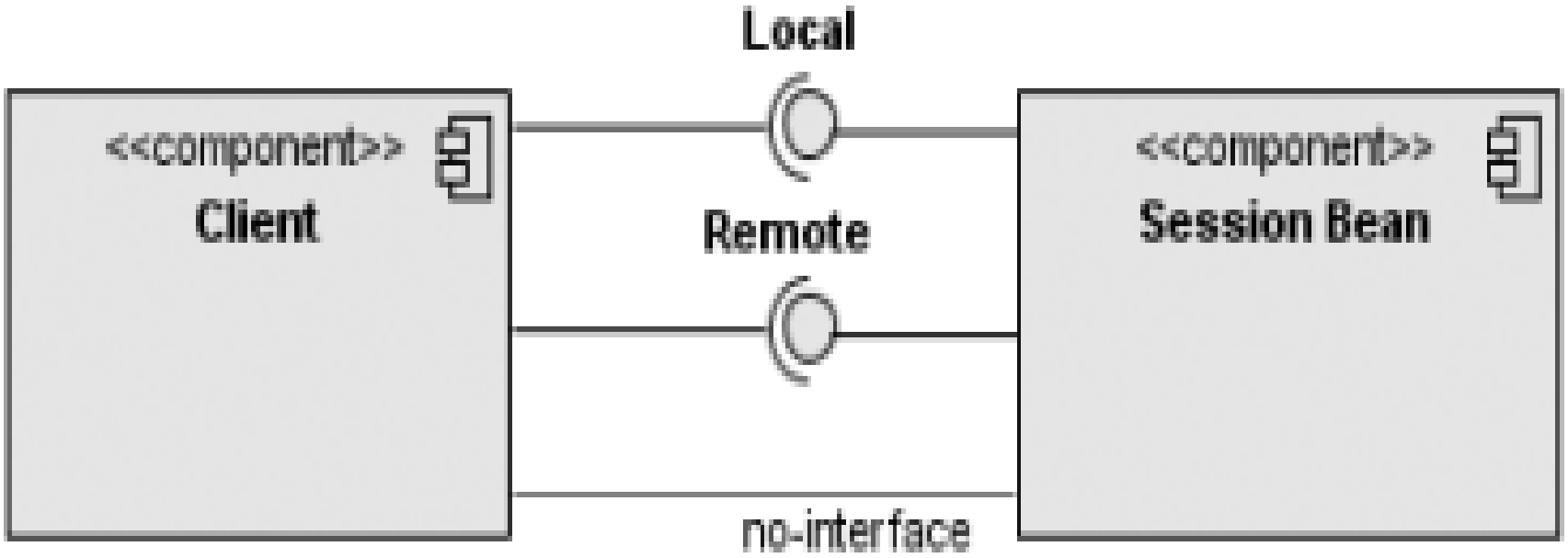
Eliminación de instancias

- @Remove puede ser aplicada sobre métodos del EJB
- La invocación de un método marcado con @Remove, hará que la instancia del bean sea eliminada del servidor de aplicaciones
- @StatefulTimeout(milis), aplicada sobre la clase puede ser usada para indicar la cantidad de milisegundos máxima que el bean puede estar sin actividad, antes de ser destruido

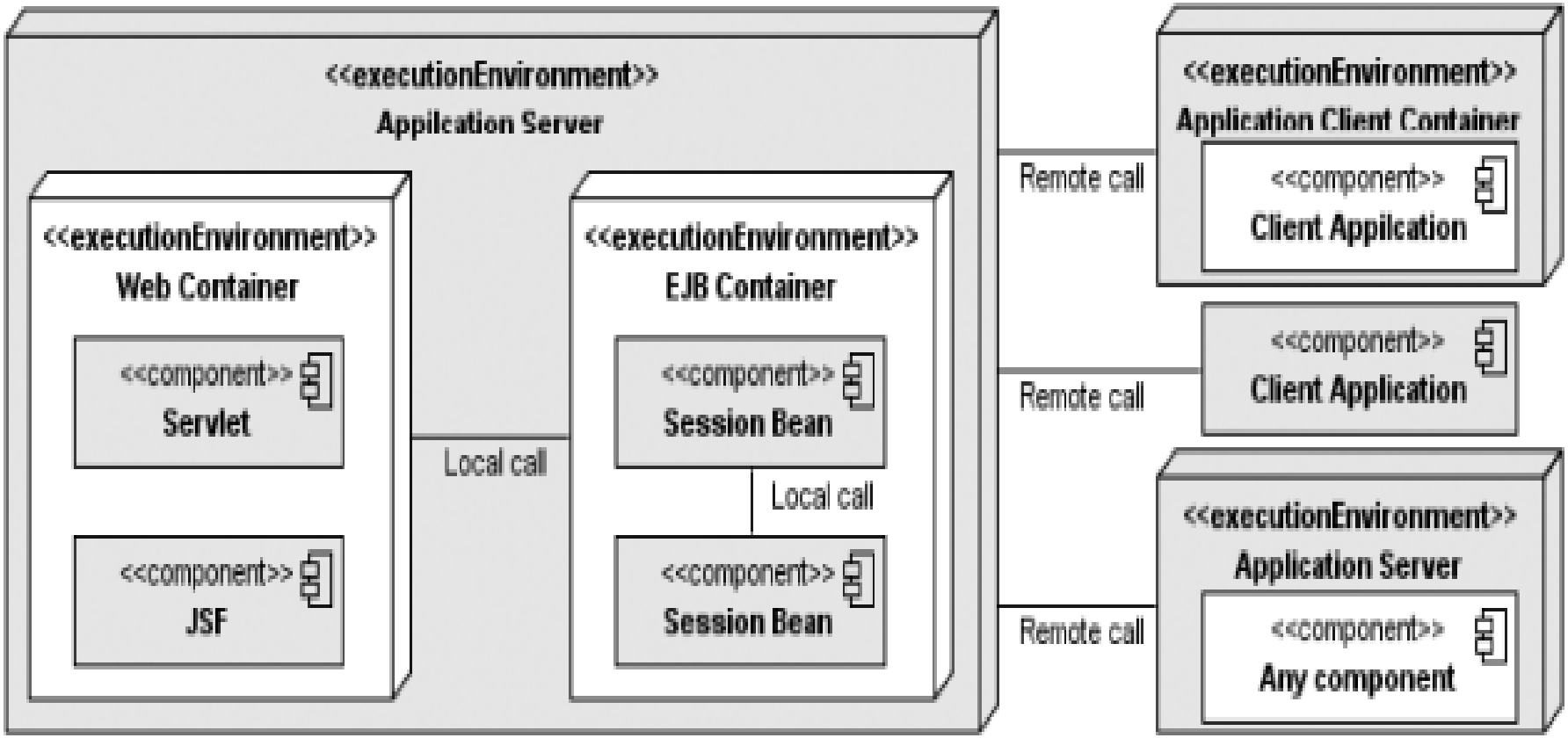
Eliminación de instancias

- Alternativamente, podemos no utilizar estas anotaciones, y confiar en que el bean sea reciclado por el container, cuando la sesión del cliente termine o expire
- Sin embargo, remover explícitamente el bean disminuye el consumo de memoria de la aplicación

Tipos de interfaces



Tipos de interfaces



Tipos de interfaces

- Un session bean puede implementar multiples interfaces o ninguna interfaz
- Una interfaz de un Session Bean, es una interfaz Java tradicional
- Pueden utilizar las anotaciones
  - @Remote para indicar que es una interfaz accedida remotamente
  - @Local para indicar que es una interfaz accedida localmente

Tipos de interfaces

```
@Local
public interface ItemLocal {
        List<Book> findBooks();
        List<CD> findCDs();
}

@Remote
public interface ItemRemote {
        List<Book> findBooks();
        List<CD> findCDs();
        Book createBook(Book book);
        CD createCD(CD cd);
}

@Stateless
public class ItemEJB implements ItemLocal, ItemRemote {
...
}
```

Tipos de interfaces

```
public interface ItemLocal {
        List<Book> findBooks();
        List<CD> findCDs();
}

public interface ItemRemote {
        List<Book> findBooks();
        List<CD> findCDs();
        Book createBook(Book book);
        CD createCD(CD cd);
}

@Stateless
@Remote (ItemRemote)
@Local (ItemLocal)
public class ItemEJB implements ItemLocal, ItemRemote {
...
}
```

Clase de implementación

- También se denomina “bean class”
- Es una clase Java que se adhiere a ciertas restricciones, además de ser un POJO
- La clase debe estar anotada con @Stateles, @Stateful o @Singleton
- Debe implementar todos los métodos de sus interfaces (si existen)
- La clase debe ser publica, y no puede ser final ni abstract

Clase de implementación

- Debe tener un constructor sin argumentos, usado por el container para crear instancias de ese bean
- No debe definir el método finalize()
- Los métodos de negocio no pueden comenzar con el prefijo “ejb”, y no pueden ser final o static
- En un método remoto, los valores intercambiados deben ser valores validos para RMI

Clientes

- Los clientes de un Session Bean, pueden ser cualquier tipo de artefacto Java capaz de realizar la invocación a un método o a un web service
- Para invocar un método, nunca se instancia el componente a partir de new
- Se utiliza una referencia al bean, la cual puede obtenerse a través de la anotación @EJB o a través de búsquedas JNDI

Clientes: @EJB

```
@Stateless
@Remote (ItemRemote)
@Local (ItemLocal)
public class ItemEJB implements ItemLocal, ItemRemote {
        ...
}

// Codigo cliente
@EJB ItemEJB itemEJB; // No es posible acceder a la clase
@EJB ItemLocal itemEJBLocal;
@EJB ItemRemote itemEJBRemote;
```

Clientes: @EJB

- Dependiendo del tipo de cliente de un Session Bean, esta anotación puede no funcionar
- Este es el caso cuando el cliente no es un componente administrado en el que se pueda inyectar la referencia al bean
- Para estos casos, debemos utilizar busquedas a través de JNDI
  - Java Naming and Directory Interface

Clientes: JNDI

- Una vez que un EJB es instalado en el servidor de aplicaciones, es automaticamente asociado a un nombre JNDI
- Antes de Java EE 6, este nombre no estaba estandarizado, por lo que habia diferencias entre los servidores de aplicación
- El formato estándar es:
  - java:global[/<app-name>]/<module-name>/<bean-name>[!<fully-qualified-interface-name>]

Clientes: JNDI

- <app-name>
  - Es opcional, solo aplica cuando el componente se encuentra dentro de un EAR
  - Por defecto su valor es igual al nombre del archivo del EAR, sin la extensión .ear
- <module-name>
  - Es el nombre del modulo que contiene el EJB
  - Puede ser un archivo jar o un war
  - Por defecto, su valor es igual al nombre del archivo sin la extensión

Clientes: JNDI

- <bean-name>
  - Es el nombre del componente
  - Por defecto es igual a la clase de implementación
- <fully-qualified-interface-name>
  - Es el nombre completo (package + clase) de la interfaz que se utilizara para acceder al bean
  - Para el caso de no usar interfaces, este valor es igual al nombre completo (package + clase) de la clase del bean

Clientes: JNDI

```
package com.javaee6;

@Stateless
@LocalBean
@Remote (ItemRemote)
public class ItemEJB implements ItemRemote {
        ...
}

// Nombres JNDI
java:global/cdbookstore/ItemEJB!com.javaee6.ItemEJB
java:global/cdbookstore/ItemEJB!com.javaee6.ItemRemote
```

Session Context

- Si bien los session beans viven dentro del container, en general no necesitan acceder directamente al container o a los servicios brindados por el mismo
- Estos servicios estan pensados para que se resuelvan en forma transparente
- Cuando es necesario acceder directamente al container, usamos la interfaz javax.ejb.SessionContext

Session Context

- getCallerPrincipal
  - Retorna el java.security.Principal asociado a la invocación
- getRollbackOnly
  - Testea si la transaccion actual ha sido marcada para rollback
- getTimerService
  - Retorna la interfaz java.ejb.TimerService, un servicio de scheduling dentro del container

Session Context

- getUserTransaction
  - Retorna la interfaz UserTransaction, usada para demarcar explicitamente transacciones
- isCallerInRole
  - Testea si el llamador se encuentra en un rol determinado
- lookup
  - Permite que el Session Bean haga busquedas JNDI en su contexto

Session Context

- setRollbackOnly
  - Permite que el bean marque la transacción actual para ser rollbackeada
  - Solo puede ser utilizado en beans que usen demarcación explicita
- wasCancelCancelled
  - Determina si el método cancel() fue invocado para el bean actual
  - Solo se usa en invocaciones asíncronas

Session Context

- Inyectamos una referencia al contexto, usando la anotación @Resource

```
@Stateless
public class ItemEJB {
        @Resource
        private SessionContext context;
        ...
        public Book createBook(Book book) {
                ...
                if (cantFindAuthor())
                        context.setRollbackOnly();
        }
}
```

Inyección de dependencias

- El container puede inyectar diferentes tipos de recursos en un Session Bean, a traves de diferentes anotaciones
  - @EJB inyecta un referencia a otro EJB (local, remoto o sin interfaz)
  - @PersistenceContext y @PersistenceUnit inyectan referencias a un EntityManager o un EntityManagerFactory respectivamente
  - @WebServiceRef inyecta una referencia a un web service

Inyección de dependencias

- @Resource inyecta una referencia a diferentes tipos de recursos
- Estos pueden ser
  - Conexiones JDBC (Datasources JDBC)
  - El Session Context
  - La UserTransaction
  - JMS Connection factories
  - Timer service
  - JMS Destinations

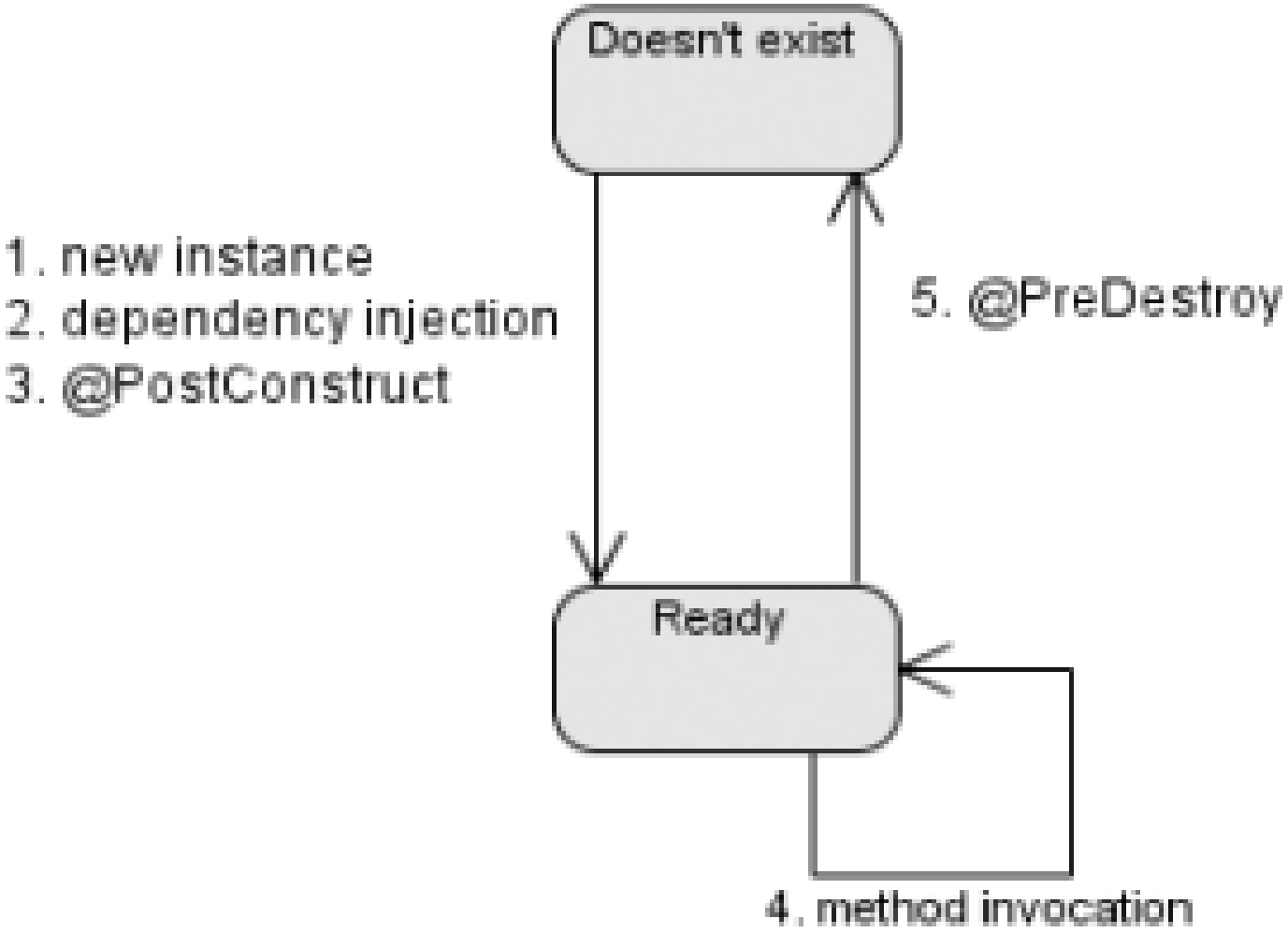
Inyección de dependencias

```
@Stateless
public class ItemEJB {

        @PersistenceContext(unitName = "LIBRARYDS")
        private EntityManager em;

        @EJB
        private CustomerEJB customerEJB;

        @WebServiceRef
        private ArtistWebService artistWebService;

        private SessionContext context;
        @Resource
        public void setCtx(SessionContext ctx) {
                this.ctx = ctx;
        }
        ...
}
```

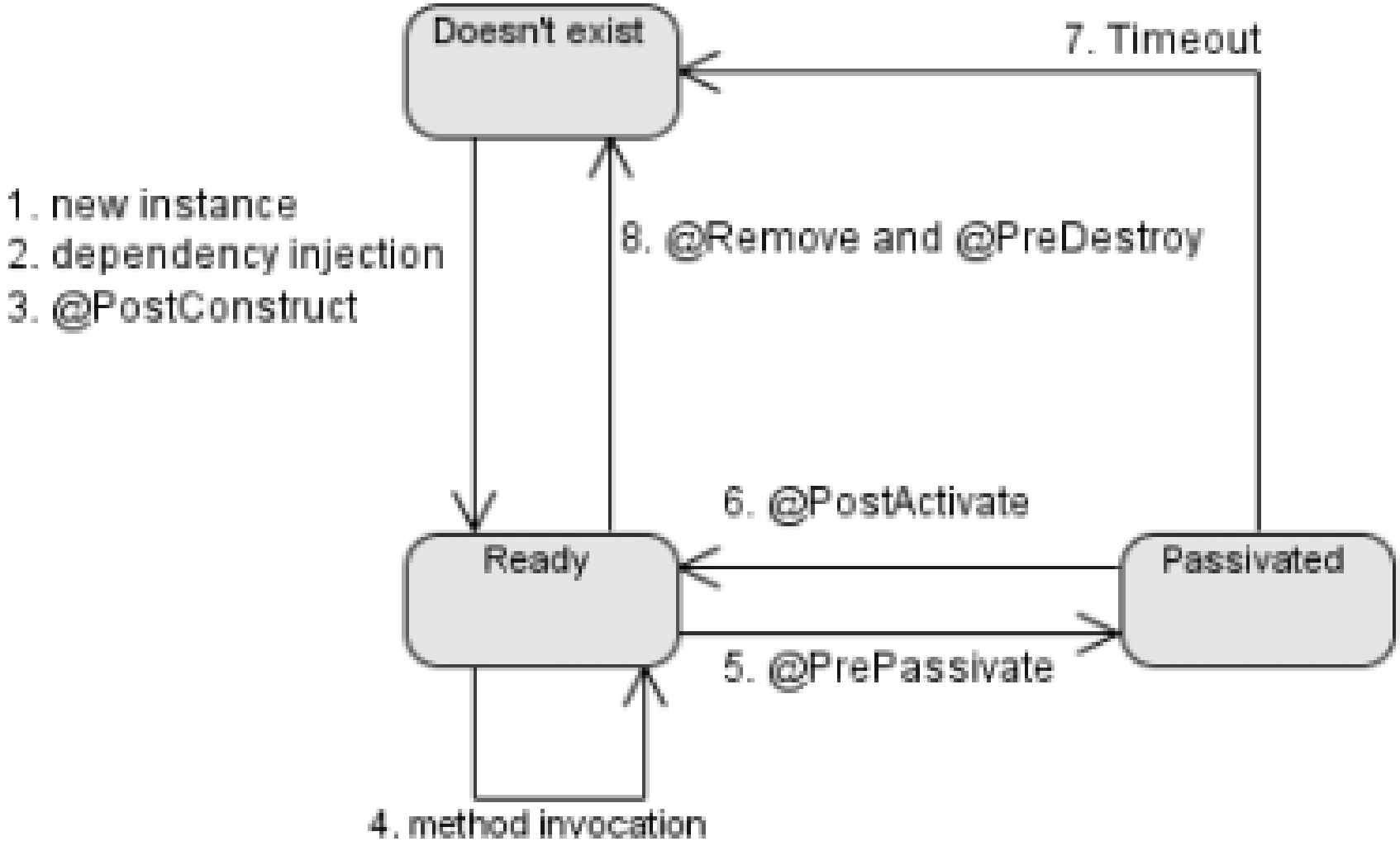
Ciclo de vida

- Dos de los servicios importantes provistos por el container para los EJB son el ciclo de vida y la intercepción de funciones
- Ciclo de vida significa que el Session Bean pasa por una serie de etapas definidas, las cuales pueden variar segun el tipo de Bean
- Los interceptors permiten agregar funcionalidad a los beans interceptados
  - Al estilo AOP, con los cross cutting concerns

Ciclo de vida: Stateless / Singleton



Ciclo de vida: Stateful



Callbacks

- Si bien el container maneja el ciclo de vida, este permite ejecutar código de negocio cada vez que un bean cambia de estado
- Este código, se denomina función de callback
- Para los Session Beans (según el tipo) tenemos cuatro tipos de callbacks, representados por cuatro anotaciones diferentes

Callbacks

- @PostConstruct
  - Marca un método que sera invocado justo después que la instancia del bean ha sido creada y la Inyección de dependencias ha sido resuelta
- @PreDestroy
  - Marca un método que sera invocado justo antes que la instancia del bean sea destruida
  - En el caso de un Stateful, este metodo se invoca después de los marcados con @Remove

Callbacks

- @PrePasivate
  - Marca un método que sera invocado cuando el container pasive una instancia
  - El método permite preparar el bean para poder ser sacado de la memoria
- @PostActivate
  - Marca un método que sera invocado luego de que el container active una instancia
  - Permite reinicializar recursos que el bean necesite

Callbacks

- Los métodos de callback tienen la siguiente forma:
  - void METODO() { ... }
- El método no puede retornan ni recibir parámetros
- No puede propagar Checked Exceptions, pero si Runtime Exceptions
- El método no puede ser static o final
- Solo podemos usar las anotaciones anteriores en un solo método, pero un método puede tener mas de una anotación

Callbacks

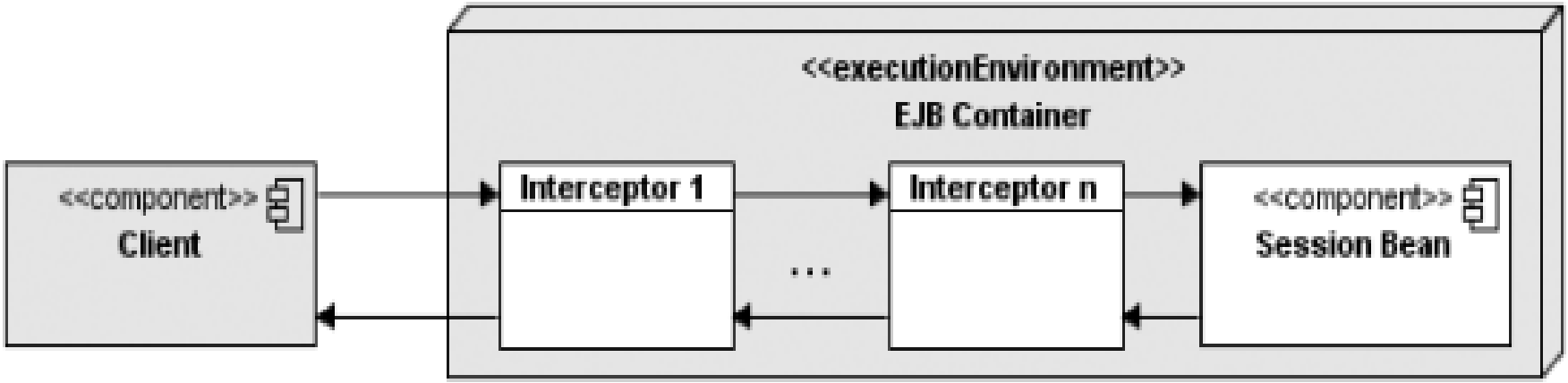
```
@Singleton
public class CacheEJB {
        private Map<Long, Object> cache = new HashMap<Long, Object>();

    >>  @PostConstruct
        private void initCache() {
                // Initializes the cache
        }

        public Object getFromCache(Long id) {
                if (cache.containsKey(id))
                        return cache.get(id);
                else
                        return null;
        }
}
```

Callbacks

```
@Stateful
public class ShoppingCartEJB {
        @Resource
        private DataSource ds;
        private Connection connection;
        private List<Item> cartItems = new ArrayList<Item>();

    >>  @PostConstruct
    >>  @PostActivate
        private void init() {
                connection = ds.getConnection();
        }
    >>  @PreDestroy
    >>  @PrePassivate
        private void close() {
                connection.close();
        }
        // ...
        @Remove
        public void checkout() {
                cartItems.clear();
        }
}
```

Interceptores

- Los interceptores pueden ser aplicados sobre Session Beans y Message Driven Beans



```
@Stateless
public class CustomerEJB {
        @PersistenceContext(unitName = "LIBRARYDS")
        private EntityManager em;
        private Logger logger = Logger.getLogger("com.javaee6");
        public void createCustomer(Customer customer) {
                em.persist(customer);
        }
        public Customer findCustomerById(Long id) {
                return em.find(Customer.class, id);
        }
    >>  @AroundInvoke
        private Object logMethod(InvocationContext ic) throws Exception {
                logger.entering(ic.getTarget().toString(),
                                ic.getMethod().getName());
                try {
                        return ic.proceed();
                } finally {
                        logger.exiting(ic.getTarget().toString(),
                                       ic.getMethod().getName());
                }
        }
}
```

Interceptores Around Invoke

- La forma del método de intercepción debe ser la siguiente

  @AroundInvoke

  Object <METHOD>(InvocationContext ic) throws Exception;

- El método no puede ser static o final
- El método debe devolver y aceptar los tipos especificados antes
  - Si es void el método interceptado, se devuelve null
- El método puede propagar Checked Exceptions

Method Interceptors

```
public class LoggingInterceptor {
        private Logger logger = Logger.getLogger("com.javaee6");
        @AroundInvoke
        public Object logMethod(InvocationContext ic) throws Exception {
                logger.entering(ic.getTarget().toString(),
                                ic.getMethod().getName());
                try {
                        return ic.proceed();
                } finally {
                        logger.exiting(ic.getTarget().toString(),
                                       ic.getMethod().getName());
                }
        }
}
```

Method Interceptors

```
@Stateless
public class CustomerEJB {

        @PersistenceContext(unitName = "LIBRARYDS")
        private EntityManager em;

        @Interceptors(LoggingInterceptor.class)
        public void createCustomer(Customer customer) {
                em.persist(customer);
        }
        public Customer findCustomerById(Long id) {
                return em.find(Customer.class, id);
        }
}
```

Method Interceptors

- En el caso anterior, el interceptor esta asociado a un solo método, **createCustomer**
- Podemos asociarlo a todos, o asociarlo a toda la clase

```
@Stateless
@Interceptors(LoggingInterceptor.class)
public class CustomerEJB {
        public void createCustomer(Customer customer) { ... }
        public Customer findCustomerById(Long id) { ... }
}
```

Method Interceptors

- Si queremos aplicar el interceptor a toda la clase, pero queremos excluir ciertos métodos, entonces hacemos...

```
@Stateless
@Interceptors(LoggingInterceptor.class)
public class CustomerEJB {
        public void createCustomer(Customer customer) { ... }
        public Customer findCustomerById(Long id) { ... }
        public void removeCustomer(Customer customer) { ... }

    >>  @ExcludeClassInterceptors
        public Customer updateCustomer(Customer customer) { ... }
}
```

Encadenamiento de Interceptores

- @Interceptors acepta una lista de interceptores a aplicar

```
@Stateless
@Interceptors(I1.class, I2.class)  <<
public class CustomerEJB {
        public void createCustomer(Customer customer) { ... }
    >>  @Interceptors(I3.class, I4.class)
        public Customer findCustomerById(Long id) { ... }
        public void removeCustomer(Customer customer) { ... }
        @ExcludeClassInterceptors
        public Customer updateCustomer(Customer customer) { ... }
}
```

Encadenamiento de Interceptores

- Si sobre una clase o método aplica mas de un interceptor, entonces se sigue la siguiente prioridad...