
Java™

ORACLE

The Java EE 7 Platform Productivity++ and HTML5

Arun Gupta
Java EE & GlassFish Guy
blogs.oracle.com/arungupta
@arungupta

Java EE 6 – Key Statistics

- **50+ Million Java EE 6 Component Downloads**
- #1 Choice for Enterprise Developers
- #1 Application Development Platform
- Fastest implementation of a Java EE release


Java™ ORACLE

Java EE 7 Platform
Jun 12, 2013

Java™ ORACLE®

Java EE 7 Themes

- More annotated POJOs
- Less boilerplate code
- Cohesive integrated platform

- WebSockets
- JSON
- Servlet 3.1 NIO
- REST

- Batch
- Concurrency
- Simplified JMS

Top Ten Features in Java EE 7

1. WebSocket client/server endpoints
2. Batch Applications
3. JSON Processing
4. Concurrency Utilities
5. Simplified JMS API
6. `@Transactional` and `@TransactionScoped`
7. JAX-RS Client API
8. Default Resources
9. More annotated POJOs
10. Faces Flow

Java API for WebSocket 1.0

- Server and Client WebSocket Endpoint
  - Annotated: `@ServerEndpoint`, `@ClientEndpoint`
  - Programmatic: `Endpoint`
- Lifecycle methods
- Packaging and Deployment

```
@ServerEndpoint("/chat")
 public class ChatServer {
   @OnMessage
   public void chat(String m) {
     . . .
   }
 }
```

Java API for WebSocket 1.0

Chat Server

```
@ServerEndpoint("/chat")
public class ChatBean {
    static Set<Session> peers = Collections.synchronizedSet(…);

    @OnOpen
    public void onOpen(Session peer) {
        peers.add(peer);
    }

    @OnClose
    public void onClose(Session peer) {
        peers.remove(peer);
    }

    . . .
```

Java API for WebSocket 1.0

Chat Server (contd.)

```
    . . .

    @OnMessage
    public void message(String message, Session client) {
        for (Session peer : peers) {
            peer.getRemote().sendObject(message);
        }
    }
}
```

JSON Processing 1.0

- API to parse and generate JSON
- Streaming API
  - Low-level, efficient way to parse/generate JSON
  - Similar to StAX API in XML world
- Object Model API
  - Simple, easy to use high-level API
  - Similar to DOM API in XML world

Java API for JSON Processing 1.0

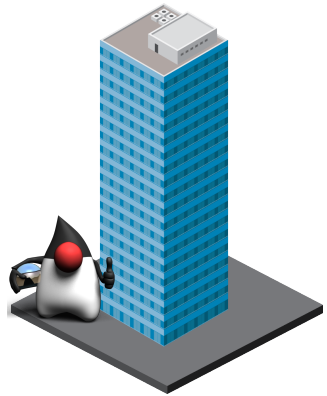
Streaming API

```
{
     "firstName": "John", "lastName": "Smith", "age": 25,
   "phoneNumber": [
      { "type": "home", "number": "212 555-1234" },
      { "type": "fax", "number": "646 555-4567" }
    ]
}
```

```
Iterator<Event> it = parser.iterator();
Event event = it.next();                    // START_OBJECT
event = it.next();                          // KEY_NAME
event = it.next();                          // VALUE_STRING
String name = parser.getString();           // "John"
```

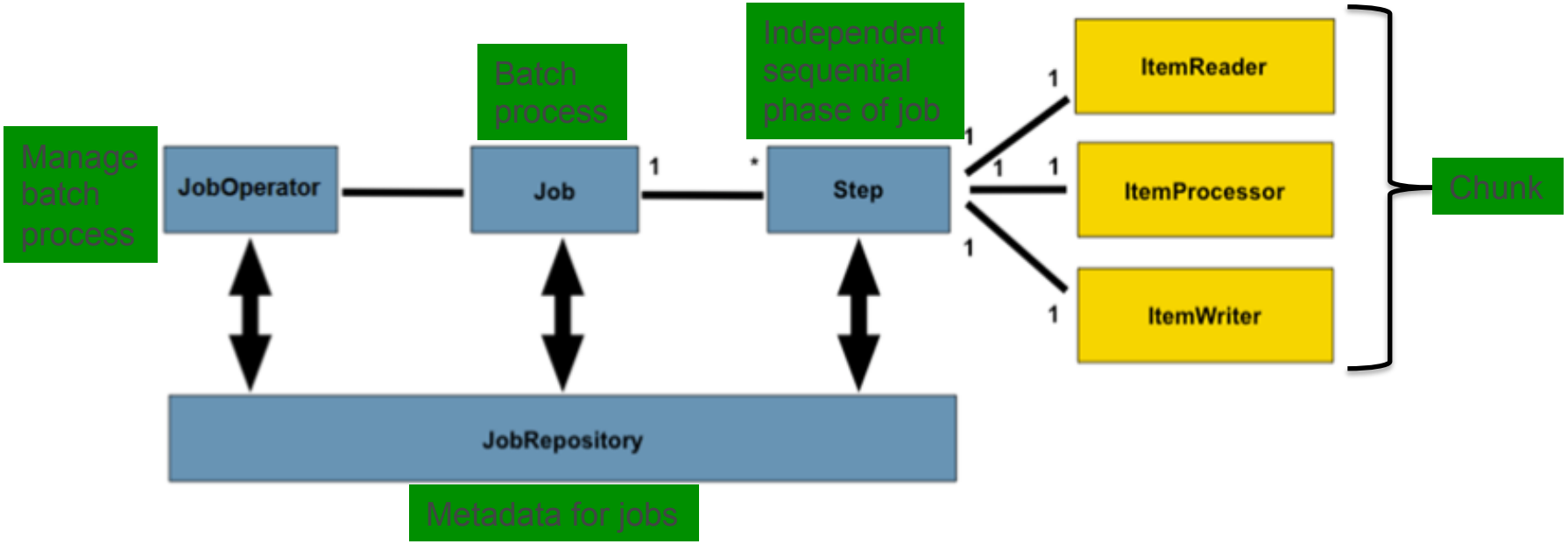
Batch Applications for Java Platform 1.0

- Suited for non-interactive, bulk-oriented, and long-running tasks
- Batch execution: sequential, parallel, decision-based
- Processing Styles
  - Item-oriented: Chunked (primary)
  - Task-oriented: Batchlet



Batch Applications 1.0

Concepts



Batch Applications 1.0

Chunked Job Specification

```
<step id="sendStatements">
  <chunk item-count="3">
    <reader ref="accountReader"/>
    <processor ref="accountProcessor"/>
    <writer ref="emailWriter"/>
</step>
```

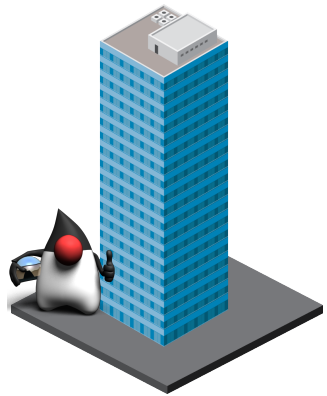
```
…implements ItemReader {
public Object readItem() {
    // read account using JPA
}
```

```
…implements ItemProcessor {
Public Object processItems(Object account) {
    // read Account, return Statement
}
```

```
…implements ItemWriter {
public void writeItems(List accounts) {
    // use JavaMail to send email
}
```

Concurrency Utilities for Java EE 1.0

- Extension of Java SE Concurrency Utilities API
- Provide asynchronous capabilities to Java EE application components
- Provides 4 types of managed objects
  - `ManagedExecutorService`
  - `ManagedScheduledExecutorService`
  - `ManagedThreadFactory`
  - `ContextService`
- Context Propagation



Concurrency Utilities for Java EE 1.0

Submit Tasks to ManagedExecutorService using JNDI

```
public class TestServlet extends HTTPServlet {
  @Resource(name="java:comp/DefaultManagedExecutorService")
  ManagedExecutorService executor;

  Future future = executor.submit(new MyTask());

  class MyTask implements Runnable {
    public void run() {
      . . .  // task logic
    }
  }
}
```

Java Message Service 2.0

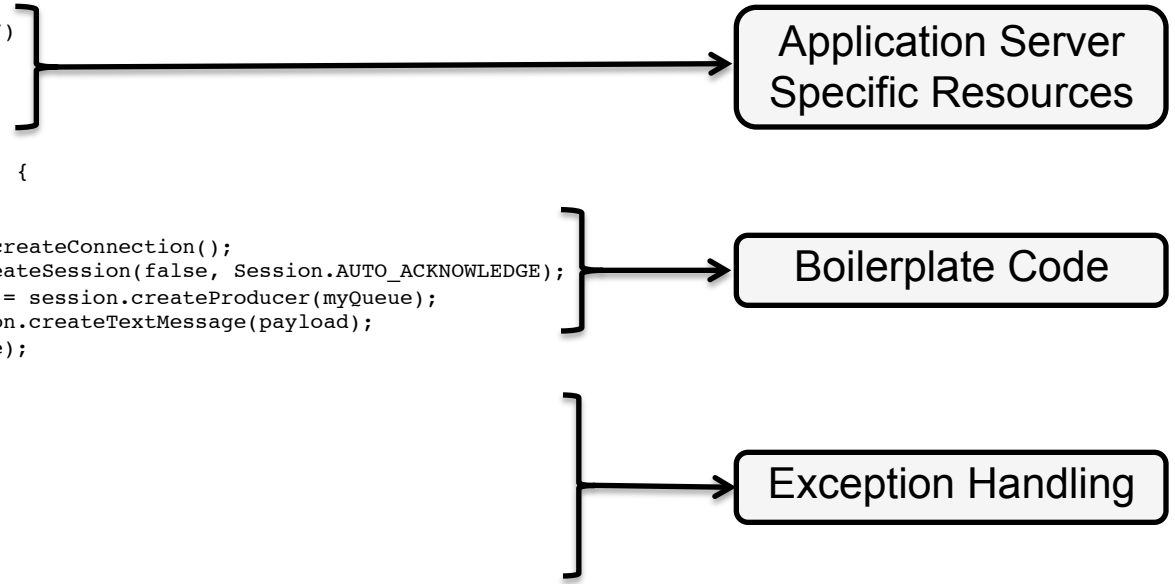
Get More from Less

- New `JMSContext` interface
- `AutoCloseable` `JMSContext`, `Connection`, `Session`, ...
- Use of runtime exceptions
- Method chaining on `JMSProducer`
- Simplified message sending

Java Message Service 2.0

Sending a Message using JMS 1.1

```
@Resource(lookup = "myConnectionFactory")
ConnectionFactory connectionFactory;

@Resource(lookup = "myQueue")
Queue myQueue;

public void sendMessage (String payload) {
    Connection connection = null;
    try {
        connection = connectionFactory.createConnection();
        Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
        MessageProducer messageProducer = session.createProducer(myQueue);
        TextMessage textMessage = session.createTextMessage(payload);
        messageProducer.send(textMessage);
    } catch (JMSException ex) {
        //. . .
    } finally {
        if (connection != null) {
            try {
                connection.close();
            } catch (JMSException ex) {

                //. . .
            }
        }
    }
}
```



Java Message Service 2.0

Sending a Message

```
@Inject
JMSContext context;


@Resource(lookup = "java:global/jms/demoQueue”)
Queue demoQueue;

public void sendMessage(String payload) {
    context.createProducer().send(demoQueue, payload);
}
```

Java API for RESTful Web Services 2.0

- Client API
- Message Filters and Entity Interceptors
- Asynchronous Processing – Server and Client
- Common Configuration

Java API for RESTful Web Services 2.0

Client API

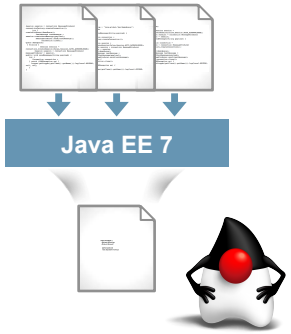
```
// Get instance of Client
Client client = ClientBuilder.newClient();

// Get customer name for the shipped products
String name = client.target("../orders/{orderId}/customer")
                    .resolveTemplate("orderId", "10")
                    .queryParam("shipped", "true")
                    .request()
                    .get(String.class);
```

Contexts and Dependency Injection 1.1

- Automatic enablement for beans with scope annotation and EJBs
  - “beans.xml” is optional
- Bean discovery mode
  - `all`: All types
  - `annotated`: Types with bean defining annotation
  - `none`: Disable CDI
- `@Vetoed` for programmatic disablement of classes
- Global ordering/priority of interceptors and decorators

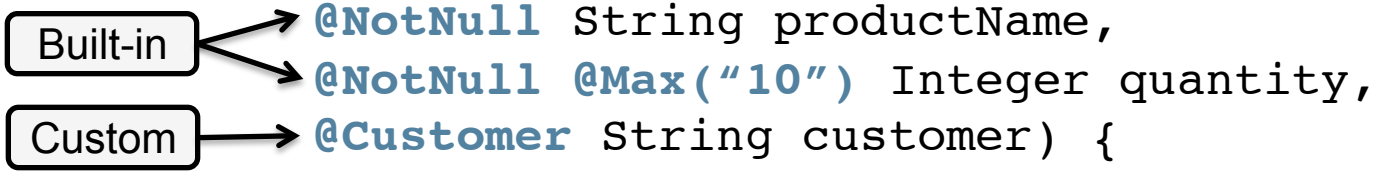
Bean Validation 1.1

- Alignment with Dependency Injection
- Method-level validation
  - Constraints on parameters and return values
  - Check pre-/post-conditions
- Integration with JAX-RS



Bean Validation 1.1

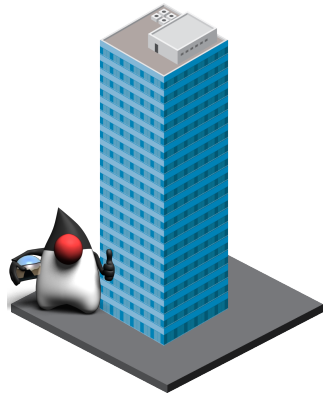
Method Parameter and Result Validation




```
public void placeOrder(
        @NotNull String productName,
        @NotNull @Max("10") Integer quantity,
        @Customer String customer) {
            //. . .
}

@Future
public Date getAppointment() {
    //. . .
}
```

Java Persistence API 2.1

- Schema Generation
  - `javax.persistence.schema-generation.*` properties
- Unsynchronized Persistence Contexts
- Bulk update/delete using `Criteria`
- User-defined functions using `FUNCTION`
- Stored Procedure Query



Servlet 3.1

- Non-blocking I/O
- Protocol Upgrade
- Security Enhancements
  - `<deny-uncovered-http-methods>`: Deny request to HTTP methods not explicitly covered

Servlet 3.1

Non-blocking I/O Traditional

```
public class TestServlet extends HttpServlet
  protected void doGet(HttpServletRequest request,
                       HttpServletResponse response)
                  throws IOException, ServletException {
    ServletInputStream input = request.getInputStream();
    byte[] b = new byte[1024];
    int len = -1;
    while ((len = input.read(b)) != -1) {
      . . .
    }
  }
}
```

Servlet 3.1

Non-blocking I/O: doGet

```
AsyncContext context = request.startAsync();
ServletInputStream input = request.getInputStream();
input.setReadListener(
    new MyReadListener(input, context));
```

Servlet 3.1

Non-blocking read

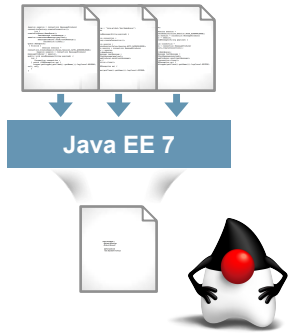
```
@Override
public void onDataAvailable() {
  try {
    StringBuilder sb = new StringBuilder();
    int len = -1;
    byte b[] = new byte[1024];
    while (input.isReady() && (len = input.read(b)) != -1) {
      String data = new String(b, 0, len);
      System.out.println("--> " + data);
    }
  } catch (IOException ex) {
    . . .
  }
}
```

JavaServer Faces 2.2

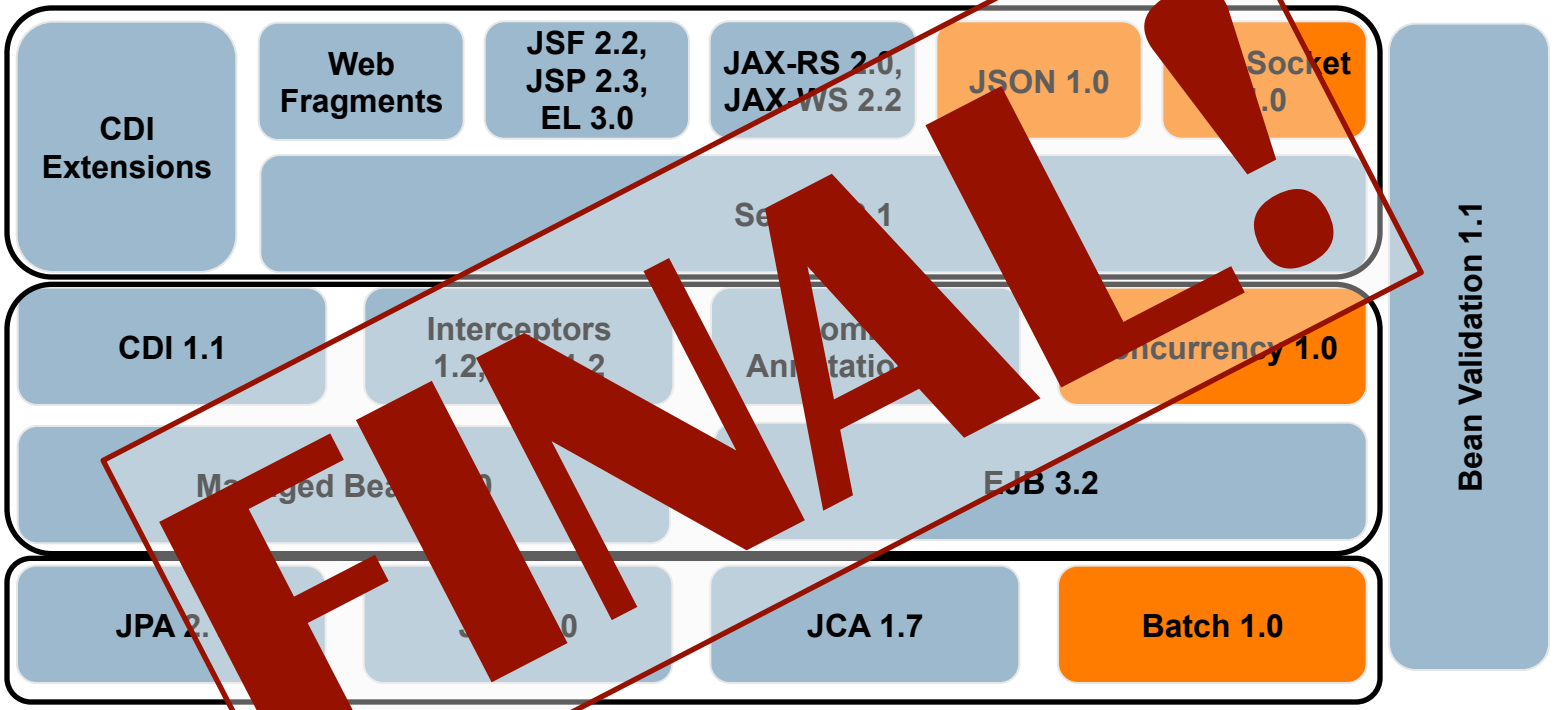
- Faces Flow
- Resource Library Contracts
- HTML5 Friendly Markup Support
  - Pass through attributes and elements
- Cross Site Request Forgery Protection
- Loading Facelets via `ResourceHandler`
- File Upload Component

Java Transaction API 1.2

- `@Transactional`: Define transaction boundaries on CDI managed beans
- `@TransactionScoped`: CDI scope for bean instances scoped to the active JTA transaction



Java EE 7 JSRs



Java™ ORACLE®

Jun 12

DOWNLOAD

Java EE 7 SDK

oracle.com/javaee

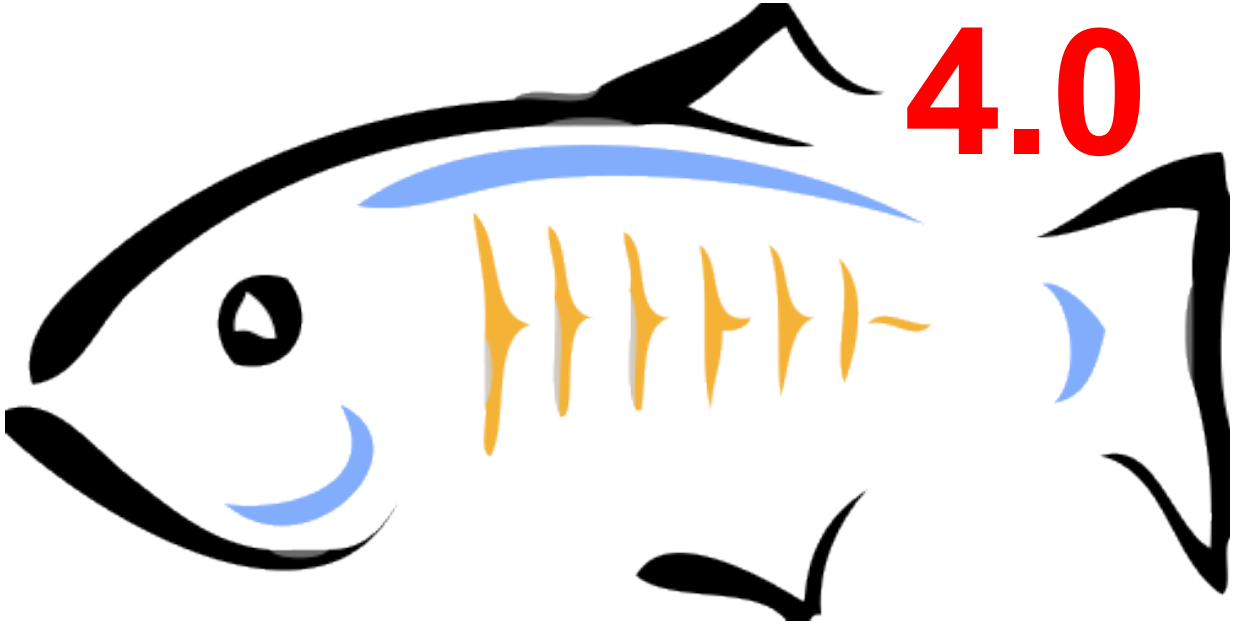
GlassFish 4.0

Full Platform or Web Profile

glassfish.org

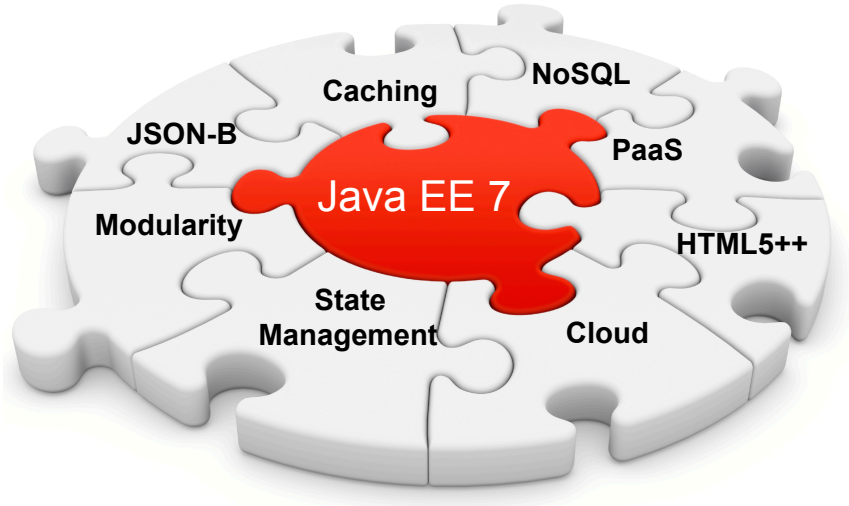
Java™ ORACLE®

Java EE 7 Implementation



download.java.net/glassfish/4.0/promoted/

Java EE 8 and Beyond



Adopt-a-JSR

Participating JUGs

Call to Action

- **Specs**: javaee-spec.java.net
- **Implementation**: glassfish.org
- **Blog**: blogs.oracle.com/theaquarium
- **Twitter**: @glassfish
- **NetBeans**: wiki.netbeans.org/JavaEE7


Java™