Introduction to Java EE

Introduction to CDI – Contexts and Dependency Injection
CDI overview

- A set of interlocking functionality:
  - typesafe **dependency injection**,
  - contextual **lifecycle management** for injectable objects,
  - **events**
  - **interceptors**, 
  - **decorators**, 
- Based around the philosophy of 
  - *loose coupling with strong typing*
Named beans

@Named
class HelloBean {
    public String hello() {
        return "Hello, World!";
    }
}

- Annotation @Named is used to give CDI managed class a name
  - the name matches class name with the first letter being lower case - helloBean
  - custom name can be specified: @Named("hello")
Dependency Injection

@SessionScoped // or other scopes
public class A {
    ...
}

public class B {
    @Inject
    private A a;

    public void mb() {
        a.ma();
        ...
    }
}

- @Inject looks at the component’s A scope, takes corresponding context and:
  - if finds instance of component A, uses that instance
  - otherwise creates a new instance of A, puts it to this context, and returns to component B
The scope of the bean defines:

- the lifecycle of each instance of the bean
- which clients share a reference to a particular instance of the bean

For example, if we have a @SessionScoped bean, CurrentUser:

- all beans that are called in the context of the same HttpSession will see the same instance (object) of CurrentUser
- this instance will be automatically created the first time a CurrentUser is needed in that session, and automatically destroyed when the session ends
For a given thread in a CDI application, there may be an active context associated with the scope of the bean.
- this context may be unique to the thread (for example, if the bean is @RequestScoped),
- or it may be shared with certain other threads (for example, if the bean is @SessionScoped)
- or even all other threads (if it is @ApplicationScoped).

Clients (for example, other beans) executing in the same context will see the same instance of the bean
But clients in a different context may see a different instance (depending on the relationship between the contexts)
CDI scopes available in Java EE

- JSF-based application may use these CDI scopes:
  - @RequestScoped
  - @ViewScoped (JSF specific)
  - @ConversationScoped
  - @FlowScoped (JSF specific)
  - @SessionScoped
  - @ApplicationScoped
  - @Dependent

- An instance of a @Dependent bean is never shared between different parents
  - It is instantiated when the parent is created, and destroyed when the parent is destroyed.
Application Context

Session Context

Conversation context (short)

Request Context

Conversation context (long-running)

Request Context

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean

bean
The Conversation Scope

- The *conversation scope* is a bit like the traditional session scope in that it holds state associated with a user of the system, and spans multiple requests to the server.
- However, unlike the session scope, the conversation scope:
  - is demarcated explicitly by programmer, and
  - holds state associated with a **particular web browser tab** in a Web application
- Browsers tend to share domain cookies, and hence the session cookie, between tabs
  - So one **session context spans multiple conversations**
Conversation demarcation

- CDI provides a built-in bean for controlling the lifecycle of conversations in a Web application. This bean may be obtained by injection:
  - `@Inject Conversation conversation;`
- To promote the conversation associated with the current request to a long-running conversation, call the `begin()` method from application code.
- To schedule the current long-running conversation context for destruction at the end of the current request, call `end()`.
@Named @ConversationScoped @Stateful
public class UseCaseController implements Serializable {
    @PersistenceContext(type = EXTENDED,
            synchronization = UNSYNCHRONIZED)
    private EntityManager em;

    @Inject @Getter
    private Conversation conversation;

    @Inject
    private CourseService courseService;
    @Inject
    private StudentService studentService;

    @Getter
    private Course course = new Course();
    @Getter
    private Student student = new Student();

    ...

Conversation Example (2)

// The first conversation step.
public void createCourse() {
    conversation.begin();
    courseService.create(course);
}

// The second conversation step.
public void createStudent() {
    studentService.create(student);
    student.getCourseList().add(course);
    course.getStudentList().add(student);
}

...
// The last conversation step.
public String ok() {
    try {
        em.joinTransaction();
        em.flush();
        // Success!
    } catch (OptimisticLockException ole) {
        // Please try again...
    } catch (PersistenceException pe) {
        // Finita la commedia...
    }
    conversation.end();
    ...
}

public String cancel() {
    conversation.end();
    ...
}
Conversation Example

- **Before** `conversation.begin()` the component acts as request scoped – its instances are destroyed at the end of each request
  - So called *transient conversation*
- **After** `conversation.begin()` the component instance becomes *conversation scoped*
  - So called *long-running conversation*
- **After** `conversation.end()` the component instance becomes request scoped and is destroyed as soon as current request ends
Events

- Dependency injection enables **loose-coupling** by allowing the implementation of the injected bean type to vary, either a deployment time or runtime.
- CDI Events go one step further, allowing beans to interact with **no compile time dependency at all**.
  - Event *producers* raise events that are delivered to event *observers* by the container.
Event observers and producers

- An *observer method* is a method of a bean with a parameter annotated `@Observes`:
  - Component A:
    - `public void onDocumentEvent(@Observes MyEvent myEvent) { ... }`

- Event producers fire events using an instance of the parameterized `Event` interface:
  - Component B:
    - `@Inject Event<MyEvent> myEventProducer;`
    - `myEventProducer.fire(new MyEvent(...));`

- Components A and B do NOT have to know one another! (No compile-time dependencies)
Events are synchronous!

- Firing an event:
  - The CDI container simply calls all the observer methods, passing the event object as the value of the event parameter.
    - Calls are synchronous!
  - If any observer method throws an exception, the container stops calling observer methods, and the exception is rethrown by the fire() method.
- The event object (payload) carries state from producer to consumer.
  - The event object is nothing more than an instance of a concrete Java class.
- `public class MyEvent { ... }`
Integration with EJB

- So, when should we use an EJB bean instead of a plain POJO bean?
- Whenever we need the **middleware services** offered by EJB, such as:
  - Transaction management,
  - Concurrency management,
  - Memory management (pooling and passivation),
  - Asynchronous methods
  - Timers,
  - Security,
  - Remote or web service invocation
Caution

- You CANNOT declare the scope of:
  - stateless session bean and
  - singleton session bean
- The EJB container controls the **full** lifecycle of these beans:
  - stateless == @Dependent
  - singleton == @ApplicationScoped
- A **stateful** session bean MAY have any scope