JPassion.com

JASMINE CONSEIL

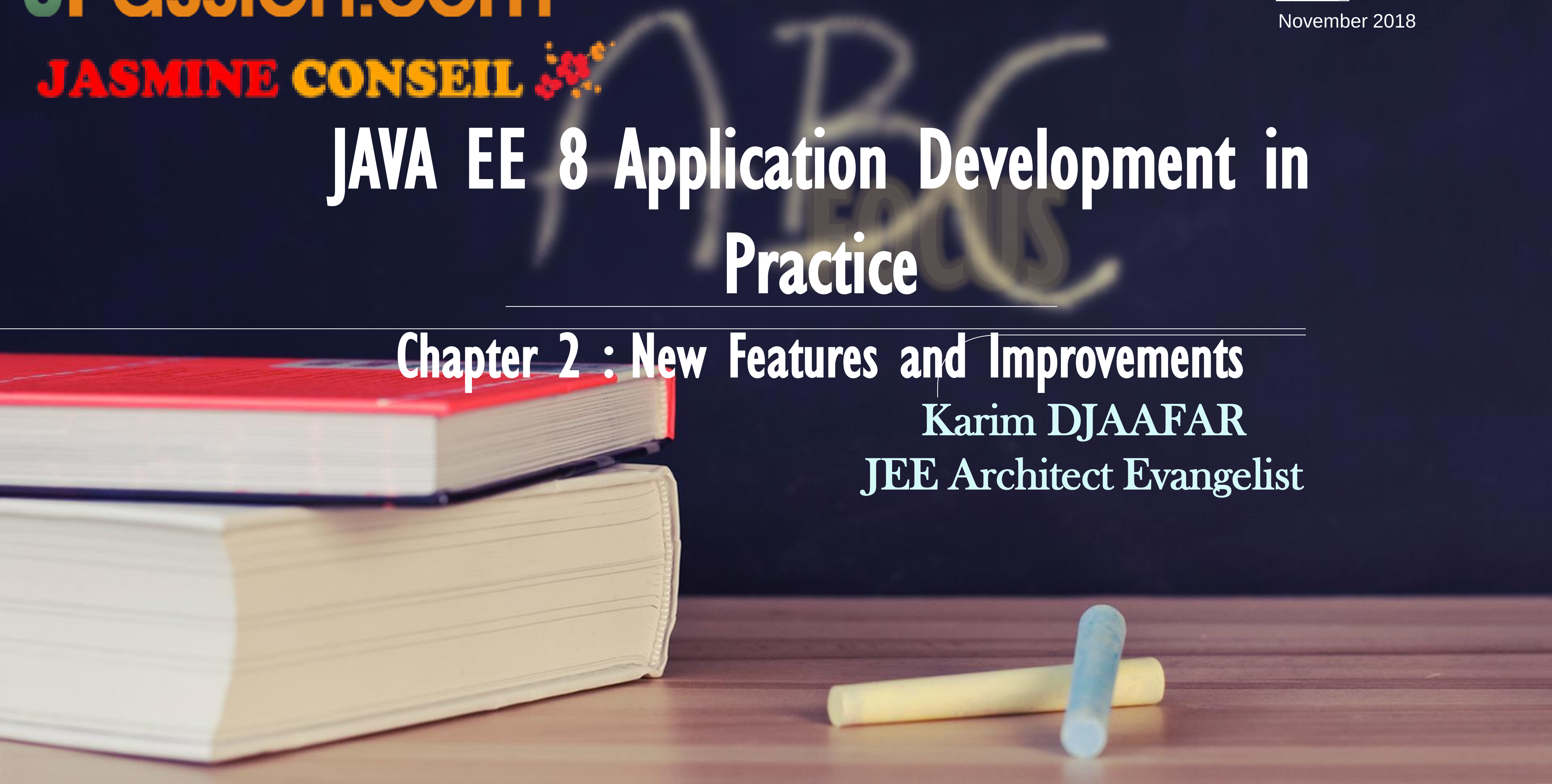
UPDATE!

November 2018

JAVA EE 8 Application Development in Practice

Chapter 2 : New Features and Improvements

Karim DJAAFAR
JEE Architect Evangelist



AGENDA

- Web Technologies and new APIs
- JAX-RS 2.1
- Java EE 8 Backend Technologies

Web Technologies

Overview

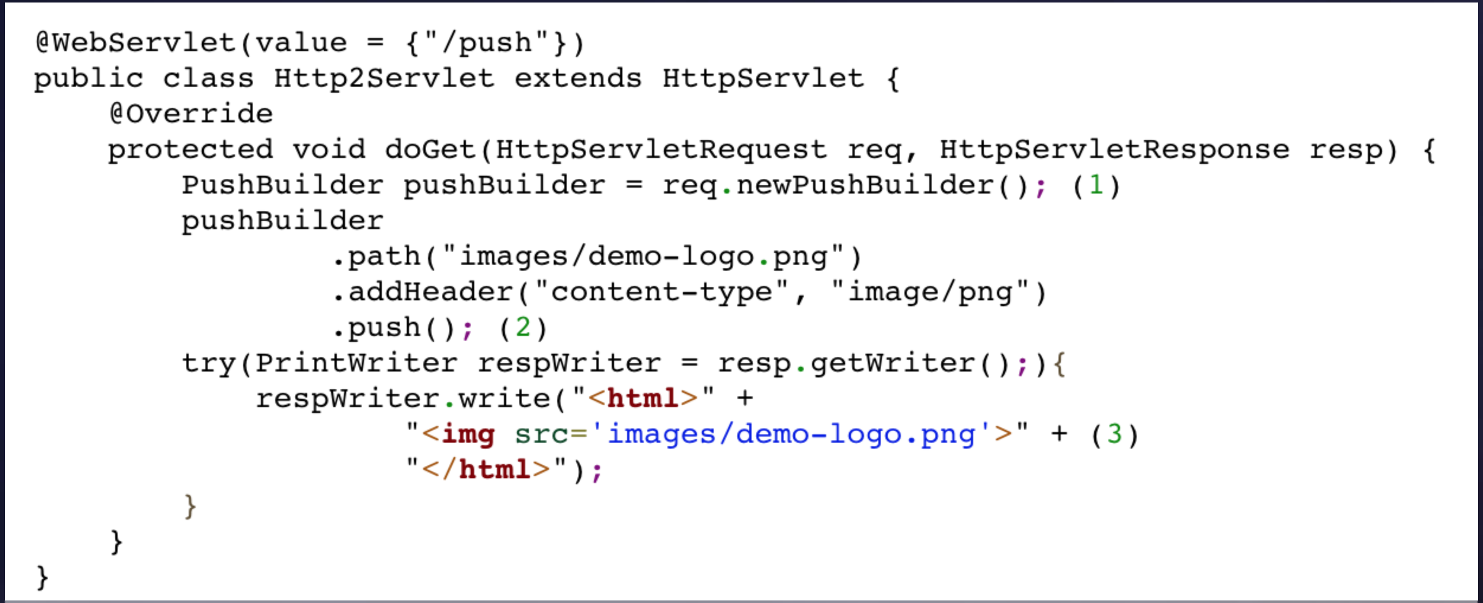
- **Servlet 4.0**: the basis of many frameworks technologies which offer a fairly low level request/response based API which serve the basis of many frameworks Java
- **JSF 2.3**: Stand for JavaServer Faces amore High-level API to create graphical user interfaces
- JSF is build in top of Servlets
- Offer a component model for the developer to work with
- In Java EE 8, servlets get a major revision

Servlet 4.0 (JSR 369)

- The Servlet version 4.0 specification represents the first major release since 2009 and embraces the capabilities afforded by the new digital HTTP 2.0 protocol
- The new HTTP protocol is the first for nearly 20 years and addresses many of the limitations of HTTP 1.0 and 1.1
- The headline feature in Servlet 4.0 is ***server push***
- This new capability is used to anticipate the resource requirements of the client by pushing those resources into the browser’s cache. So when the client receives the response the resources it needs are already in the cache

Server Push: Example

- The *PushBuilder* interface is designed for the HTTP/2 Server Push feature in the JSR 369, Java Servlet 4.0 specification
- It looks like below

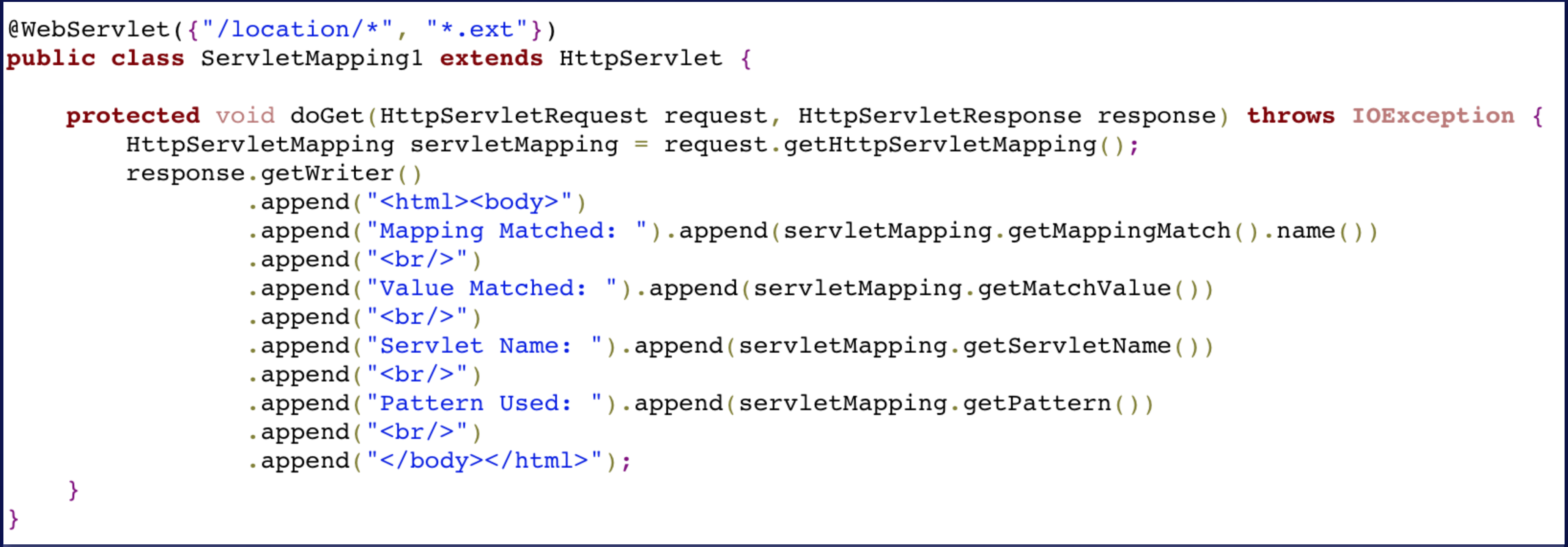


```
@WebServlet(value = {"/push"})
public class Http2Servlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) {
        PushBuilder pushBuilder = req.newPushBuilder(); (1)
        pushBuilder
                .path("images/demo-logo.png")
                .addHeader("content-type", "image/png")
                .push(); (2)
        try(PrintWriter respWriter = resp.getWriter();){
            respWriter.write("<html>" +
                    "<img src='images/demo-logo.png'>" + (3)
                    "</html>");
        }
    }
}
```

Runtime Discovery of Mappings

- A new API for the runtime discovery of URL mappings has been added, its goal is to make it easier to determine the mapping that caused the servlet to be activated
- The servlet mapping is obtained from the *HttpServletRequest* instance and has four methods:
  - *getMappingMatch()* : returns the type of the match
  - *getPattern()* : returns the URL pattern that activated the servlet request
  - *getMatchValue()* : returns the String that was matched
  - *getServletName()* : returns the fully qualified name of the servlet class that was activated with the request

Example



```java
@WebServlet({"/location/*", "*.ext"})
public class ServletMapping1 extends HttpServlet {

    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws IOException {
        HttpServletMapping servletMapping = request.getHttpServletMapping();
        response.getWriter()
                .append("<html><body>")
                .append("Mapping Matched: ").append(servletMapping.getMappingMatch().name())
                .append("<br/>")
                .append("Value Matched: ").append(servletMapping.getMatchValue())
                .append("<br/>")
                .append("Servlet Name: ").append(servletMapping.getServletName())
                .append("<br/>")
                .append("Pattern Used: ").append(servletMapping.getPattern())
                .append("<br/>")
                .append("</body></html>");
    }
}
```

Bean validation 2.0 (JSR 380)

- The role of bean validation is *cross-cutting*: it ensures data integrity from the client to the database by applying constraints to value in fields, return values and method parameters
- Among the enhancements are some useful constraints.
- They are:
  - @Email
  - @NotEmpty
  - @NotBlank
  - @Positive
  - @PositiveOrZero
  - @Negative
  - @NegativeOrZero
  - @PastOrPresent
  - @FutureOrPresent

Example

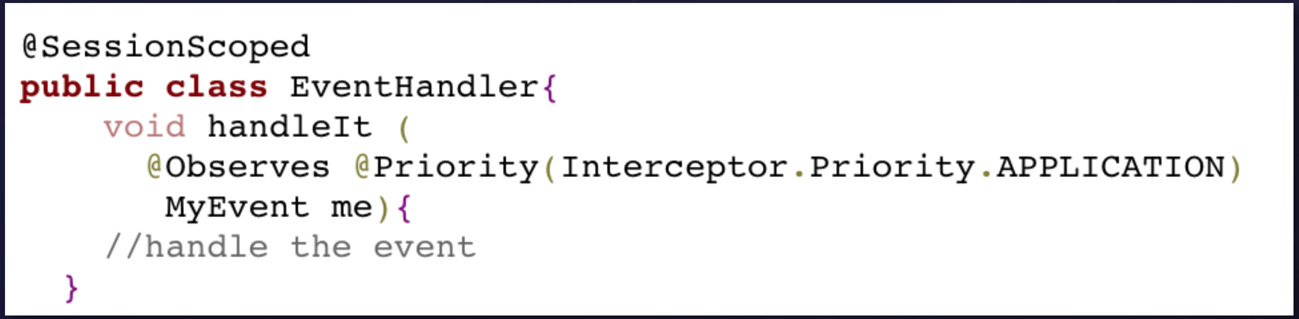
```java
public class Dates {

    @FutureOrPresent
    private Date past;

    @FutureOrPresent
    private LocalDate now;

    @FutureOrPresent
    private Year future;

    public Date getPast() {
        return past;
    }

    public void setPast(Date past) {
        this.past = past;
    }

    public LocalDate getNow() {
        return now;
    }

    public void setNow(LocalDate now) {
        this.now = now;
    }

    public Year getFuture() {
        return future;
    }

    public void setFuture(Year future) {
        this.future = future;
    }
}
```

Context and Dependency Injection for Java 2.0 (JSR 365)

- The Context and Dependency Injection API (CDI) is a backbone technology that has been in Java EE since version 6
- CDI brings several additions to the existing 1.1 version, The most relevant are relative to ***Observers and Events*** and ***Asynchronous events***

Observers and Events

- Event Ordering
  - Another new feature introduced in CDI 2.0 is the ability to specify in which order our observer methods handle CDI events.
  - This can be accomplished via the @Priority annotation, as illustrated in the following example



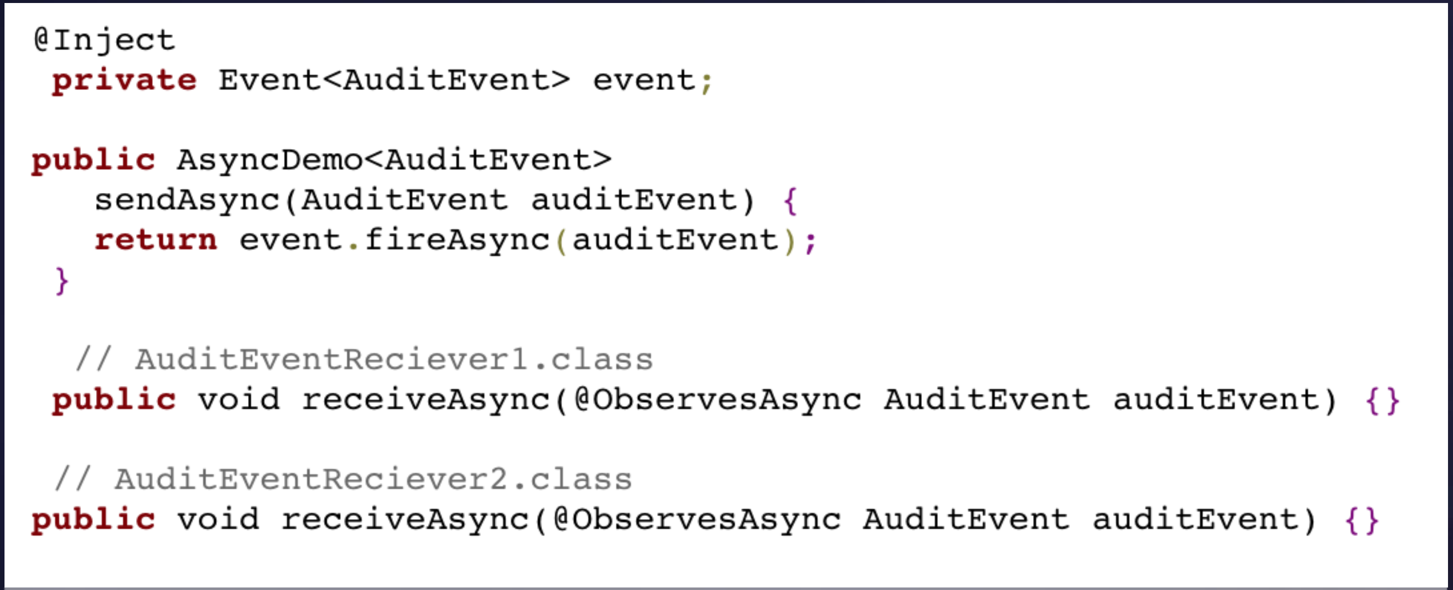
```
@SessionScoped
public class EventHandler{
    void handleIt (
      @Observes @Priority(Interceptor.Priority.APPLICATION)
       MyEvent me){
    //handle the event
  }
```

Async Events

- Another interesting addition to the observer feature is the capability to fire event asynchronously
- There is a new firing method that support this feature fireAsync() and a corresponding observer's annotation @ObserverAsync()

Async Events

- The code below shows the asynchronous firing of an AuditEvent and the observer methods that receive notification of the event asynchronously:



```
@Inject
 private Event<AuditEvent> event;

public AsyncDemo<AuditEvent>
   sendAsync(AuditEvent auditEvent) {
   return event.fireAsync(auditEvent);
 }

  // AuditEventReciever1.class
 public void receiveAsync(@ObservesAsync AuditEvent auditEvent) {}

 // AuditEventReciever2.class
public void receiveAsync(@ObservesAsync AuditEvent auditEvent) {}
```

New Java EE Security API (JSR 375)

- Completely new API
- Simplified Programming Model
- Layered APIs with delegation
- New Java EE Security API which supports relational and LDAP databases directly, and allows to integrate with custom identity stores, if necessary using *@LdapIdentityStoreDefinition* annotation

JSF 2.3 (JSR 372)

- Server push integration
- Multi fields validation
- Better integration for CDI
- Close alignment to Java 8
- WebSocket Support

Java API for RESTful Web Services 2.1 (JSR 370)

Introduction

- ***JAX-RS*** stands for ***JAVA API for RESTful Web Services***
- ***JAX-RS*** is a JAVA based programming language API and specification to provide support for created RESTful Web Services and become part of Java EE in version 6 of the Java EE specification
- The JAX-RS 2.1 release focuses on two main features:
  - Reactive Client API,
  - Server Sent Events

JAX-RS Reactive Client API

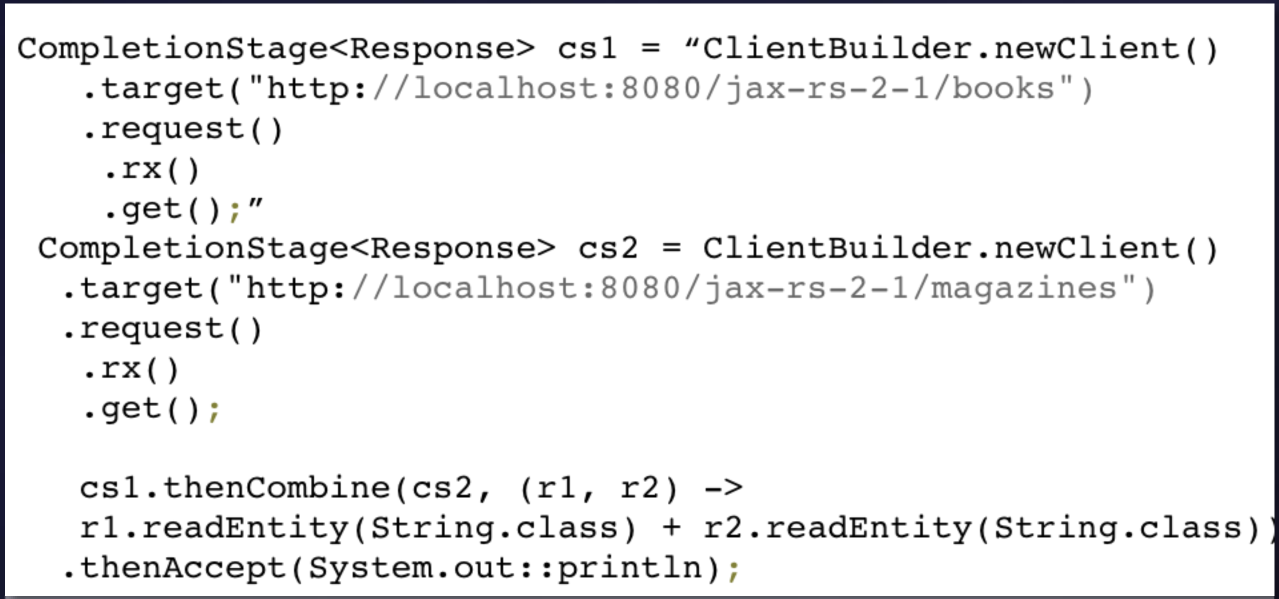
- The client API is a high level way to access web resources and has been a feature since release 1.1
- In version 2.1 it is given support for the reactive programming style
- The reactive JAX-RS 2.1 client integrates seamlessly with the asynchronous JAX-RS resources and Java 8+CompletableFuture
- **CompletableFuture<T>** is a new abstraction introduced in Java 8
- It extends Future<T> and adds callbacks support to handle event-driven work

CompletionStage support

- The new specification defines a new type of invoker named *RxInvoker*, and a default implementation of this type named CompletionStageRxInvoker
- *CompletionStageRxInvoker* implements Java 8's interface *CompletionStage*
- This interface declares a large number of methods dedicated to managing asynchronous computations

Sample

- The code snippet below, two calls are made to different endpoints and the result is combined:



```
CompletionStage<Response> cs1 = “ClientBuilder.newClient()
   .target("http://localhost:8080/jax-rs-2-1/books")
   .request()
    .rx()
    .get();”
 CompletionStage<Response> cs2 = ClientBuilder.newClient()
  .target("http://localhost:8080/jax-rs-2-1/magazines")
  .request()
   .rx()
   .get();

   cs1.thenCombine(cs2, (r1, r2) ->
   r1.readEntity(String.class) + r2.readEntity(String.class))
  .thenAccept(System.out::println);
```

Java EE 8 Backend Technologies

JPA 2.2 support

- The Java EE specification contains a complete stack to develop an enterprise application in the backend part combined with the frontend components that we have see in the preceding sections
- The Java Persistence API is a lightweight, POJO-based framework for Java persistence
- The JPA 2.2 which is the new Java Persistence API for Java EE 8 specification bring much power and flexibility for this specification

JPA 2.2 and EJB 3.2

- JPA 2.2 maintenance release was published by Oracle in June 2017
- In general changes in JPA 2.2, listed in the changelog file, included:
  - Ability to stream the result of a query execution
  - *@Repeteable* for all relevant annotations
  - Object-relational mapping Java 8 alignment
  - Allowing *AttributeConverters* to support CDI injection
- The JPA 2.2 changelog file can be found here:
  - https://jcp.org/aboutJava/communityprocess/maintenance/jsr338/ChangeLog-JPA-2.2-MR.txt”

JPA 2.2 and Stream Support

- Query: getResultStream
  - Get back a stream of results not only a list of results !

Example

```
Stream<reservations> reservations =
  em.createQuery(“SELECT b from reservations b”,
  Reservations.class).setMaxResults(10)
 .getresultStream();
```

JPA 2.2 Repeatable Annotations

- *@Repeatable(NamedQueries.class)* annotation
- JPA 2.2 introduced 2 new container annotations and 16 annotations became repeatable
- The new container annotations are *TableGenerators* and *SequenceGenerators* which store multiple *TableGenerator* and *SequenceGenerator* annotations.

JPA 2.2 Repeatable Annotations

- You can find some repeatable annotations in the following table:

Annotation	Description
AssociationOverride	Override the mapping for an entity relationship
AttributeOverride	Override the mapping of a Basic property
Convert	Activates or deactivates an AttributeConverter for a Basic property
JoinColumn	Defines a join column for an association or element collection

EJB 3.2 enhancements

- Since 2013, a final release of EJB 3.2 was also developed as part of Java EE 7
- New features of the Enterprise JavaBeans 3.2 release (EJB 3.2) included JNDI and EJB Lite

JSON-P 1.1 JSON-B 1.0 the new exchange format for Java EE

- JSON-P stands for JSON Processing and JSON-B stands for JSON Binding which is a new technology in Java EE
- JSON-B is all about mapping your specific Java objects to JSON and back
- The growing adoption of REST APIs with JSON as its preferred data-interchange format has led to better support of JSON in the platform, JSON now enjoys the same status as XML for support in Java EE
- We will see in our course of this new technology combined with JSON-P can make your JEE applications more powerful …

In Summary

JEE 8 is an enhanced platform platform

Java EE 8 is a major improvement of Java EE 7

As a developer working on Java EE solutions, these capabilities provide a big boost to one's productivity.

JPassion.com
Your friend to teach you
Java
With Passion !