Java 7, JEE 6, and OSGI
Agenda

- The Terminology
- Java SE 7
- Java EE 6
- Java Modularization with OSGi
Java, a Few Terms

- Java Platform is both
  - A Java Virtual Machine for a particular hardware and software platform that runs Java technology applications
  - The Java APIs that define the release

- J2SE, Java Standard Edition
  - The core functionality of the Java programming language expressed in an application programming interface (API)
  - The Java SE platform consists of a virtual machine, development tools, deployment technologies, and other class libraries and toolkits commonly used in Java

- Java EE, Java Enterprise Edition
  - The Java EE platform is built on top of the Java SE platform. The Java EE platform provides an API and runtime environment for developing and running large-scale, multi-tiered, scalable, reliable, and secure network applications.

http://www.ibm.com/developerworks/java/
What is Java 7? The themes

- **Major Java platform release, touching on all aspects of the language and JVM.** From the draft JSR document (*), the goals:
  - **Compatibility** - “Any program running on a previous release of the platform must also run *unchanged* on an implementation of Java SE 7”
  - **Productivity** - “…promote best coding practices and reduce boilerplate… minimal learning curve…”
  - **Performance** - “…new concurrency APIs… enable I/O-intensive applications by introducing a true asynchronous I/O API..”
  - **Universality** - “…accelerate the performance of dynamic languages on the Java Virtual Machine.”
  - **Integration** - “Java SE 7 will include a new, flexible filesystem API as part of JSR 203…”

(*) Available from jcp.org
New Language Constructs – Project Coin

- **It's a “bunch of small change(s)”**.
- **Strings in switch**

  ```java
  switch(myString) {
  case “one”: <do something>; break;
  case “red”: <do something else>; break;
  default: <do something generic>;
  }
  ```

- **Improved Type Inference for Generic Instance Creation (diamond)**

  ```java
  Map<String,MyType> foo = new Map<String,MyType>();
  ```

  **Becomes:**

  ```java
  Map<String,MyType> foo = new Map<>();
  ```
Project Coin continued...

- **An omnibus proposal for better integral literals**
  - Allow binary literals (0b10011010)
  - Allow underscores in numbers to help visual blocking (34_409_066)
  - Unsigned literals (0xDEu)

- **Simplified Varargs Method Invocation**
  - Moves warnings to method declaration, rather than on each user. Reduces unavoidable warnings.

```java
@SafeVarargs
class List<T> asList(T... a) {
    return new ArrayList<T>(a);
}
```
Coin – Automatic Resource Management

- Dealing with all possible failure paths is hard.
- Closing resources is hard.

Idea: Get the compiler to help, and define an interface on resources that knows how to tidy up automatically.

```java
try(InputStream inFile = new FileInputStream(aFileName);
    OutputStream outFile = new FileOutputStream(aFileName)) {
    byte[] buf = new byte[BUF_SIZE];
    int readBytes;
    while ((readBytes = inFile.read(buf)) >= 0) {
        inFile.write(buf, readBytes);
    }
```
Project Coin – Multi-Catch

- Developers often want to catch 2 exceptions the same way, but can't in Java 6:

```java
try {
    ...
    } catch(Exception a) {
        handle(a);
    } catch(Error b) {
        handle(b);
    }
```

- The following now works

```java
try {
    ...
} catch(Exception|Error a) {
    handle(a);
}
```
NIO.2 – New filesystem API

- **Address long-standing usability issues & boilerplate**
  - User-level modeling of more file system concepts like symlinks
  - File attributes modeled to represent FS-specific attributes (eg: owner, permissions...)
  - `DirectoryStream` iterates through directories
    - Scales very well, using less resources.
    - Allows glob, regex or custom filtering.
  - Recursive walks now provided, modeled on Visitor pattern.

- **Model entirely artificial file systems much like Windows® Explorer extensions**

- **File Change Notification**
  - Improves performance of apps that currently poll to observe changes.
Directory Visit - example

Files.walkFileTree(myPath, new SimpleFileVisitor<Path>() {
    public FileVisitResult visitFile(Path file,
        BasicFileAttributes attrs) {
        try {
            file.doWhatIWanted();
        } catch (IOException exc) {
            // failed to do op, do error handling here
        }
        return FileVisitResult.CONTINUE;
    }
});
java.util.concurrent updates

- As multicore becomes more prevalent, data structures and algorithms to match are key.

- **Major new abstraction:** *Fork/Join framework*
  - Very good at 'divide and conquer' problems
  - Specific model for parallel computation acceleration, significantly more efficient than normal Thread or Executor-base synchronization models.
  - Implements work stealing for lopsided work breakdowns

- **Other enhancements**
  - *TransferQueue* – model producer/consumer queues efficiently
  - *Phaser* – very flexible synchronization barrier
JSR 292 - invokedynamic

- **The JVM managed runtime is becoming home to more languages** (eg: jruby, jython, fan, clojure, etc..) **but is missing some of the fundamentals that help make those languages go fast.**

- **JSR 292** decouples method lookup and method dispatch
  - Get away from being purely Java (the language) centric.

- **Approach:** Introduce a new bytecode that executes a given method directly, and provides the ability at runtime to rewire what method that is.
  - Include a model for building up mutators (add a parameter, drop a parameter, etc..)
  - Ensure the JIT can efficiently exploit these constructs to ensure efficient code generation.
Smaller Items

- **ClassLoader changes**
  - Enable parallel classloading capability via new “safe” API.
  - URLClassLoader gains a close() method.
- **I18N - Unicode 6.0, Locale enhancement, Separate user locale and user-interface locale**
- **TLS 1.2** – Security updates.
- **JDBC 4.1** – ARM awareness.
- **Client (UI) updates**
  - Create new platform APIs for 6u10 graphics features
  - Nimbus look-and-feel for Swing
  - Swing JLayer component
  - XRender support
- **Update the XML stack**
Agenda

- The Terminology
- Java SE 7
- Java EE 6
  - Servlet 3.0
  - JAX-RS 1.1
  - CDI 1.0
  - EJB 3.0
  - Others
- Java Modularization with OSGi
Java EE Enhancement Themes

- Starting in Java EE 5, productivity, simplicity, and rightsizing became major themes

- Java EE5 ushered in widespread use of annotations
  - EE6 continued to expand the use of annotations

- Annotations enable a simpler POJO-based programming model in many cases

- Java EE6 also focused on rightsizing by deprecating older and little used technologies
  - JAX-RPC, EJB entity beans, JAXR

- Extensibility continues to be an important theme
Servlet 3.0 Annotations

- @WebServlet, @WebFilter, @WebListener annotations can replace web.xml configuration
- Promotes developer productivity
- Prior to Servlet 3.0, deployers were required to package a web.xml within the application:

```xml
<servlet>
    <servlet-name>myAnnotatedServlet</servlet-name>
</servlet>
<servlet-mapping>
    <servlet-name>myAnnotatedServlet</servlet-name>
    <url-pattern>/MyAnnotatedServlet</url-pattern>
</servlet-mapping>
```

- Developers can now simply annotate their Servlet class:

```java
@WebServlet(name="myAnnotatedServlet", urlPatterns="/MyAnnotatedServlet")
public class AnnotatedServlet extends HttpServlet {
```
Programmatic Configuration

- Programmatic methods such as addServlet dynamically configure at Web app initialization
- Allows one to dynamically customize the application based on configuration or other observable conditions
- Example:

```java
public class TestServletContextListener implements ServletContextListener {

    public void contextInitialized(ServletContextEvent servletContextEvent) {
        ServletContext context = servletContextEvent.getServletContext();

        if (context.getInitParameter("status").equals("VIP")) {
            context.addServlet("VIPSERVLET", "com.mybiz.VIPSERVLET");
        }
    }
}
```
Asynchronous servlets

- Supports a suspend/resume paradigm that allows you to detach a request/response from normal thread lifecycle
- Good for server push operations
- Improves scalability
- Uses and applications:
  - Asynchronous EJB method invocation
  - Accessing RESTful Web services
  - Chat
  - Quality of Service
Async Servlet Example

```java
@WebServlet("/foo" asyncSupported=true)
public class MyServlet extends HttpServlet {
    public void doGet(HttpServletRequest req, HttpServletResponse res) {
        ...  
        AsyncContext aCtx = request.startAsync(req, res);
        ScheduledThreadPoolExecutor executor = new ThreadPoolExecutor(10);
        executor.execute(new AsyncWebService(aCtx));
    }
}

public class AsyncWebService implements Runnable {
    AsyncContext ctx;
    public AsyncWebService(AsyncContext ctx) {
        this.ctx = ctx;
    }
    public void run() {
        // Invoke web service and save result in request attribute
        // Dispatch the request to render the result to a JSP.
        ctx.dispatch("/render.jsp");
    }
}
```
What is REST?

- Representational State Transfer, an architectural style described in Roy Fielding's doctoral dissertation
- Manipulate resource representations (nouns) defined at URIs with pre-defined methods (verbs)
- Use HTTP to its full capability

- A RESTful Web service is formed like a sentence – it simplifies how developers access services
  - Verb = HTTP Action (GET, POST, PUT, DELETE)
  - Noun = the URI of the Service (the document)
  - Adjective = MIME type of the resulting document

What is JAX-RS?

- JSR-311, new to Java EE 6

- Use annotations to declare services (resources) with helper classes to build implementation
  - Also use annotations to specify RESTful “contract”

- Takes the resource mapping logic out of the hands of the developer
  - Annotations contain necessary meta to map URI paths to resources

- Provides a pluggable system for resource rendering logic
@Path("/orders")
public class OrderResource {
    private static Map<Integer,Order> orders = new ConcurrentHashMap<Integer,Order>();
    private static AtomicInteger ids = new AtomicInteger();
    // root resource class must have a public constructor
    public OrderResource() throws NamingException {
    }
    @POST
    @Consumes(MediaType.APPLICATION_XML)
    public Response createOrder(Order order) {
        order.setId(ids.incrementAndGet());
        order.put(order.getId(), order);
        return Response.created(URI.create("/" + order.getId())).build();
    }
    @GET
    @Path("/{id}" )
    @Produces(MediaType.APPLICATION_XML)
    public Order getOrder(@PathParam("id") int id) {
        Order order = orders.get(id);
        if (order == null) {
            throw new WebApplicationException(Response.Status.NOT_FOUND);
        }
        return order;
    }
}
What is Context and Dependency Injection (CDI)?

- New spec (JSR 299) in EE6
- EE Standard that was Inspired by SEAM, Google Guice, Spring
- Uses annotations from JSR 330 (Dependency Injection for Java)
- Promotes loose coupling of components
- Provides extensible lifecycle contexts that components are bound to
- Provides integration with the Unified Expression Language
Defining a CDI Bean

Following a simple set of rules, any POJO may be treated as a CDI Bean

- Opt-in by deploying class inside of a Bean Deployment Archive. A EAR/WAR/JAR with a beans.xml in WEB-INF/ or META-INF/ directory
- Must have a no-argument constructor or constructor annotated @Inject
- Must be a concrete class (exception for @Decorator)
- Must not be a non-static inner class
Dependency Injection

- CDI supports injection into fields and methods
- Injection points are resolved using a combination of Type and Qualifiers

```java
class Order {
    @Inject @Selected Product product;  //field injection

    private Quantity quantity;

    @Inject  //Initializer method
    void setQuantity(Quantity q) {
        this.quantity = q;
    }
}
...
Enterprise Java Bean 3.0

- EJB extends Managed Beans model to provide complete annotated POJO programming model, including
  - EJB beans with no interface
  - Async method invocation extensions for long running work
  - Packaging in WAR file
  - Integration with UI technologies
## EJB 3.0: The Before and After

<table>
<thead>
<tr>
<th>2.x Code</th>
<th>3.0 Code</th>
</tr>
</thead>
</table>
| public interface ShoppingCart  
  extends EJBOBJECT {  
  public int someShoppingMethod()  
  throws RemoteException;  
}                          | public interface ShoppingCart {  
  public int someShoppingMethod();  
} |

| public class CartBean  
  implements SessionBean {  
  private float total;  
  private Vector productCodes;  
  
  public int someShoppingMethod() { … }  
}                                      | @Stateful public class CartBean implements ShoppingCart {  
  private float total;  
  private Vector productCodes;  
  
  public int someShoppingMethod() { … }  
} |

| public void ejbActivate() { }                                                     |  
| public void ejbPassivate() { }                                                   |
Singleton session beans

- New session bean type in which a single instance is created in the JVM
- Especially useful for caching commonly used data within the application
- Supports eager initialization during application startup
  - Ensures data availability once application starts servicing requests
- Provides for concurrency management via annotations
  - @ConcurrencyManagement(BEAN)
    e.g. public synchronized setProductInfo....
  - @ConcurrencyManagement(CONTAINER)
    • @Lock(LockType.READ)
    • @Lock(LockType.WRITE)
Asynchronous session bean invocations

- Allows EJB methods to run asynchronously
- Improves performance and increases scalability
- Has two modes:
  - Fire and forget
  - Fire and return results
Embeddable EJB Container

- In the past, testing EJB business logic required access to a configured Java EE application server environment

- This hampers developer productivity by increasing the cycle time between development and simple unit testing

- EJB 3.0 included a requirement for an embeddable EJB container

- The embeddable container only requires Java SE to run and as such provides an easy to obtain environment for many EJB unit testing scenarios

- The embeddable container supports EJB lite
public class EmbeddableContainerSample {

    public static void main(String[] args) throws Throwable {
        // Create a properties map to pass to the embeddable container:
        Map<String, String> properties = new HashMap<String, String>();
        // Specify that you want to use the WebSphere embeddable container:
        properties.put(EJBContainer.PROVIDER, "com.ibm.websphere.ejbcontainer.EmbeddableContainer");
        properties.put(EJBContainer.APP_NAME, "myappname");
        properties.put(EJBContainer.MODELES, "MyEJBModule");
        // Create the container instance, passing it our properties map:
        EJBContainer ec = EJBContainer.createEJBContainer(properties);
        MyBeanIface bean = (MyBeanIface) ec.getContext().lookup("java:global/MyEJBModule/MyBean!com.myCompany.MyBeanIface");
        // Invoke a method on the bean instance:
        bean.doStuff();
        // Close the embeddable container:
        ec.close();
    }
}
Packaging of EJBs in WAR files

- EJBs can now be packaged directly in a WAR module
- EJB can be defined via annotations or standard deployment descriptor
- EJB module can also exist in the lib directory of the WAR
- Provides a convenient packaging option when exploiting the new CDI capabilities in UEL
- Except for entity beans, EJB 2.x and 1.x content is supported in a WAR module.
JPA 2.0 Highlights

- Standard mappings enhancements for additional scenarios
  - Embedded collections
  - Multi-level embedding
  - Ordered lists

- Standardized query hints

- Standardized pessimistic locking behavior

- Standardized level 2 cache plugin and configuration

- Criteria based query API
  - On-the-fly dynamic queries
  - Query language independent (see Hibernate & JDO)

- Integration with Java EE Bean Validation
Java EE Bean Validation

- A **metadata model** for declaring the validation constraints of a JavaBean type

- An **application programming interface** (API) for determining whether a JavaBean instance violates any validation constraints declared for that JavaBean type

- Bean Validation requirements specified in JSR 303

```java
public class Stuart implements SmalleyBean {
    @Resource ValidatorFactory validatorFactory;
    @Resource Validator validator;

    @NotNull public String state = "acceptance";

    public Boolean validateMe(Stuart me) {
        Set<ConstraintViolation<Stuart>> cvSet = null;
        try {
            cvSet = validator.validate(me);
        }
        catch (Throwable t) {
            t.printStackTrace(); // The validator failed unexpectedly!
        }
        return cvSet != null && cvSet.size() == 0;
    }
    . . .
```
Web Service simplification with JAX-WS

@WebService
public class MyService {

    @Resource
    private WebServiceContext ctx;

    @Resource
    private SampleService svc;

    @WebServiceRef
    private SamplePort port;

    public String echo (String input) {
        ...
    }
}
JAX-WS 2.2 Updates

- Client Service factory method support for web services features for MTOM, Respect Binding, and Addressing

- Not returning a response to the requester when a WSDL-less WebServiceProvider implementation returns null

- Allowing a service to be packaged without the JAX-B wrapper bean classes

- Only exposing public, non-static, and non-final methods in services with an implicit SEI, based on the WebMethod and/or WebService annotations

- Supporting web service feature annotations for MTOM, RespectBinding, and Addressing on injected WebServiceRef port references

- Configuring WS-Addressing via WS-Policy assertions in WSDL
The whole is greater than the sum of the parts
Agenda

- The Terminology
- Java SE 7
- Java EE 6
- Java Modularization with OSGi
What is OSGi?

- "The dynamic module system for Java"
  - Mature 10-year old technology
  - Governed by OSGi Alliance: [http://www.osgi.org](http://www.osgi.org)
  - Used *inside* just about *all* Java-based middleware
    - IBM WebSphere, Oracle WebLogic, Red Hat JBoss, Sun GlassFish, Paremus Service Fabric, Eclipse Platform, Apache Geronimo, (non-exhaustive list)
How does OSGi help?

- Enforces architecture and simplifies maintenance
- Enables modular deployment
- Enables co-existence of multiple versions of libraries
  - Simplifies independent evolution of applications
  - Better separation of concern between application and middleware
- Enables truly dynamic update of modules within applications
OSGi Bundles and Class Loading

- **OSGi Bundle** – A jar containing:
  - Classes and resources.
  - OSGi Bundle manifest.

- **What’s in the manifest:**
  - Bundle-Version: Multiple versions of bundles can live concurrently.
  - Import-Package: What packages from other bundles does this bundle depend upon?
  - Export-Package: What packages from this bundle are visible and reusable outside of the bundle?

- **Class Loading**
  - Each bundle has its own loader.
  - No flat or monolithic classpath.
  - Class sharing and visibility decided by declarative dependencies, not by class loader hierarchies.
  - OSGi framework works out the dependencies including versions.

```manifest
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: MyService bundle
Bundle-SymbolicName: com.sample.myservice
Bundle-Version: 1.0.0
Bundle-Activator: com.sample.myservice.Activator
Import-Package: com.something.i.need;version=1.1.2
Export-Package: com.myservice.api;version=1.0.0
```
Declarative OSGi Services Using Blueprint

- XML Blueprint definition describes component configuration and scope
  - Optionally publish and consume components to/from OSGi service registry.
  - Standardizes established Spring conventions

- Simplifies unit test outside either Java EE or OSGi r/t.
**Blueprint Simplifies Service Dynamism**

- Dynamic service lifecycle is managed by the Blueprint container
- Service reference injected by container
  - service can change over time
  - can be temporarily absent without the bundle caring
How Do Enterprise Applications Use OSGi?

- *Enterprise OSGi* focuses on application concerns including web technologies, fine-grained component assembly and access to persistence frameworks
  - through exploitation of familiar Java EE technologies
  - in a manner suitable for a dynamic OSGi environment
  - using standards defined by the JCP and OSGi Alliance
  - introduces a multi-bundle Application archive
- Enables enterprise Application containers and deployment systems to provide better support for:
  - Sharing modules between applications
  - Multiple concurrent versions of modules
  - Dynamic update and extensions of applications
We appreciate your feedback.
Please fill out the survey form in order to improve this educational event.
WebSphere is fully Certified for Java SE and Java EE

http://www.oracle.com/technetwork/java/javaee/overview/compatibility-jsp-136984.html
# WebSphere Releases and Java Standards

<table>
<thead>
<tr>
<th>WebSphere</th>
<th>V4.0</th>
<th>V5.x</th>
<th>V6.0</th>
<th>V6.1</th>
<th>V7.0</th>
<th>V8.0</th>
<th>V8.5</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Standards</strong></td>
<td>J2EE 1.2</td>
<td>J2EE 1.3</td>
<td>J2EE 1.4</td>
<td>J2EE 1.4</td>
<td>JEE 5</td>
<td>JEE 6</td>
<td>JEE 6</td>
</tr>
<tr>
<td></td>
<td>J2SE 1.3</td>
<td>J2SE 1.3 (V5.0)</td>
<td>J2SE 1.4</td>
<td>J2SE 1.4</td>
<td>JSE 6</td>
<td>JSE 6</td>
<td>JSE 6/7</td>
</tr>
<tr>
<td><strong>IDE</strong></td>
<td>WebSphere Studio 5.0</td>
<td>WebSphere Studio 5.1</td>
<td>RAD V6.0</td>
<td>RAD V7.0</td>
<td>RAD V7.5</td>
<td>RAD V8.0</td>
<td>RAD V8.5</td>
</tr>
</tbody>
</table>

- **IDE**, Integrated Development Environment
- **J2EE**, Java 2 Platform, Enterprise Edition
- **J2SE**, Java 2 Platform, Standard Edition
- **JEE**, Java Enterprise Edition
- **JSE**, Java Standard Edition
- **RAD**, Rational Application Developer