CONTENTS INCLUDE:

Java Enterprise Edition 6

The Most Elegant Enterprise Java Yet

By Andrew Lee Rubinger

ABOUT THE PLATFORM

Enterprise software development is inherently complex, and multi-user systems open the door to concerns such as transactional integrity, security, persistence integration, and interaction between components. Very simply put, the mission of the Java Enterprise Edition is to enable an out-of-the-box set of configurable services that allows the programmer to write less and focus on delivering clean business logic.

To this end, Java EE 6 is an aggregate of many interoperable technologies designed to deliver a unified experience. Application Servers that are certified to the standards defined by the Java Community Process are intended to service applications written to the specifications within the platform.

For the sake of brevity, this reference card will focus on the key APIs of Java EE 6 that are most relevant to modern development.

JAVA PLATFORM, ENTERPRISE EDITION 6 (JAVA EE 6)

JSR-316

This umbrella specification ties together the various subsystems that comprise the platform and provides additional integration support.

Profiles

New to Java EE 6 is the notion of the "Web Profile", a subset of the full specification that is targeted to cater to more specialized and minimal web-based application requirements. It is guaranteed to support the platform common annotations for injection and lifecycle (JSR-250), JNDI Naming standards, and the Java Transaction API. Additionally, the Web Profile focuses on Servlet and closely related technologies such as persistence, view presentation (JavaServer Faces and JavaServer Pages), and the business logic elements of EJB Lite.

Code Example

The simple class below is all it takes in Java EE 6 to define a POJO-based managed component capable of lifecycle callbacks, interceptors, and resource injection.

```
/**
 * Interceptor logging that an invocation has been received
 */
public class LoggingInterceptor {

    @AroundInvoke
    public Object intercept(InvocationContext context) throws Exception {
        System.out.println("Been intercepted: " + context);
        return context.proceed();
    }
}

/**
 * Defines a simple managed bean able to receive an injection
 */
@ManagedBean
@Interceptors({LoggingInterceptor.class}) // Invocations will be
   // intercepted
public class ComponentResourceInjection {

    @Resource
    private UserTransaction userTransaction;

    // ... business methods will be intercepted
    // by LoggingInterceptor
```

Reference

The full JavaDoc for the Java EE 6 API is located at:
http://download.oracle.com/javaee/6/api/

COMMON ANNOTATIONS FOR THE JAVA PLATFORM

JSR-250

he common annotations for Java EE are a shared package used throughout the platform specifications that generally focus on shared services like lifecycle, injection, and security.

Class Name	Description
Generated	Marks generated code
ManagedBean	Defines a class as a Java EE 6 Managed Bean
PostConstruct	Lifecycle callback after an instance has been created but before it is put into service
PreDestroy	Lifecycle callback before an instance is to be removed from service
Resource	Defines an injection point, marks that this is to be provided by the container
Resources	Allows for injection of N resources
DeclareRoles	Class-level target defining valid security roles
DenyAll	Marks that no roles are allowed to access this method
PermitAll	Marks that all roles are allowed to access this method (or all methods if applied to a class)
RolesAllowed	Specifies list of roles permitted to access this method (or all methods if applied to a class)

RunAs	Defines the identity context when run in the container

JAVA SERVLET 3.0

JSR-315

Servlet technology models the request/response programming model and is commonly used to extend HTTP servers to tie into server-side business logic. In Servlet 3.0, the specification has been expanded to include support for annotations, asynchronous processing, pluggability, and general ease-of-configuration.

Code Example

Because Servlet 3.0 includes annotations to define servlets, the descriptor web.xml is no longer required in most of the common cases; below is an example of all that's needed to create a simple servlet.

```
/**
 * Simple Servlet which welcomes the user by name specified
 * by the request parameter "name".
 */
@WebServlet(urlPatterns =
{"/welcome"}) // Will service requests to the path "/welcome"
public class WelcomeServlet extends HttpServlet
{
   /**
    * Inject a reference to an EJB
    */
   @EJB
   private WelcomeBean bean;

   /**
    * Service HTTP GET Requests
    */
   @Override
   protected void doGet(final HttpServletRequest request, final HttpServletResponse
response) throws ServletException,
         IOException
   {
      // Get the name from the request
      final String name = request.getParameter("name");

      // Precondition checks
      if (name == null)
      {
         response.sendError(HttpServletResponse.SC_BAD_REQUEST, "Request
parameter \"name\" is required");
      }

      // Set content type
      response.setContentType("text/plain");

      // Get the welcome message
      final String welcomeMessage = bean.welcome(name);

      // Write out
      response.getWriter().write(welcomeMessage);
   }

}
```

Public API from javax.servlet.annotation:

Class Name	Description
HttpConstraint	Defines security constraints for all HTTP-servicing methods in a secured servlet
HttpMethodConstraint	Defines security constraints for an HTTP-servicing method in a servlet
MultipartConfig	Indicates the servlet is to service requests for multipart/form-data MIME type
ServletSecurity	Secures a servlet class for HTTP-servicing methods
WebFilter	Defines a class as a servlet filter
WebInitParam	Specificies an initialization parameter on a servlet or filter
WebListener	Defines a class as a web listener
WebServlet	Defines a class as a servlet
PermitAll	Marks that all roles are allowed to access this method (or all method if applied to a class)
RolesAllowed	Specifies list of roles permitted to access this method (or all methods if applied to a class)
RunAs	Defines the identity context when run in the container

JAVA API FOR RESTFUL WEB SERVICES (JAX-RS)

JSR-311

New to Java EE, the JAX-RS specification allows for standard development adhering to the Representational State Transfer (REST) paradigm. These applications are packaged as a Servlet inside a web application archive.

```
/**
 * JAXB Model of a Name
 */
@XmlRootElement
public class Name {

    private String name;

    public Name(String name) {
        this.name = name;
    }
    public Name() {}

    public String getName() {return this.name;}

    public void setName(String name) {
        this.name = name;
    }
}

/**
 * Services requests for the path "/name"
 * returning the JAXB-formed Name XML
 */
@Path("/name")
@Produces({"application/xml"})
public class NamedResource extends javax.ws.rs.core.Application {
    @GET
    public Name get(@QueryParam("name") String name) {
        return new Name(name);
    }
}
```

...and HTTP GET requests to "{baseUrl}/myapp/name?name=andrew" will return the XML form for the Name.

Public API Annotation Selection from javax.ws.rs:

Class Name	Description
Consumes	Defines the media types that may be accepted
CookieParam	Injects the value of an HTTP cookie to a method param or bean/class field
DefaultValue	Defines default values for cookies and other parameters
DELETE	Flags that a method is to service HTTP DELETE requests
Encoded	Disables automatic decoding of parameter values
FormParam	Injects the value of a form parameter to a resource method parameter
GET	Flags that a method is to service HTTP GET requests
HEAD	Flags that a method is to service HTTP HEAD requests
HeaderParam	Injects the value of a header parameter
HttpMethod	Draws an association between an HTTP method with an annotation
MatrixParam	Injects the value of a URI matrix parameter
Path	Designates the URI path that a resource class or resource method will serve
PathParam	Injects the value of a URI path parameter

POST	Flags that a method is to service HTTP POST requests
Produces	Signals the media type(s) to be served by this resource
PUT	Flags that a method is to service HTTP PUT requests
QueryParam	Injects the value of a query parameter

CONTEXTS AND DEPENDENCY INJECTION FOR JAVA

JSR-299

The Java Contexts and Dependency Injection (CDI) specification introduces a standard set of application component management services to the Java EE platform. CDI manages the lifecycle and interactions of stateful components bound to well defined contexts. CDI provides typesafe dependency injection between components. CDI also provides interceptors and decorators to extend the behavior of components, an event model for loosely coupled components and an SPI allowing portable extensions to integrate cleanly with the Java EE environment. Additionally, CDI provides for easy integration with view layers such as JavaServer Faces 2.0 (JSR-314).

Code Example

Below is a simple example of a CDI Bean that is placed in scope alongside the HTTP session, given a String-based name binding (such that it may be used in JSF view components, for example) and injects the FacesContext such that it may be used in business methods.

```
@SessionScoped // Bind to the web session
@Named // To be used in view components by name
public class GreeterBean implements Serializable {

  @Inject // Used to integrate w/ JSF
  private FacesContext context;

  public GreeterBean() {}

  // ... business methods

}
```

Class Name	Description
javax.decorator.Decorator	Declares the class as a Decorator
javax.decorator.Delegate	Specifies the injection point of a Decorator
javax.enterprise.context.ApplicationScoped	Specifies the bean has application scope
javax.enterprise.context.ConversationScoped	Specifies the bean has conversation scope
javax.enterprise.context.Dependent	Specifies the bean belongs to dependent pseudo-scope
javax.enterprise.context.NormalScope	Specifies the bean is normally-scoped
javax.enterprise.context.RequestScoped	Specifies the bean is request-scoped
javax.enterprise.context.SessionScoped	Specifies the bean is session-scoped
javax.enterprise.event.Observes	Identifies an event parameter of an observer method
javax.enterprise.inject.Alternative	Specifies that the bean is an Alternative
javax.enterprise.inject.Any	Built-in qualifier type
javax.enterprise.inject.Default	Default qualifier type
javax.enterprise.inject.Disposes	Identifies the disposed parameter of a disposer method
javax.enterprise.inject.Model	Built-in stereotype for beans defining the model layer of an MVC webapp (ie. JSF)
javax.enterprise.inject.New	Built-in qualifier type
javax.enterprise.inject.Produces	Identifies a Producer method or field
javax.enterprise.inject.Specializes	Indicates that a bean specializes another bean
javax.enterprise.inject.Stereotype	Specifies that an annotation is a stereotype
javax.enterprise.inject.Typed	Restricts the types of a bean

Relevent Public Annotation API from JSR-330, Dependency Injection for Java in package javax.inject:

Class Name	Description
Inject	Identifies injectable constructors, methods, and fields
Named	String-based qualifier
Qualifier	Identifies qualifier annotations
Scope	Identifies scope annotations
Singleton	Identifies a type that the injector only instantiates once

For a deeper dive into CDI, check out DZone's CDI Refcard: http://refcardz.dzone.com/refcardz/contexts-and-depencency

BEAN VALIDATION 1.0

JSR-303

New to Java EE 6, the Bean Validation Specification provides for unified declaration of validation constraints upon bean data. It may be used to maintain the integrity of an object at all levels of an application: from user form input in the presentation tier all the way to the persistence layer.

Code Example

Here is an example of how to apply Bean Validation constraints in a declarative fashion to ensure the integrity of a User object.

```
public class User {
    @NotNull
    @Size(min=1, max=15)
    private String firstname;

    @NotNull
    @Size(min=1, max=30)
    private String lastname;

    @Pattern(regexp="\b[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,4}\b")
    public String email;

}
```

Public Annotation API for javax.validation

Class Name	Description
Constraint	Link between a constraint annotation and its constraint validation implementations
GroupSequence	Defines a group sequence
OverridesAttribute	Mark an attribute as overriding the attribute of a composing constraint
OverridesAttribute.List	Defines several @OverridesAttribute annotations on the same element
ReportAsSingleViolation	A constraint annotation hosting this annotation will return the composed annotation error report if any of the composing annotations fail.
Valid	Mark an association as cascaded

JAVASERVER FACES 2.0

JSR-314

JavaServer Faces is a user interface (UI) framework for the development of Java web applications. Its primary function is to provide a component-based toolset for easily displaying dynamic data to the user. It also integrates a rich set of tools to help manage state and promote code reuse.

Additionally, JSF is an extensible specification that encourages the authoring of custom user views. It's designed with tooling in mind such that integrated development environments (IDEs) can intelligently assist with design.

Code Example

Here we'll show a simple example of a JSF managed bean whose state is scope to the HTTP request, and is registered to a bean name that may be accessed by the view layer.

```
/**
 * This class defines a bean to live bound to the HTTP
 * request scope, and may be accessed by the view layer
 * under the name "hitchhikersGuide".
 */
import javax.annotation.PostConstruct;
import javax.enterprise.context.RequestScoped;
import javax.faces.application.ProjectStage;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.ManagedProperty;

@RequestScoped
@ManagedBean(name = "hitchhikersGuide")
public class HitchhikersGuide {
    private String ultimateAnswer;

    @ManagedProperty(value = "#{facesContext.application.projectStage}")
    private ProjectStage journeyStage;

    public String getUltimateAnswer() {
        return ultimateAnswer;
    }

    public void setUltimateAnswer(String ultimateAnswer) {
        this.ultimateAnswer = ultimateAnswer;
    }

    public ProjectStage getJourneyStage() {
        return journeyStage;
    }

    public void setJourneyStage(ProjectStage journeyStage) {
        this.journeyStage = journeyStage;
    }

    @PostConstruct
    public void findUltimateAnswerToUltimateQuestion() {
        ultimateAnswer = "42";
    }
}

index.xhtml View:

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
   xmlns:h="http://java.sun.com/jsf/html"
   xmlns:f="http://java.sun.com/jsf/core">
   <head>
      <title>Sample JSF Page</title>
   </head>
   <body>
   <f:view>
      <h2><h:outputText value="Test Managed Bean Annotations"/></h2>
      <p><h:outputText value="hitchhikersGuide.ultimateAnswer =
#{hitchhikersGuide.ultimateAnswer}"/></p>
      <h2><h:outputText value="Test Project Stage"/></h2>
      <p><h:outputText value="project stage = #{hitchhikersGuide.
journeyStage}"/></p>
   </f:view>
   </body>
</html>
```

Public API from javax.faces.bean:

Class Name	Description
ApplicationScoped	Denotes a ManagedBean is to be in application scope
CustomScoped	Denotes a ManagedBean is to be in a custom-defined scope of the specified value
ManagedBean	Defines a class as a ManagedBean type
ManagedProperty	Injection point for fields in ManagedBean classes
NoneScoped	Denotes a ManagedBean is to be in the "none" scope
ReferencedBean	Denotes the class as a referenced bean
RequestScoped	Denotes a ManagedBean is to be in request scope
SessionScoped	Denotes a ManagedBean is to be in session scope
ViewScoped	Denotes a ManagedBean is to be in view scope

Public API from javax.faces.application:

Class Name	Description
Application	Singleton scoped per web application to provide configuration for things like validators, components and converters
ApplicationFactory	Creates (if required) and returns Application instances
ApplicationWrapper	Application type which may be used by developers to add to an existing ResourceHandler
ConfigurableNavigationHandler	Extends NavidationHandler to provide runtime inspection of the NavigationCase which defines the rule base for navigation
FacesMessage	A single validation message
NavigationCase	A single navigation case in the navigation rule base
NavigationHandler	Handles navigation from a given action and outcome
Resource	Models a client resource request
ResourceHandler	API by which UIComponent and Renderer instances can reference Resource instances
ResourceWrapper	Simple base implementation of Resource to be extended by developers looking to add custom behaviors
StateManager	Coordinates the process of saving and restoring the view between client requests
StateManagerWrapper	Base implementation of a StateManager which may be subclasses by developers looking to add custom behaviors
ViewHandler	Pluggable point for the render response and restore view phases of the request lifecycle.
ViewHandlerWrapper	Base implementation of a ViewHandler which may be subclasses by developers looking to add custom behaviors.

ENTERPRISE JAVABEANS 3.1

JSR-318

Enterprise JavaBeans provide a component model to encapsulate business logic. As such, they provide a few bean types:

- Session Beans
- Stateless
- No conversational state between requests
- Stateful
- Same bean instance used to service each client/session
- Singleton
- One shared bean instance for all users
- Message-Driven
- Event Listener
- JCA endpoint
- Asynchronous
- Entity
- Integration point w/ Java Persistence

Code Example

It's simple to define an EJB capable of greeting a user.

```
/**
 * Stateless Session EJB
 */
@Stateless // Defines a Stateless Session Bean
@LocalBean // No-interface view
public class GreeterBean {
    @Override
    public String greet(String name) {
        return "Hi, " + name + "!";
    }
}
```

Other components, such as Servlets, may now inject a proxy to the above EJB:

@EJB private GreeterBean greeterEjb;

Public API Annotation Selection from javax.ejb:

Class Name	Description
AccessTimeout	Designates the amount of time a concurrent method should block before timing out
ActivationConfig-Property	Used to configure Message-Driven Beans
AfterBegin	Defines a Tx callback method for the "after begin" event
AfterCompletion	Defines a Tx callback method for the "after completion" event
ApplicationException	Defines a user exception which should not be wrapped
Asynchronous	Defines methods to be executed asynchronously.
BeforeCompletion	Defines a Tx callback method for the "before completion" event
ConcurrencyManage-ment	Configures the concurrency management mode for a singleton session bean
DependsOn	Designates initialization order for Singleton Session Beans
EJB	Defines an injection point and dependency upon an EJB component
EJBs	Plural injection point and dependency for EJB components, to be applied at the class-level
Local	Defines a local business interface view
LocalBean	Defines a no-interface view
Lock	Defines a concurrency lock for Singleton Session Beans using container-managed concurrency
MessageDriven	Defines a Message-Driven Bean
PostActivate	Defines an event callback after a Stateful Session Bean has been activated
PrePassivate	Defines an event callback before a Stateful Session Bean is to be passivated
Remote	Defines a remote business interface view
Remove	Defines a business method that should trigger the removal of a Stateful Session Bean instance (i.e., destroy the session)
Schedule	Defines a new timer to be created with a specified schedule
Schedules	Plural of Schedule
Singleton	Defines a Singleton Session Bean
Startup	Denotes that a Singleton Session Bean should eagerly load
Stateful	Defines a Stateful Session Bean
StatefulTimeout	Defines the amount of idle time before a Stateful Session is eligible for removal
Stateless	Defines a Stateless Session Bean
Timeout	Defines a timer expiration method
TransactionAttribute	Configures the transactional context under which business methods should execute
TransactionManage-ment	Configures transactional management for a Session or Message-Driven Bean (container or bean provided)

JAVA PERSISTENCE 2.0

JSR-317

Most enterprise applications will need to deal with persistent data, and interaction with relational databases can be a tedious and difficult endeavor. The Java Persistence specification aims to provide an object view of backend storage in a transactionally aware manner. By dealing with POJOs, JPA enables developers to perform database operations without the need for manually tuning SQL.

Code Example

A JPA entity is simply a POJO with some annotations to provide additional mapping metadata. For instance:

```
@Entity
public class SimpleEmployee {
   @Id @Auto
   private Long id;
   private String name;

   public Long getId(){ return id; }
   public void setId(final Long id) { this.id = id; }
   public String getName() { return name; }
   public void setName(final String name) { this.name = name; }
}
```

Now a managed component such as an EJB or CDI bean can interact with the database through our entity by associating it with an EntityManager.

```
@PersistenceContext
private EntityManager em;

public void createUser(){
  final SimpleEmployee andrew = new SimpleEmployee();
  andrew.setId(100L);
  andrew.setName("Andrew Lee Rubinger");
  em.persist(andrew); // Store in the DB
  // Now any state changes done to this managed object will be
  // reflected in the database when the Tx completes
}
```

Public API Annotation Selection from javax.persistence:

Class Name	Description
Basic	Describes mapping for a database column
Column	Specifies column mapping for a persistent property
DiscriminatorColumn	Notes the discriminator column used for SINGLE_TABLE and JOINED inheritance strategies
DiscriminatorValue	Specifies the value of the discriminator column for entities of this type
ElementCollection	Defines a collection of instances
Embeddable	Defines a class whose instances are stored as part of the owning entity
Embedded	Specifies a persistent property whose value is an instance of an embeddable class
EmbeddedId	Denotes composite primary key of an embeddable class
Entity	Defines a POJO as a persistent entity
EntityListeners	Specifies callback listeners
Enumerated	Specifies that a persistent property should be persisted as an enumeration

GeneratedValue	Specifies generation strategies for primary keys
Id	Denotes a primary key field
IdClass	Denotes a composite primary key class
Inheritance	Denotes the inheritance strategy for this given entity
JoinColumn	Specifies a column for joining an entity association or element collection.
JoinColumns	Plural of JoinColumn
JoinTable	Maps associations
Lob	Denotes a binary large object persistent property
ManyToMany	Defines an N:N relationship
ManyToOne	Defines an N:1 relationship
NamedQuery	Defines a static query in JPAQL
OneToMany	Defines a 1:N relationship
OneToOne	Defines a 1:1 relationship
OrderBy	Specifies ordering of elements when a Collection is retrieved
PersistenceContext	Used for injecting EntityManager instances
PersistenceUnit	Used for injecting EntityManagerFactory instances

PostLoad	Define an event callback method
PostPersist	Define an event callback method
PostRemove	Define an event callback method
PostUpdate	Define an event callback method
PrePersist	Define an event callback method
PreRemove	Define an event callback method
PreUpdate	Define an event callback method
Table	Specifies the primary table for this entity
Temporal	Denotes persistence of Date and Calendar field types
Transient	Denotes that a property is not persistent
Version	Specifies the persistent property used as the optimistic lock

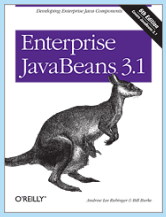
ABOUT THE AUTHOR

Andrew Lee Rubinger

Open Source Software Engineer and Author

Advocate for and speaker on testable enterprise Java development, author of "Enterprise JavaBeans 3.1" from O'Reilly Media. Member of the JBoss Application Server development team and technical lead of the ShrinkWrap project. Proudly employed by JBoss / Red Hat.

RECOMMENDED BOOK



Learn how to code, package, deploy, and test functional Enterprise JavaBeans with the latest edition of bestselling guide. Written by the developers of the JBoss EJB 3.1 implementation, this book brings you up to speed on each of the component types and container services in this technology, while the workbook in the second section provides several hands-on examples for putting the concepts into practice. Enterprise JavaBeans 3.1 is the most complete reference you'll find on this specification.

DZone, Inc.
140 Preston Executive Dr.
Suite 100
Cary, NC 27513

888.678.0399
919.678.0300

Refcardz Feedback Welcome
refcardz@dzone.com

Sponsorship Opportunities
sales@dzone.com

ISBN-13: 978-1-936502-40-0
ISBN-10: 1-936502-40-2
50795
9 781936 502400
$7.95

Version 1.0