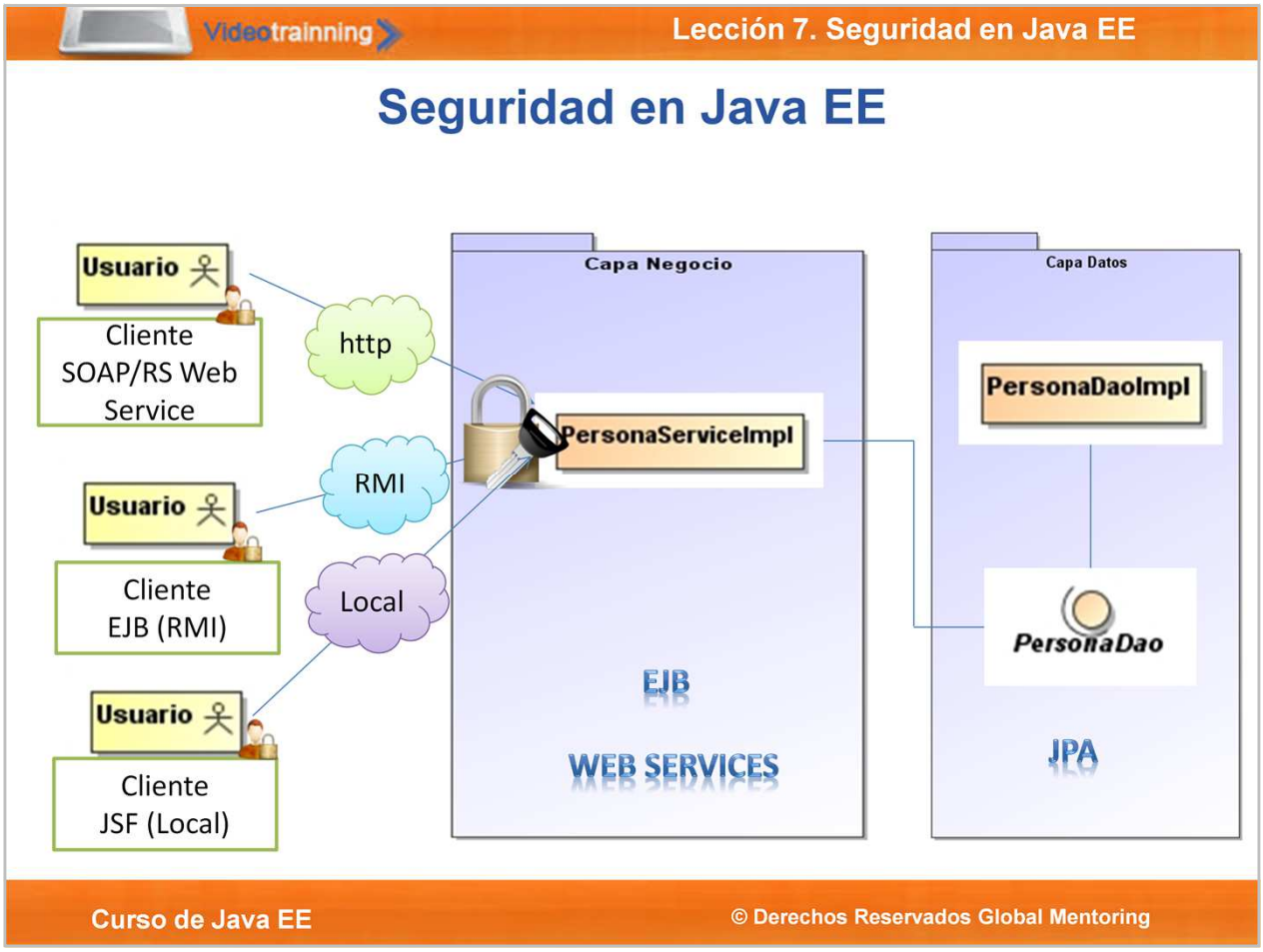


Uno de los temas cruciales al momento de desarrollar aplicaciones empresariales Java es el tema de seguridad.

Ya sea que nuestra aplicación sea consultada a través de una interface Web utilizando Servlets o JSF, que expongamos la funcionalidad utilizando Web Services SOAP o REST, o que directamente un Cliente Swing tenga acceso a nuestros EJB, es necesario establecer los mecanismos que garanticen la seguridad de la información del sistema.

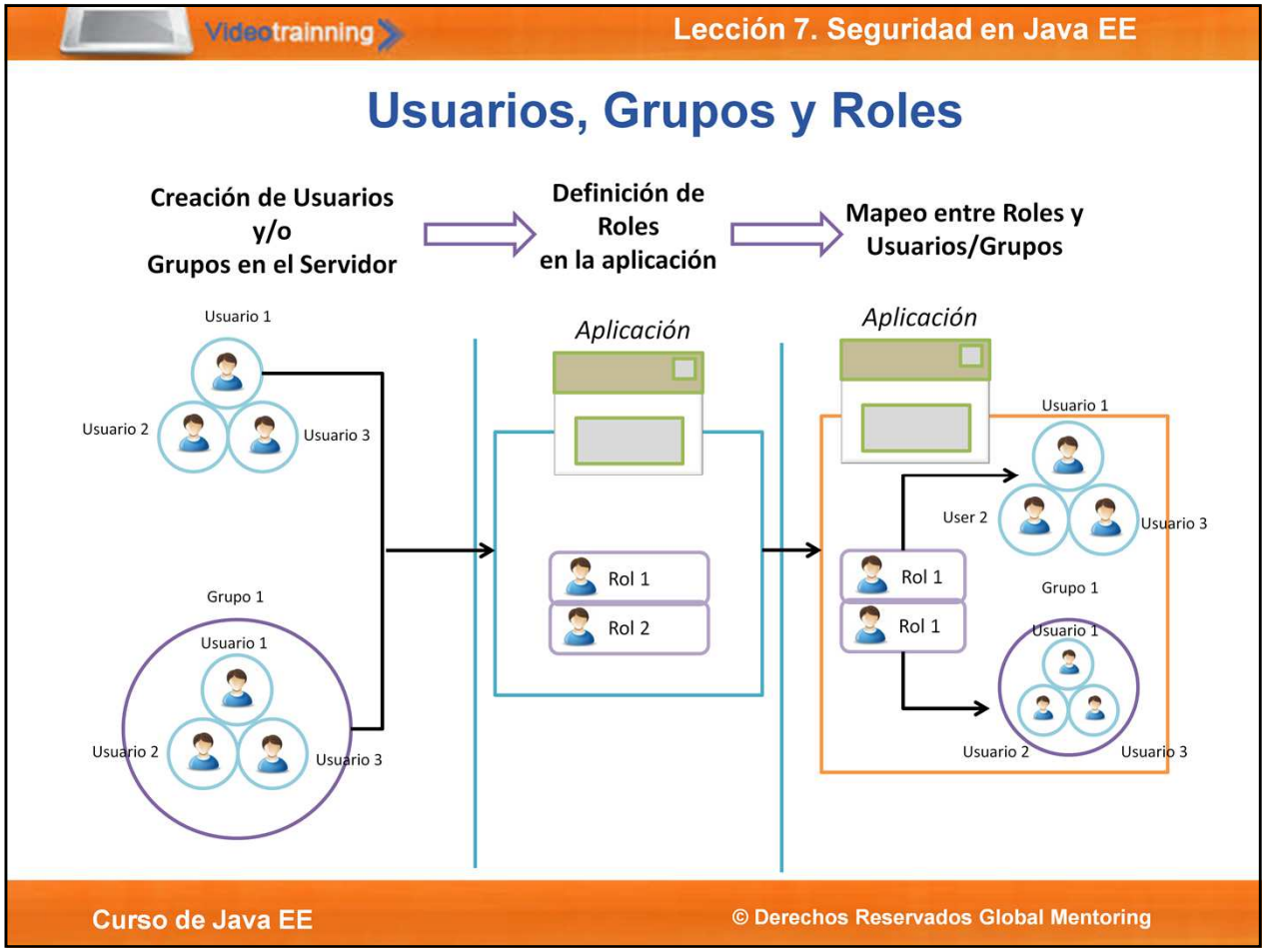
La seguridad se debe asegurar desde el punto de vista de la vista y la capa de negocio, debido a que un usuario que no pueda visualizar una página, no significa que no pueda ejecutar un método de la capa de negocio.

Uno de los beneficios de utilizar EJB, es la simplicidad y flexibilidad para agregar seguridad en la ejecución de sus métodos. Similar al manejo de transacciones, el manejo de Seguridad contamos con la Seguridad Declarativa y la Seguridad Programática. Revisaremos cada una de ellas en las siguientes láminas.

Dos conceptos son básicos respecto a la seguridad de un sistema. Los conceptos de Autenticación y Autorización:

Autenticación: La autenticación es el proceso de verificar la identidad del usuario. Lo más común es identificarnos ante el sistema por medio de un usuario y contraseña. A estos datos se les conoce como credenciales. Formas más avanzadas de autenticación es por medio de autenticación biométrica, tales como: huellas dactilares, patrones de Iris y retina, detección de rasgos físicos como el rostro, etc. La autenticación se debe ejecutar antes de la autorización.

Autorización: Una vez que el usuario se ha autenticado en el sistema, el siguiente paso es verificar los permisos del usuario a ejecutar cierta funcionalidad del sistema. De tal manera, que sin importar que un usuario se haya autenticado correctamente, no necesariamente tendrá la autorización para ejecutar la funcionalidad deseada en el sistema. Estos dos conceptos está ligados al manejo de usuarios, grupos y roles, los cuales revisaremos a continuación.



Los Usuarios, Grupos y Roles son 3 conceptos inter relacionados que conforman la base del esquema de seguridad en un sistema empresarial.

Para simplificar la administración los usuarios son divididos por grupos. Los grupos son una partición lógica para la identificación de usuarios, los cuales pueden tener acceso a distintos recursos. Por ejemplo, una aplicación puede tener un grupo de "Administradores" compuesto de usuarios con permisos de eliminación de registros.

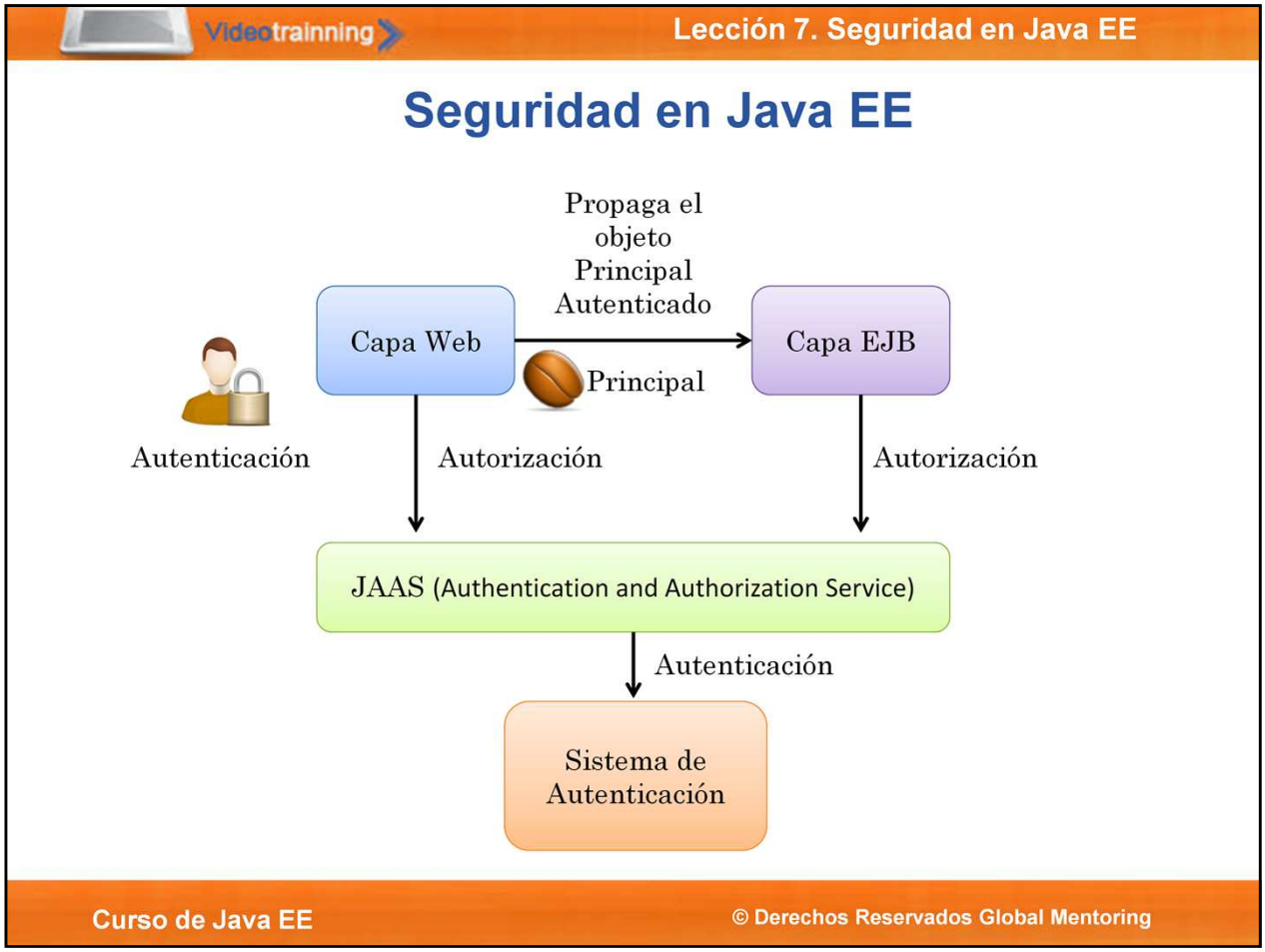
Antes de ejecutarse cualquier operación en el sistema, la aplicación revisa si el usuario es un miembro del grupo y permite o deniega la ejecución del método/recurso solicitado. El manejo de grupos permite asignar las funciones comunes de los usuarios a una sola entidad, y de esta manera facilitar la administración de cada individuo.

Un rol, es un concepto muy parecido a un grupo. Sin embargo, como se observa en la figura, es un concepto más ligado a la aplicación que se está desarrollando. En cambio, un grupo se define a nivel organizacional, ejemplo, en un Directorio Activo de Microsoft o un LDAP (Lightweight Directory Access Protocol). Un rol permite relacionar los grupos de una organización en entidades que entienda y procese correctamente la aplicación.

De esta manera, los roles son una abstracción de los grupos enfocados en el servidor de aplicaciones Java. Un mapeo entre los roles y los grupos es comúnmente realizado. Si los nombres de los grupos y los roles son equivalentes, la mayoría de los servidores de aplicaciones pueden realizar el mapeo de manera automática, de lo contrario se debe configurar el servidor para realizar esta tarea.

En caso que los nombres de los grupos y roles no coincidan, se debe especificar el mapeo de roles-grupos al servidor Java. Esto se puede hacer por medio de configuración del mismo servidor o por medio de archivos de configuración. Por ejemplo, en el caso de GlassFish se debe agregar el archivo glassfish-web.xml. Para más información del mapeo de grupos y roles pueden consultar el siguiente link:

http://docs.oracle.com/cd/E18930_01/html/821-2417/beaql.html
http://docs.oracle.com/cd/E18930_01/html/821-2418/beabg.html#scrolltoc



La seguridad en Java EE está basada ampliamente en el API de JAAS (Authentication and Authorization Service). Como su nombre lo indica, esta API es responsable del proceso de autenticación de los usuarios y autorización de recursos de los sistemas Java, sobre todo enfocado en agregar seguridad en la capa Web y la capa de negocio donde se encuentran los EJB.

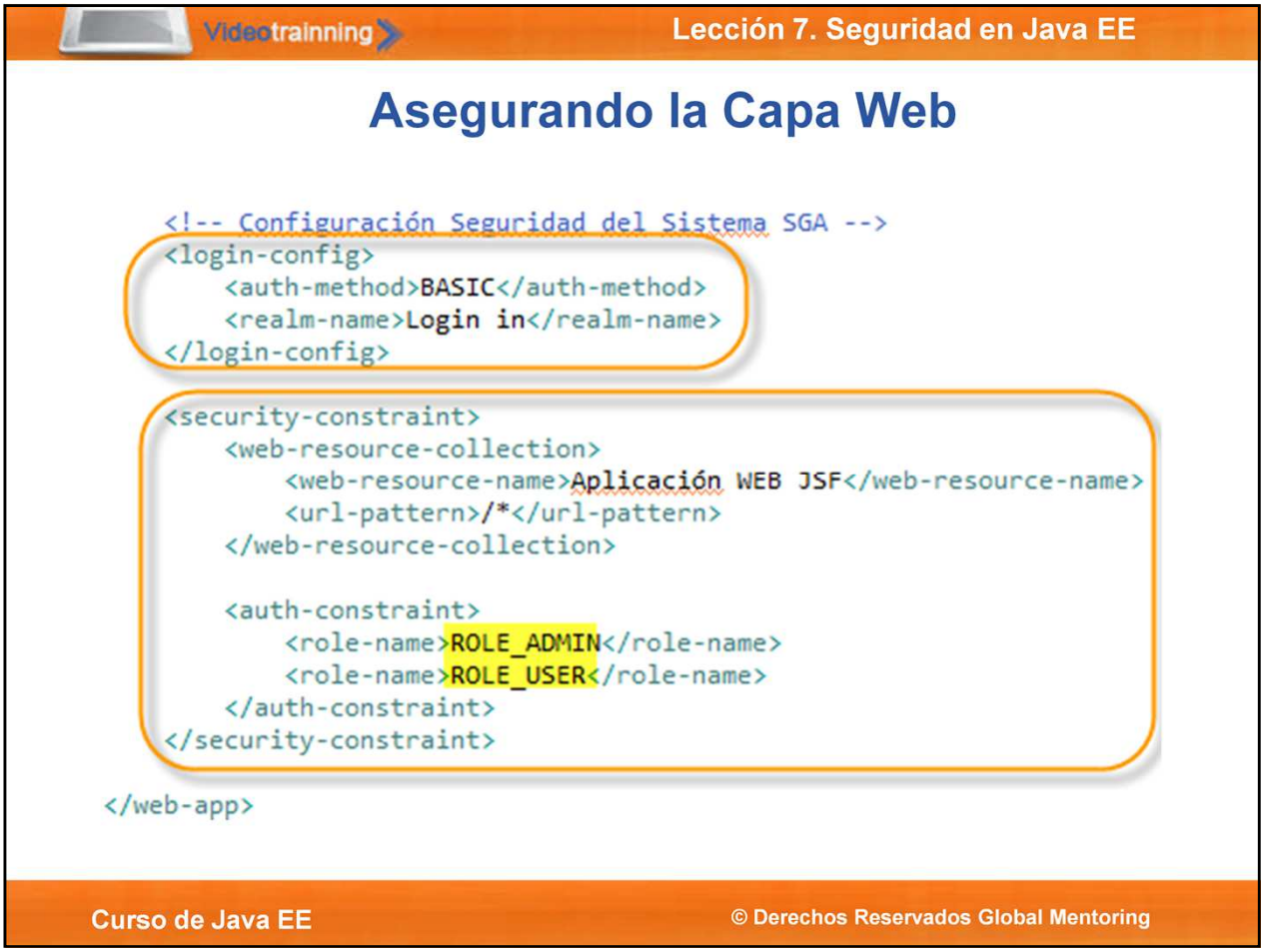
Una vez que el usuario del sistema es autenticado, el contexto de la autenticación es propagado a través de las distintas capas de una aplicación empresarial, siempre que sea posible. Esto se realiza para que evitar el proceso de autenticación por cada una de las capas.

Una vez que el usuario se ha autenticado, el API de seguridad JAAS crea un objeto conocido como ***Principal***. Este objeto es propagado entre las capas con el objetivo de ya no solicitar nuevamente la autenticación del usuario.

Como se observa en la figura, un usuario se puede autenticar por medio de una aplicación Web, a su vez la capa Web recupera la información y autentica al usuario, creando el objeto ***Principal*** en caso de una autenticación exitosa.

El objeto ***Principal*** está asociado con uno o más roles. El sistema de seguridad revisa por cada acción, ya sea en la capa Web o en la capa de EJBs, que se tengan los permisos para ejecutar dicho recurso.

El objeto ***Principal*** es enviado de manera transparente entre las capas Web y EJB según sea necesario.



La seguridad en la capa Web se realizar a través del archivo descriptor ***web.xml.*** Existen distintos tipos de autenticación los cuales son HTTP Basic, HTTP DIGEST, HTTPS Client-Cert y FORM based. El estudio a detalle de cada una de estas configuración está fuera del alcance de este curso, sin embargo en los ejercicios utilizaremos los métodos más comunes, con el objetivo de familiarizarse con esta configuración.

Según observamos en el código de la figura, el primer elemento que agregamos es <login-config>, el cual nos permite especificar la forma en que el contenedor Web recuperará la información para poder autenticar a los usuarios.

Posteriormente tenemos el elemento <security-constraint>, el cual nos permitirá especificar los recursos (URL) a las cuales agregaremos seguridad, así como los Roles que participarán en la revisión de la seguridad. Es posible especificar incluso el tipo de Método HTTP que se permitirá (ej. GET, POST, PUT, DELETE, etc).

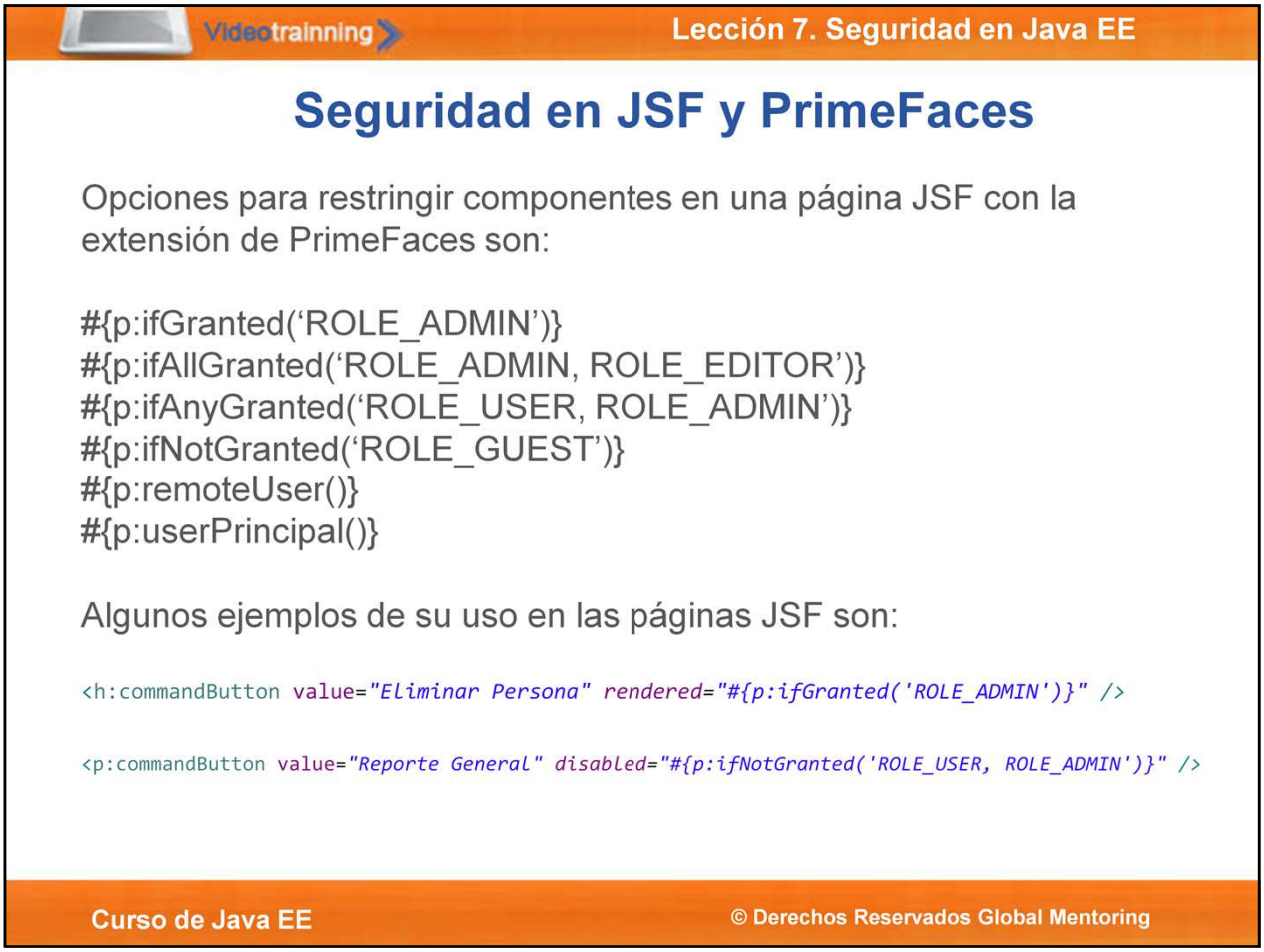
El tipo de autenticación mostrado en la figura es de tipo BASIC, el cual el navegador Web mostrará un pop-up genérico solicitando el usuario y password para poder autenticarse al sistema. Otro método muy comúnmente utilizado es el método basado en una forma HTML (FORM based). Esta configuración tiene la ventaja de que es posible personalizar la página de autenticación que se muestra al usuario.

Posteriormente el real-name. Un realm es una abstracción del servidor de aplicaciones, donde se especifican las políticas de seguridad del sistema. Un realm contiene una colección de usuarios, los cuales pueden o no ser asociados a un grupo. Un realm se configurar utilizando las herramientas de administración del servidor de aplicaciones en cuestión. En nuestro ejercicio utilizaremos un real basado en un archivo (file), sin embargo existen realm basado en JDBC, LDAP, y podemos agregar realms personalizados, ya que en ocasiones utilizaremos sistemas externos para realizar el proceso de autenticación.

El elemento <security-constraint> nos permite especificar una o más colecciones de URL que deseamos agregar seguridad. Por medio de un url-pattern, especificamos las páginas a asegurar. Finalmente con el elemento <auth-constraint> especificamos los roles permitidos para ejecutar el recurso especificado.

De esta manera hemos agregado tanto el concepto de Autenticación como el concepto de Autorización para una aplicación Web. El tema de cómo asegurar un EJB lo estudiaremos más adelante. Para un estudio más detallado de los tipos de autenticación pueden consultar el siguiente link:

http://docs.oracle.com/javaee/6/tutorial/doc/gijrp.html



Al autenticarnos correctamente por medio de la aplicación Web, JSF y en particular la extensión de PrimeFaces, ha agregado varios elementos que nos permiten agregar muy fácilmente seguridad a nuestras páginas JSF.

De tal manera que podremos habilitar o deshabilitar funcionalidad, incluso a nivel de componentes, botones, links, tablas, etc, dependiendo del rol del usuario que se ha autenticado.

Para más información de esta funcionalidad pueden consultar el siguiente link:

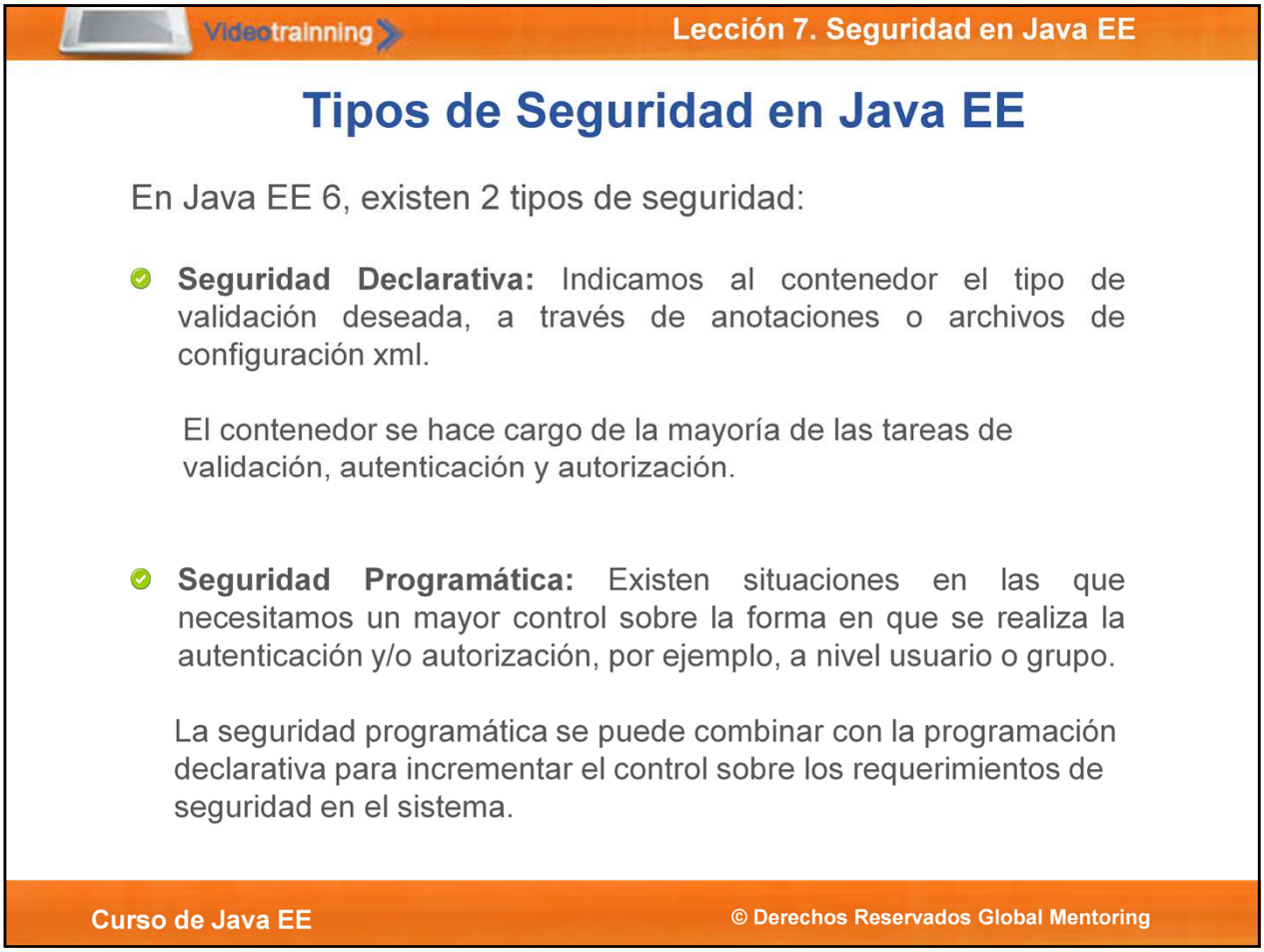
http://cagataycivici.wordpress.com/2010/03/18/primefaces-el-extensions-for-ui-authorization/

Las opciones que tenemos disponibles con la extensión de PrimeFaces son las siguientes:

#{p:ifGranted('ROLE_ADMIN')}
#{p:ifAllGranted('ROLE_ADMIN, ROLE_EDITOR')}
#{p:ifAnyGranted('ROLE_USER, ROLE_ADMIN')}
#{p:ifNotGranted('ROLE_GUEST')}
#{p:remoteUser()}
#{p:userPrincipal()}

Algunos ejemplos de su uso son:

```
<h:commandButton value="Eliminar Persona" rendered="#{p:ifGranted('ROLE_ADMIN')}" />
<p:commandButton value="Reporte General" disabled="#{p:ifNotGranted('ROLE_USER, ROLE_ADMIN')}" />
```



Similar al manejo de transacciones, en el tema de seguridad también es posible manejar la Seguridad Declarativa y la Seguridad Programática.

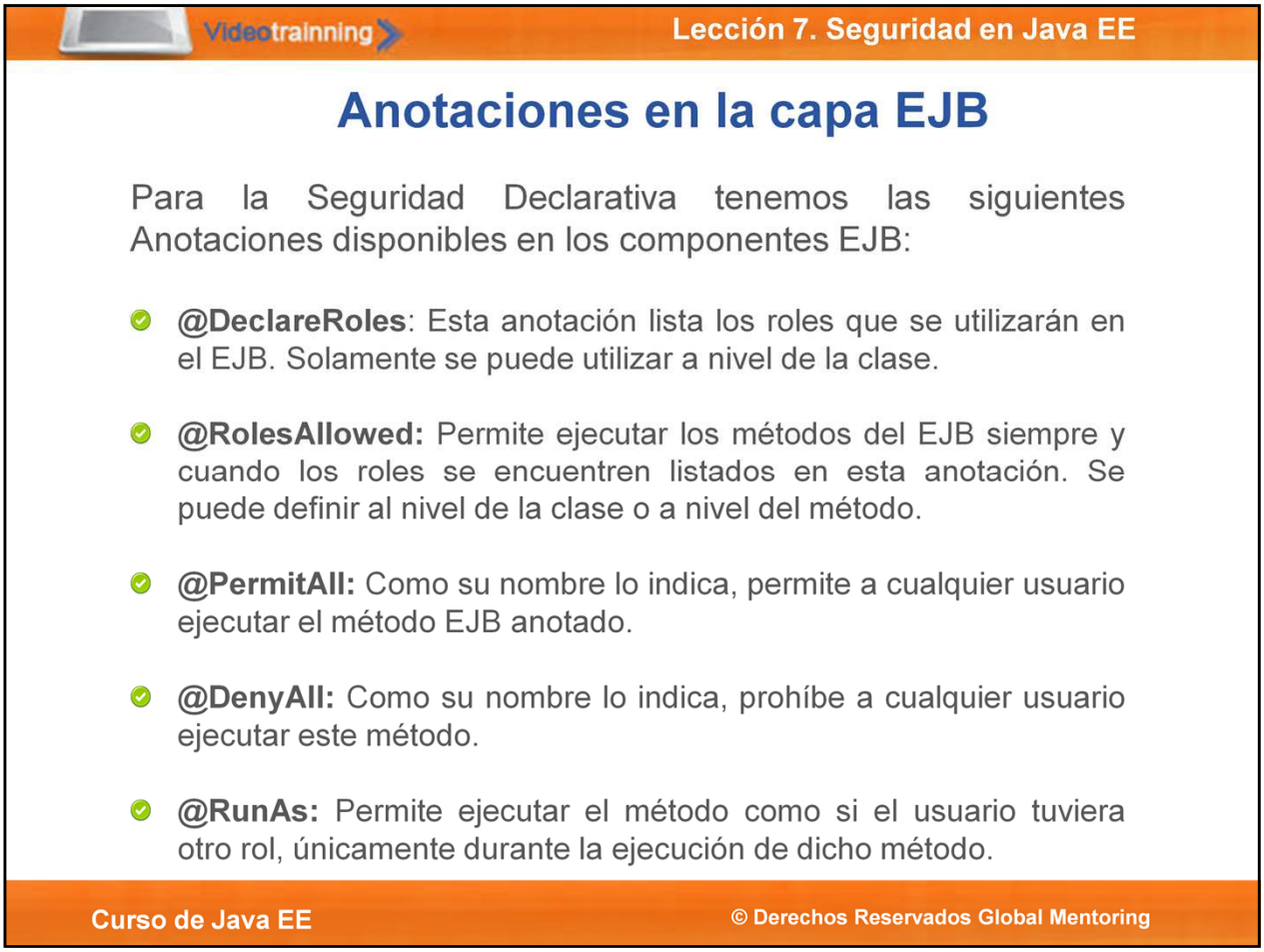
- **Seguridad Declarativa:** Indicamos al contenedor el tipo de validación deseada, a través de anotaciones o archivos de configuración xml. El contenedor se hace cargo de la mayoría de las tareas de validación, autenticación y autorización.

En el caso de la seguridad declarativa, es posible aplicarla a nivel de la clase o a nivel de cada método. El contenedor verifica el rol asignado del usuario autenticado previamente, y si cuenta con el rol solicitado, entonces permite ejecutar el método del EJB respectivo.

Para manejar la Seguridad Declarativa es muy común que utilicemos anotaciones, las cuales estudiaremos en la siguiente lámina.

- **Seguridad Programática:** Existen situaciones en las que necesitamos un mayor control sobre la forma en que se realiza la autenticación y/o autorización, por ejemplo, a nivel usuario o grupo. La seguridad programática se puede combinar con la programación declarativa para incrementar el control sobre los requerimientos de seguridad en el sistema.

La seguridad programática nos sirve cuando tenemos requerimientos más detallados, y que serían complicados o imposibles de lograr con la Seguridad Declarativa. La buena noticia es que podemos combinar la Seguridad Declarativa en conjunto con la Seguridad Programática, de tal manera que en los casos que necesitemos realizar validaciones más detalladas, incluso por cada usuario, podremos combinar el poder de ambas opciones de seguridad.



Para la Seguridad Declarativa tenemos las siguientes Anotaciones disponibles en los componentes EJB:

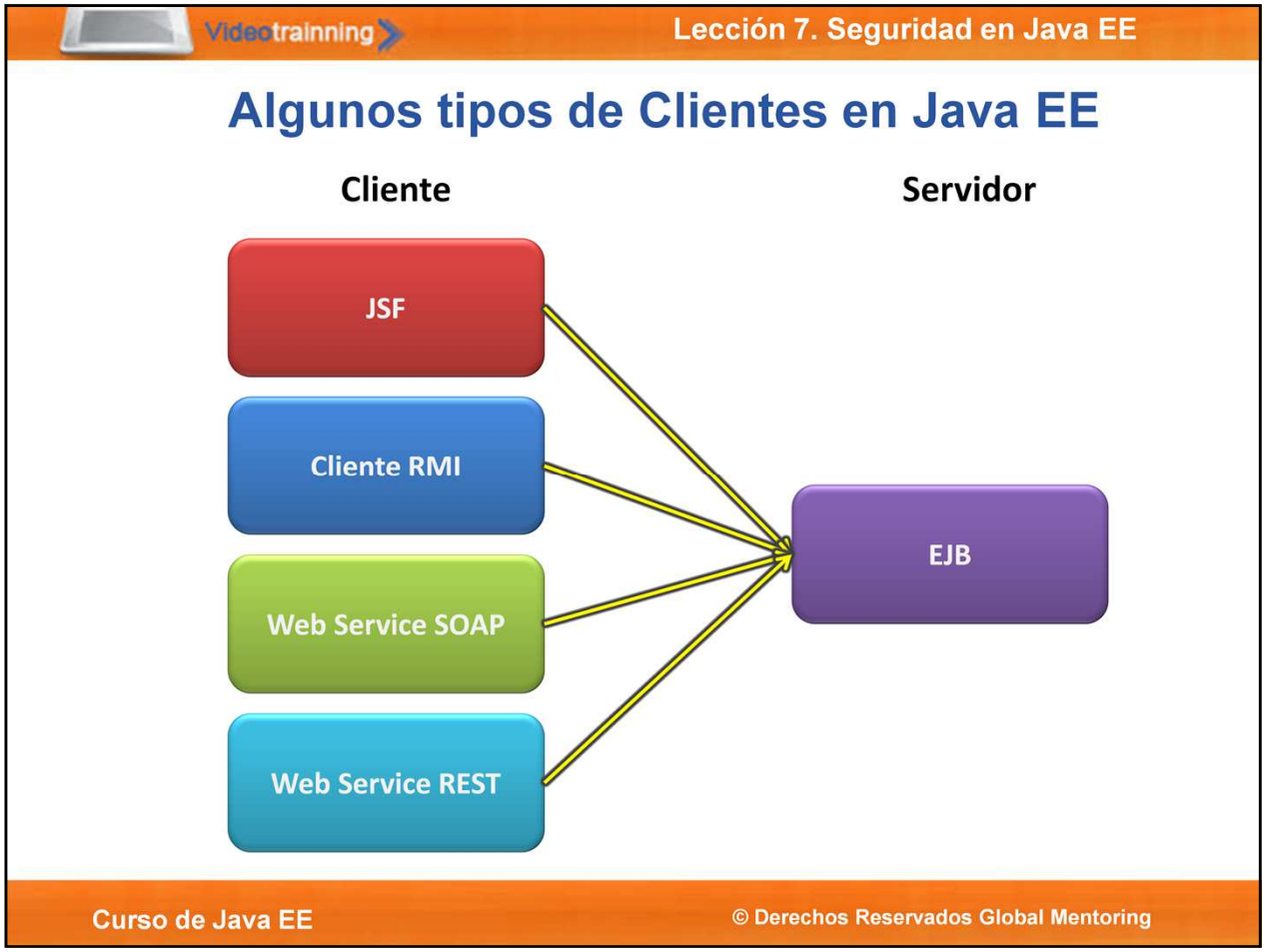
- **@DeclareRoles**: Esta anotación lista los roles que se utilizarán en el EJB. Solamente se puede utilizar a nivel de la clase. Si esta anotación no se proporciona, el contenedor buscará en los roles definidos por la anotación @RolesAllowed y construye una lista de roles. Se recomienda agregarla debido a que en algunos contenedores, al no agregarla, debemos agregar configuración extra para que detecte los roles que estamos manejando. Si el EJB extiende de otra clase, la lista de roles se concatena, es decir, también se incluyen.

- **@RolesAllowed:** Permite ejecutar los métodos del EJB siempre y cuando los roles se encuentren listados en esta anotación. Se puede definir al nivel de la clase o a nivel del método. Cuando se define al nivel de la clase, estamos indicando al contenedor que los roles listados podrán ejecutar cualquier método del EJB. Si agregamos esta anotación a nivel del método, sobreescribimos cualquier comportamiento definido a nivel de la clase. Sin embargo debemos tener precaución al combinar los roles a nivel clase y nivel método para evitar que la mezcla o sobreescritura de roles se convierta en un problema.

- **@PermitAll:** Como su nombre lo indica, permite a cualquier usuario ejecutar el método EJB anotado. Se debe utilizar esta anotación con precaución, ya que puede ser un hueco de seguridad si esta mal utilizado.

- **@DenyAll:** Como su nombre lo indica, prohíbe a cualquier usuario ejecutar este método. Esta anotación provoca que el método quede inusable, sin embargo para deshabilitar cierta funcionalidad del sistema puede ser una posible solución.

- **@RunAs:** Permite ejecutar el método como si el usuario tuviera otro rol, únicamente durante la ejecución de dicho método. Esto permite a un usuario con rol con menos privilegios, asignarle más privilegios, debido al cambio de rol, únicamente por la ejecutación del método en cuestión.



Cuando hablamos de una aplicación empresarial, podemos tener distintos clientes interesados en obtener información de nuestro sistema. Esta información es comúnmente expuesta a través de la capa de negocio o EJB.

Como podemos observar en la figura, tenemos podemos tener cliente Web, RMI, SOAP Web Service, REST Web Service, entre otros.

Desafortunadamente los mecanismos de autenticación entre los distintos clientes al día de hoy no son estándar, y muchas de las veces dependemos todavía de la implementación del servidor Java que estemos utilizando para poder comunicar cada cliente y poder autenticarnos de manera exitosa con el servidor Java.

En el ejercicio que realizaremos en esta lección incluiremos la autenticación de cada uno de los clientes mostrados en la figura, ya que cada uno tiene diferentes mecanismos, incluso en algunos casos es necesario utilizar clases que nos proporciona el servidor de aplicaciones que estemos utilizando, por lo que la solución puede depender del servidor de aplicación en cuestión.

El caso donde hay mayores pasos de configuración es el Cliente Web, ya que a su vez, según hemos visto, funciona como un punto de autenticación del sistema, por lo tanto la propagación del objeto Principal entre la capa Web y la capa EJB funciona de manera automática al utilizar la seguridad en Java EE.

Como conclusión, podemos observar la manera en que las diferentes tecnologías y tipos de clientes se ven obligados a identificarse ante el sistema antes de poder acceder a cualquier recurso que se haya agregado seguridad. Esto permite crear aplicaciones empresariales Robustas, Flexibles, Mantenibles y por supuesto Seguras.

Ejercicio 14

- Abrir el documento PDF de Ejercicios del cursos de Java EE.
- Realizar las siguientes prácticas:
- **Ejercicio 13**: Creación Proyecto SGA con Seguridad

Referencias

- La referencia del tema de Seguridad en Java EE es la siguiente:
- http://docs.oracle.com/javaee/6/tutorial/doc/bnbxj.html
- La referencia del tema de Seguridad para el servidor de GlassFish la pueden consultar en el siguiente enlace:
- http://docs.oracle.com/cd/E26576_01/doc.312/e24940/toc.htm
- Para la creación de archivos específicos para el servidor de glassfish:

Archivo glassfish-web.xml
- http://docs.oracle.com/cd/E18930_01/html/821-2417/beaql.html

Archivo glassfish-ejb-jar.xml
- http://docs.oracle.com/cd/E18930_01/html/821-2417/beaqm.html

En Global Mentoring promovemos la *Pasión por la Tecnología Java*.

Te invitamos a visitar nuestro sitio Web donde encontrarás cursos Java Online desde Niveles Básicos, Intermedios y Avanzados.

Además agregamos nuevos cursos para que continúes con tu preparación como consultor Java de manera profesional.

A continuación te presentamos nuestro listado de cursos en constante crecimiento:

- Fundamentos de Java
- Programación con Java
- Java con JDBC
- HTML, CSS y JavaScript
- Servlets y JSP's
- Struts Framework
- Hibernate Framework
- Spring Framework
- JavaServer Faces
- Java EE (EJB, JPA y Web Services)
- JBoss Administration

Datos de Contacto:

Sitio Web: www.globalmentoring.com.mx

Email: informes@globalmentoring.com.mx

Ayuda en Vivo: www.globalmentoring.com.mx/chat.html