

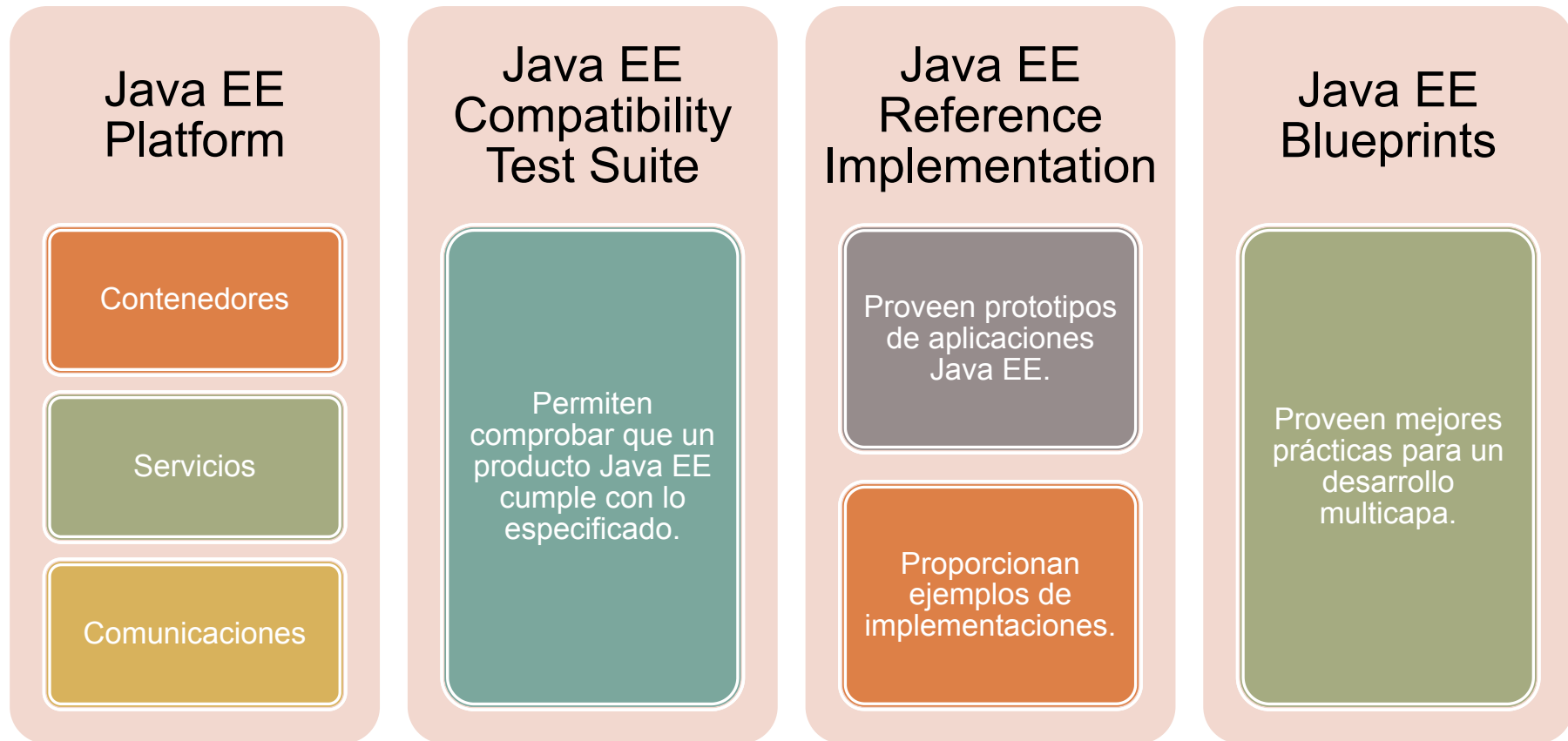
JAVA™ PLATFORM, ENTERPRISE EDITION (JAVA EE) 7

Principales Mejoras

Distribuciones de JAVA

- Java Standard Edition (SE)
 - ▣ Plataforma para desarrollar y ejecutar aplicaciones Java.
 - ▣ Base para todas las plataformas.
 - ▣ Desarrollo de aplicaciones de escritorio y applets.
- Java Enterprise Edition (EE)
 - ▣ Plataforma multiusuario y distribuida para desarrollo y despliegue de aplicaciones empresariales JAVA. Requiere tener instalado el JSE.
 - ▣ Arquitectura escalable, control de concurrencia, manejo de seguridad y transacciones.
- Java Micro Edition (ME)
 - ▣ Orientado a dispositivos móviles

Introducción sobre la especificación



JEE define 4 tipos de componentes

Enterprise JavaBeans (EJB) components

- Distributed, transactional components for business logic
- Also called business components on the business tier

Web components

- Servlets, JavaServer Faces, and JavaServer Pages
- Make up Web applications

Application clients

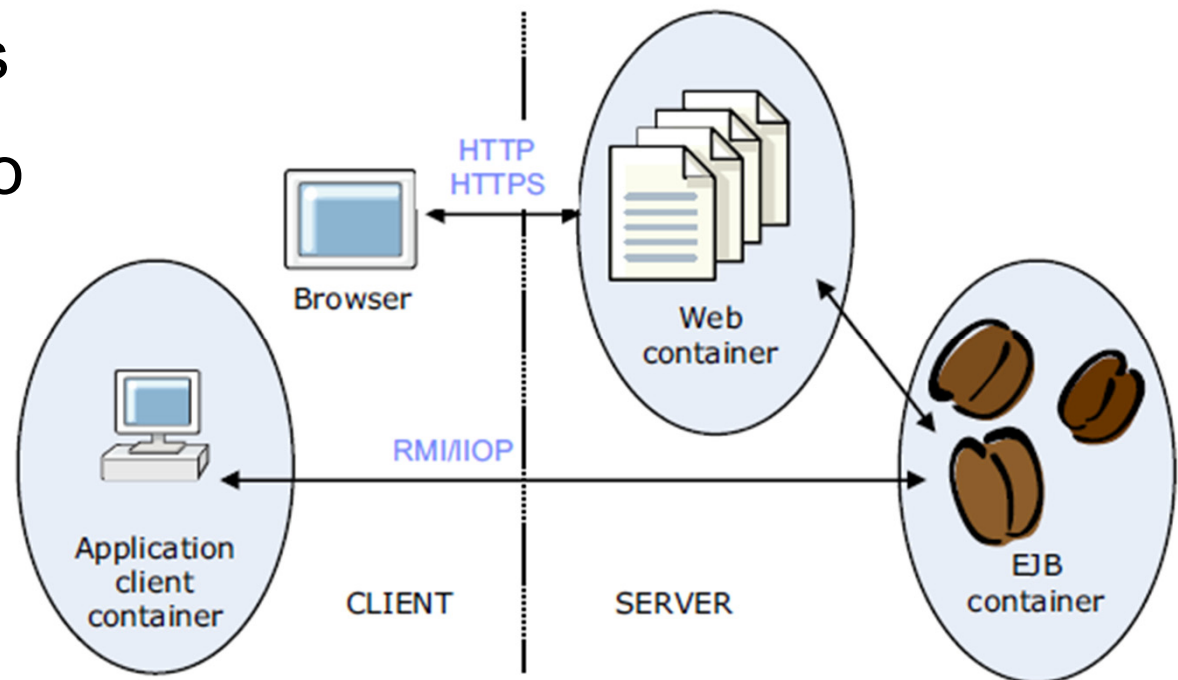
- Java programs that execute on a client machine and access other Java EE components

Applets

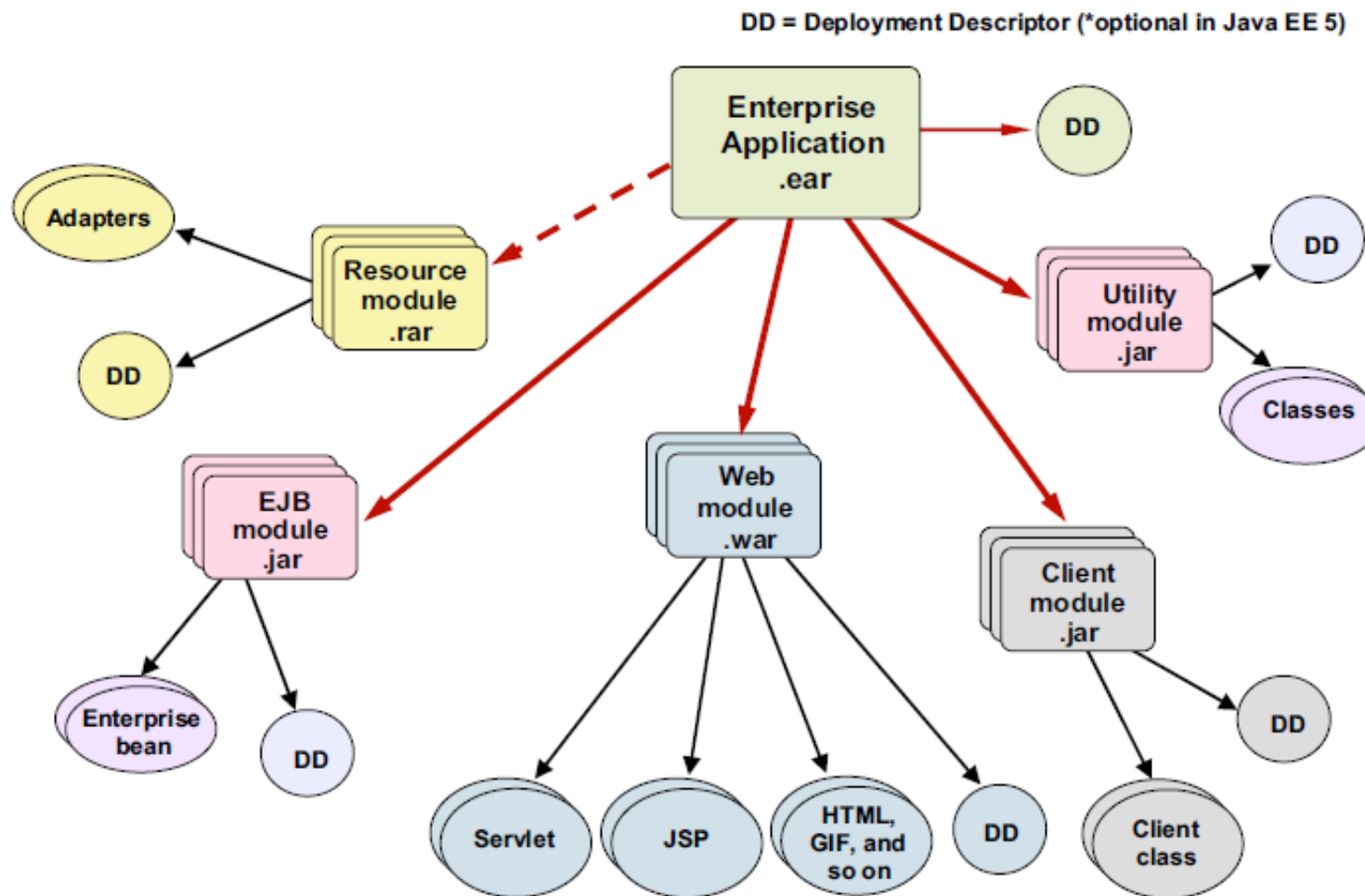
- Graphical Java components that typically execute within a browser

Java EE containers

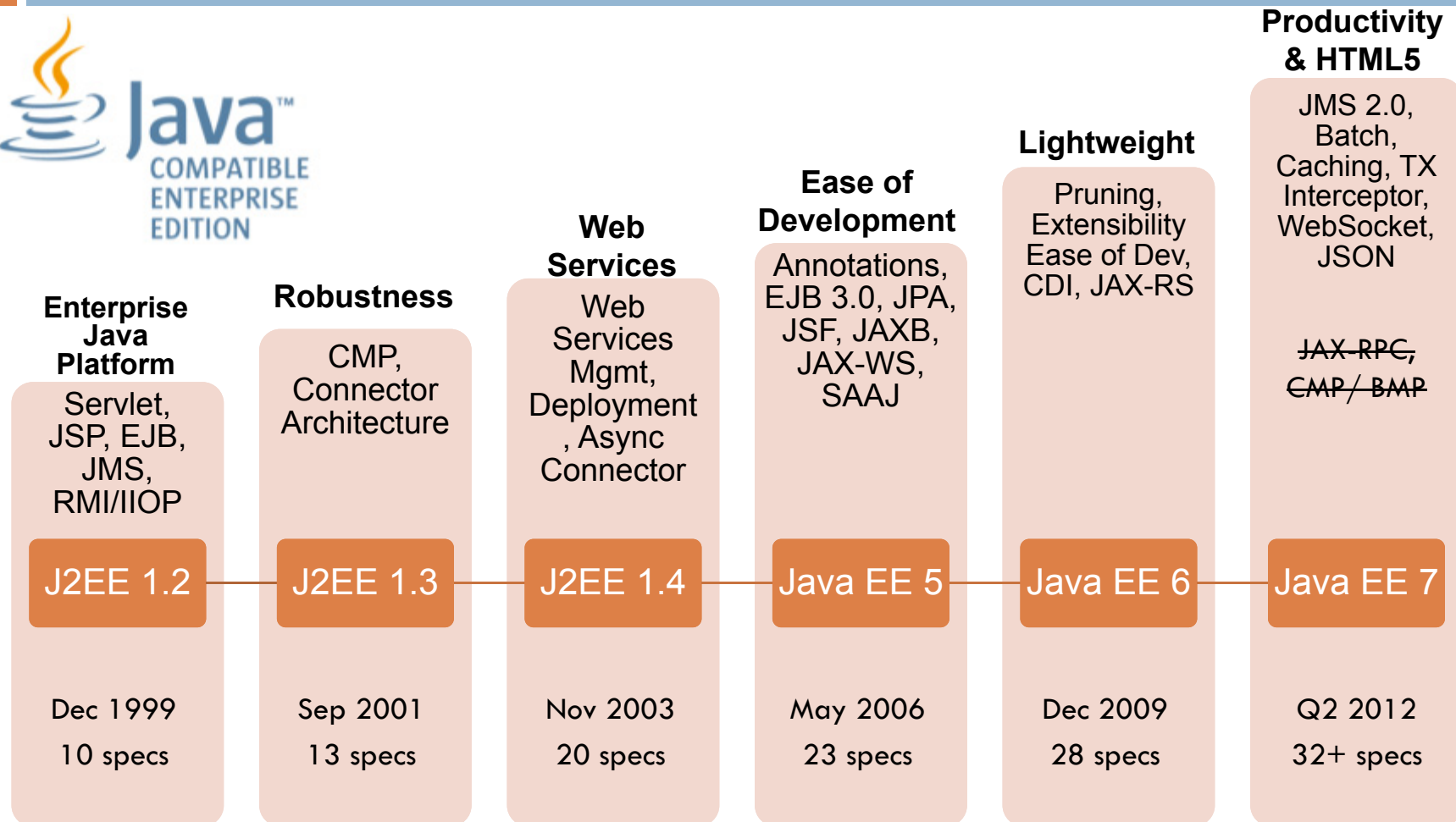
- Manejan la ejecución de los componentes.
- Permiten administrar:
 - ▣ Seguridad
 - ▣ Transacciones
 - ▣ Acceso remoto



Estructura de una aplicación enterprise



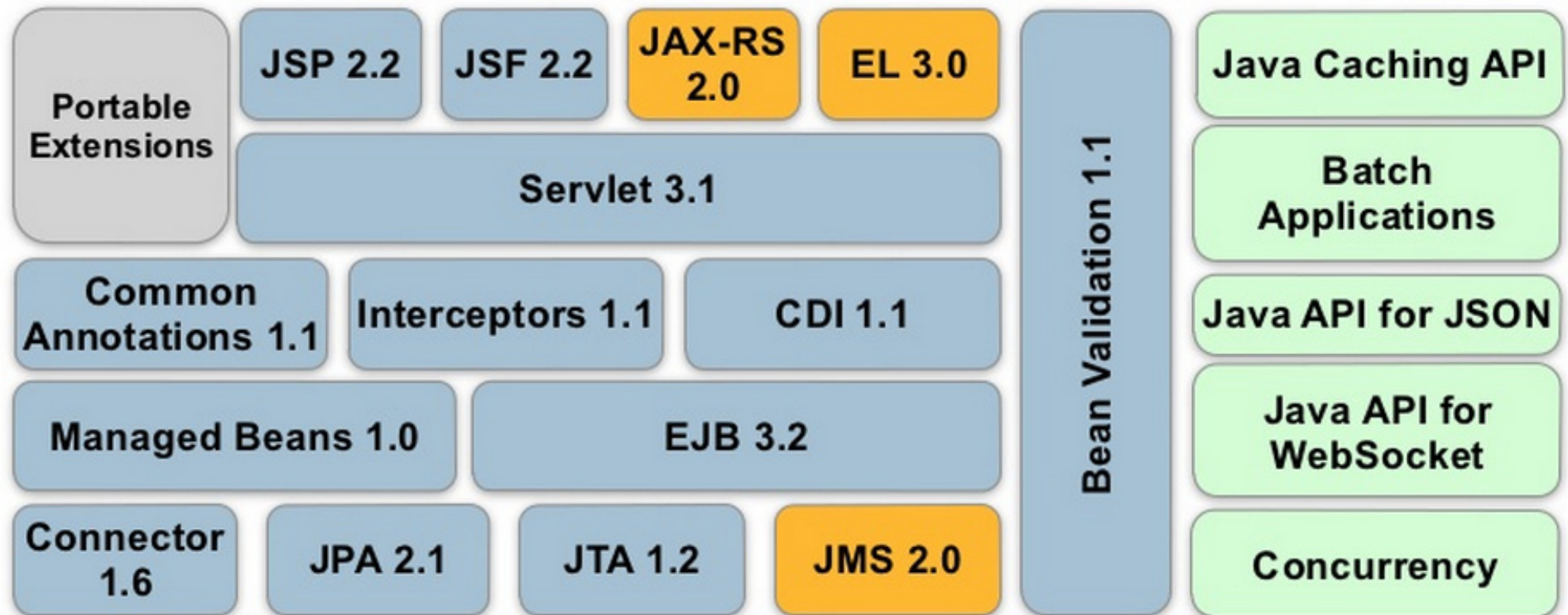
Evolución Java EE




Características principales en Java EE 6

1. EJB packaging in a WAR
2. Servlet and CDI extension points
3. Optional web.xml!
4. Type-safe dependency injection
5. CDI Events
6. JSF standardizing on Facelets
7. EJBContainer API
8. @Schedule!
9. EJB No Interface View
10. Web Profile

Especificaciones Java EE 7



 New

 Major Release

 Updated

JSON Processing 1.0

JSR 353

- Provee APIs para parsear, generar, transformar y hacer consultas Json.

Paquete	Descripción
<code>javax.json</code>	Provee un API orientado al modelo para procesar JSON .
<code>javax.json.spi</code>	Interfaz de proveedor de servicio(SPI) para conectarse con implementaciones de procesamiento de objetos Json.
<code>javax.json.stream</code>	Provee un API para parsear y generar JSON .

JSON Processing 1.0

Construire un objecto JSON

```
[  
  { "type": "home", "number": "212 555-1234" },  
  { "type": "fax", "number": "646 555-4567" }  
]
```

```
JSONArray jArray= Json.createArrayBuilder()  
    .add(Json.createObjectBuilder()  
        .add("type", "home")  
        .add("number", "212 555-1234"))  
    .add(Json.createObjectBuilder()  
        .add("type", "fax")  
        .add("number", "646 555-4567"))  
    .build();
```

```
JsonObject home = jArray.getJsonObject(0);  
String number = home.getString("number");
```

RESTful Web Services API

JAX-RS 2.0 - JSR 339

- Client API
- Filter and interceptor framework
- Asynchronous processing – server and client
- Hypermedia support
- Integration with Bean Validation

RESTful Web Services API

Ejemplo

```
@Path("/atm/{cardId}")
public class AtmService {
    @GET @Path("/balance")
    @Produces("text/plain")
    public String balance(@PathParam("cardId") String card,
        @QueryParam("pin") String pin) {
        return Double.toString(getBalance(card, pin));
    }

    @POST @Path("/withdrawal")
    @Produces("application/json")
    public Money withdraw(@PathParam("card") String card,
        @QueryParam("pin") String pin,
        String amount){
        return getMoney(card, pin, amount);
    }
}
```

RESTful Web Services API

Client API (1/2)

```
//get client
Client client = ClientFactory.newClient();

//get balance for a specific card with a PIN
String balance =
client.target("/atm/{cardId}/balance")
    .pathParam("cardId", "123456")
    .queryParams("pin", "3711")
    .request("text/plain")
    .get(String.class);
```

RESTful Web Services API

Client API (2/2)

```
//withdraw some money
```

```
Money mon =
```

```
    client.target("http://.../atm/withdraw")  
        .pathParam("card", "111122223333")  
        .queryParams("pin", "9876")  
        .request("application/json")  
        .post(text("50.0"), Money.class);
```

RESTful Web Services API

Filtros e Interceptors (1/2)

- Permiten extender una implementación de JAX-RS en términos de “logging”, confidencialidad, autenticación, compresión, etc.
- Dos interfaces:
 - ▣ Pre: `RequestFilter`
 - ▣ Post: `ResponseFilter`
- Parte de la cadena de filtros
- Cada filtro decide si continuar con el siguiente o detenerse
 - ▣ Al retornar `FilterAction.NEXT` o `FilterAction.STOP`

RESTful Web Services API

Filtros e Interceptors (2/2)

- Se define la clase y su anotación (CDI):

```
@Provider
```

```
@Logged
```

```
class LoggingFilter implements
```

```
    ContainerRequestFilter, ContainerResponseFilter {
```

```
    @Override
```

```
    public FilterAction preFilter(FilterContext ctx)
```

```
        throws IOException {
```

```
            logRequest(ctx.getRequest());
```

```
            return FilterAction.NEXT;
```

```
        }...
```

- En el servicio se llama a la anotación creada:

```
@Logged
```

```
@GET @Path("/balance")
```

```
@Produces("text/plain")
```

RESTful Web Services API

Integración con Bean Validation

```
@Path("/")
class MyResourceClass {
    @POST
    @Consumes(MediaType.APPLICATION_FORM_URLENCODED)
    public void registerUser(
        @NotNull @FormParam("firstName") String fn,
        @NotNull @FormParam("lastName") String ln,
        @Email @FormParam("email") String em) {
        ...
    }
}
```

RESTful Web Services API

Async - Servidor

```
@Path("/async/longRunning")
public class MyResource {
    @Context private ExecutionContext ctx;
    @GET @Produces("text/plain")
    public void longRunningOp() {
        Executors.newSingleThreadExecutor().submit(
            new Runnable() {
                public void run() {
                    Thread.sleep(10000); //Sleep 10 secs
                    ctx.resume("Hello async world!");
                }
            }
        );
        ctx.suspend(); //@Suspend connection and return
    } ... }
```

RESTful Web Services API

Async - Cliente

```
// Build target URI
Target target = client.target("http://.../balance")...

// Start async call and register callback

Future<String> handle =
    target.request().async().get(
        new InvocationCallback<String>() {
            public void complete(String balance) { ... }
            public void failed(InvocationException e) { ... }
        });

// After waiting for a while ...
if (!handle.isDone()) handle.cancel(true);
```

Java Message Service 2.0

JMS 2.0 - JSR 343

- API simplificada
 - ▣ Reduce líneas de código
 - ▣ Permite inyección de dependencia
 - ▣ No es un reemplazo de JMS 1.0
 - ▣ En lo posible prescinde JMSEException
- La conexión, sesión y otros objetos son autocerrados.

Java Message Service 2.0

Ejemplo JMS 1.0

```
@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

Ejemplo JMS 2.0 - JSR 343

```
@Inject
```

```
JMSContext context;
```

```
@Resource(mappedName="myQueue")
```

```
Queue myQueue;
```

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

Java Persistence API 2.1

Java™ Persistence 2.1 - JSR 338

- ❑ @NamedStoredProcedureQuery, StoredProcedureQuery
- ❑ Actualizar/Eliminar utilizando criteria
- ❑ Llamar a funciones definidas por el usuario
- ❑ Sincronización de persistencia

WebSocket Java API

JSR 356

- API para crear aplicaciones WebSocket
 - ▣ TCP, bi-direccional, mensajes full duplex
 - ▣ “Handshake”
 - ▣ Transferencia de datos

- Flujo
 - ▣ Establecer la conexión (TCP)
 - ▣ Envío de mensajes en ambas direcciones (Bi-directional)
 - ▣ Envío de mensajes independientemente (Full Duplex)
 - ▣ Conexión finalizada

WebSocket Java API

WebSocket Handshake

Handshake request

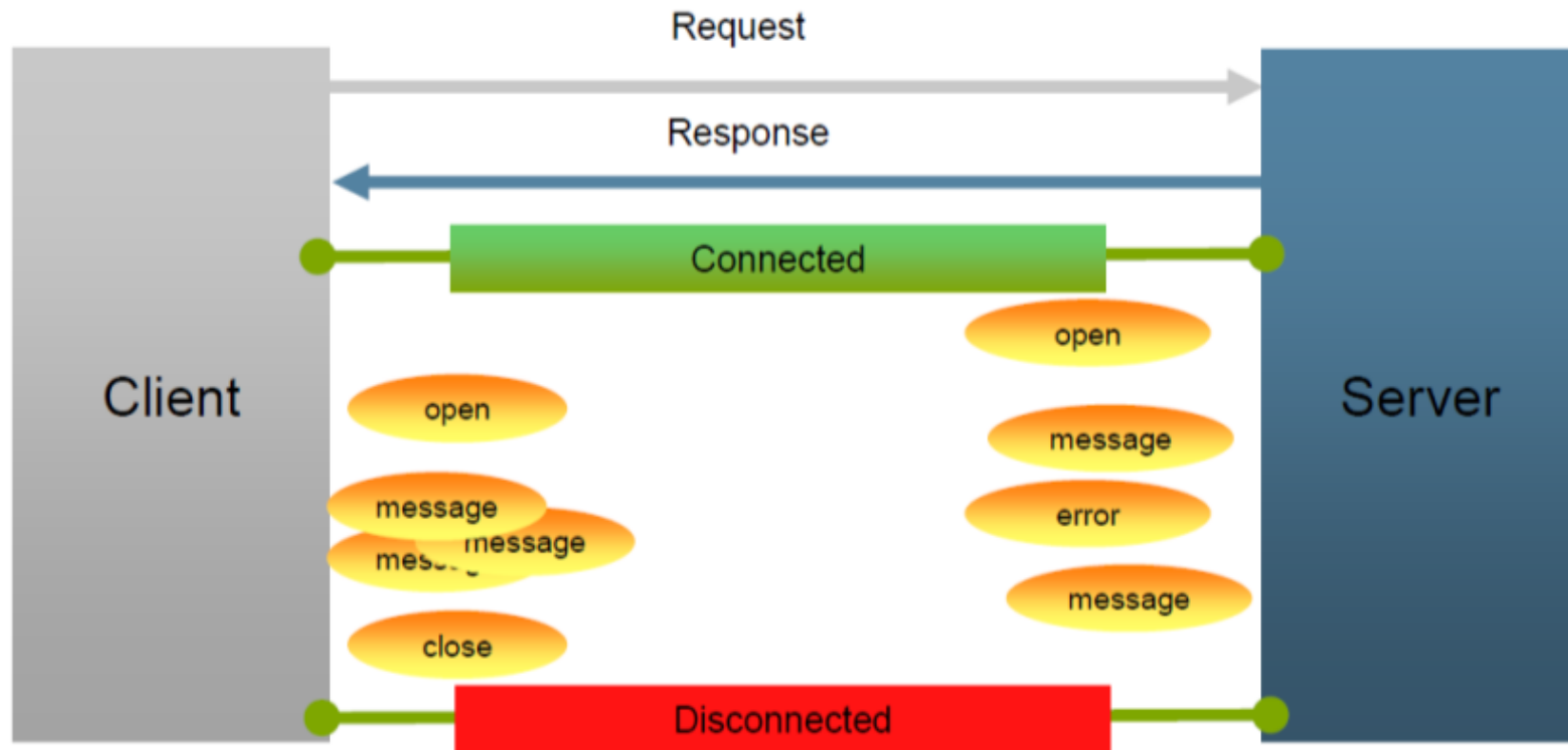
```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key:
dGhlIHhnbXBsZSBub25jZQ==
Origin: http://example.com
Sec-WebSocket-Protocol: chat,
superchat
Sec-WebSocket-Version: 13
```

Handshake response

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept:
s3pPLMBiTxaQ9kYGzzhZRbK+xOo
=
Sec-WebSocket-Protocol: chat
```

WebSocket Java API

WebSocket Handshake



WebSocket Java API

WebSocket Handshake

■ = Supported
 ■ = Not supported
 ■ = Partially supported
 ■ = Support unknown

Web Sockets - Working Draft

Bidirectional communication technology for web apps

Resources: [WebSockets information](#) [Details on newer protocol](#) [Wikipedia](#)

Global user stats*:

	IE	Firefox	Chrome	Safari	Opera	iOS Safari	Opera Mini	Android Browser	BlackBerry Browser	Opera Mobile	Chrome for Android	Firefox for Android
18 versions back			4.0									
17 versions back			5.0									
16 versions back			6.0									
15 versions back		2.0	7.0									
14 versions back		3.0	8.0									
13 versions back		3.5	9.0									
12 versions back		3.6	10.0									
11 versions back		4.0	11.0									
10 versions back		5.0	12.0									
9 versions back		6.0	13.0		9.0							
8 versions back		7.0	14.0		9.5-9.6							
7 versions back		8.0	15.0		10.0-10.1							
6 versions back		9.0	16.0		10.5							
5 versions back		10.0	17.0	3.1	10.6			2.1				
4 versions back	5.5	11.0	18.0	3.2	11.0	3.2		2.2		10.0		
3 versions back	6.0	12.0	19.0	4.0	11.1	4.0-4.1		2.3		11.0		
2 versions back	7.0	13.0	20.0	5.0	11.5	4.2-4.3		3.0		11.1		
Previous version	8.0	14.0	21.0	5.1	11.6	5.0-5.1		4.0		11.5		
Current	9.0	15.0	22.0	6.0	12.0	6.0	5.0-7.0	4.1	7.0	12.0	18.0	15.0
Near future	10.0	16.0	23.0		12.1				10.0			
Farther future		17.0	24.0		12.5							

WebSocket Java API

JSR 356

- WebSocket Endpoints
 - ▣ Annotation (@WebSocketEndpoint)
 - ▣ Interface (Endpoint)
- Integración con el contenedor Java EE Web.
- Proporciona APIs para el lado cliente y servidor.

Batch Applications Platform 1.0

JSR 352

- “Job”
 - ▣ Uno o más “Steps”
- “Step”
 - ▣ Phase secuencial de un batch job
 - ▣ Read, Process y Write
 - ▣ Se ejecuta en un determinado orden.
- Permite puntos de control
- Control del Job

Batch Applications Platform 1.0

JSR 352

```
<step id="sendStatements">  
  <chunk reader="AccountReader"  
    writer="EmailWriter"  
    processor="AccountProcessor"  
    chunk-size="10" />  
</step>
```

@ReadItem

```
public Account readAccount() {  
    // read account using JPA  
}
```

@ProcessItem

```
public Account processAccount(Account account) {  
    // calculate balance  
}
```

@WriteItems

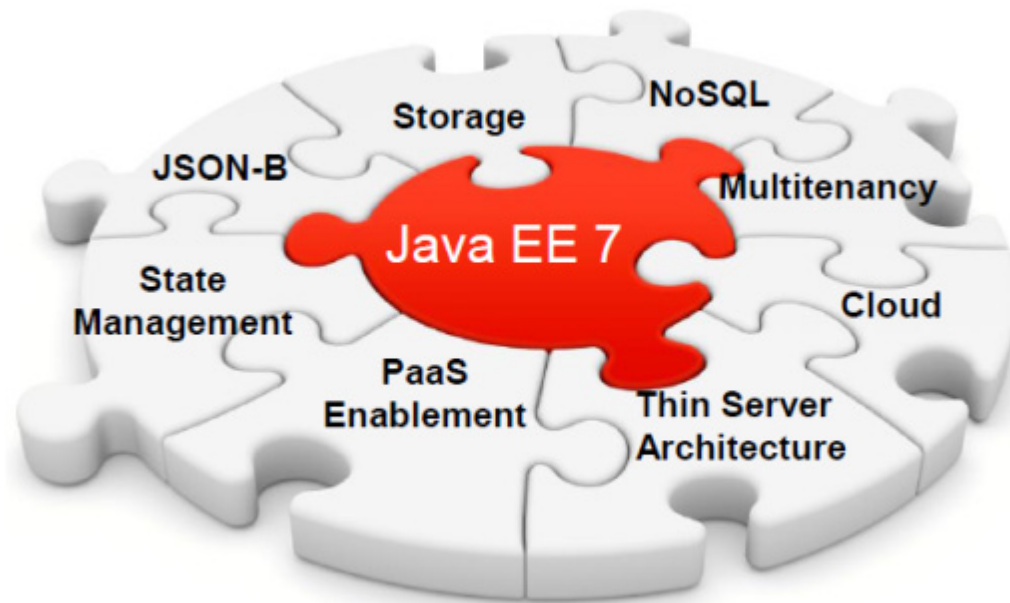
```
public void sendEmail(List<Account> accounts) {  
    // use JavaMail to send email  
}
```

Más de JEE 7

- ❑ JCache 1.0: API para cache temporalmente objetos Java
- ❑ JPA 2.1: Generación de Schema, procedimientos, ...
- ❑ EJB 3.2: Opcional CMP/BMP, fácil uso, ...
- ❑ JSF 2.2: Faces Flows, soporte HTML5,...
- ❑ Servlet 3.1: soporte WebSocket,...
- ❑ EL 3.0: Lambda expressions, Collection, Operators,...
- ❑ JTA 1.2: Interceptores transaccionales,...
- ❑ CDI 1.1: Ordenamiento de interceptors, Servlet events, ...

Java EE 8

- Arquitectura en la nube
- Aplicaciones SAS
- Entrega incremental de los JSRs
- Modularidad basada en Jigsaw



Referencias

- Java Community Process
 - ▣ <http://jcp.org/en/home/index>
- Java EE 7 transparent Expert Group
 - ▣ javaee-spec.java.net
- Java EE & GlassFish @ JavaOne 2012
 - ▣ glassfish.org/javaone2012
- Java EE 7 Reference Implementation
 - ▣ glassfish.org



Gracias

alvaro.maza@avantica.net