Clase grabada

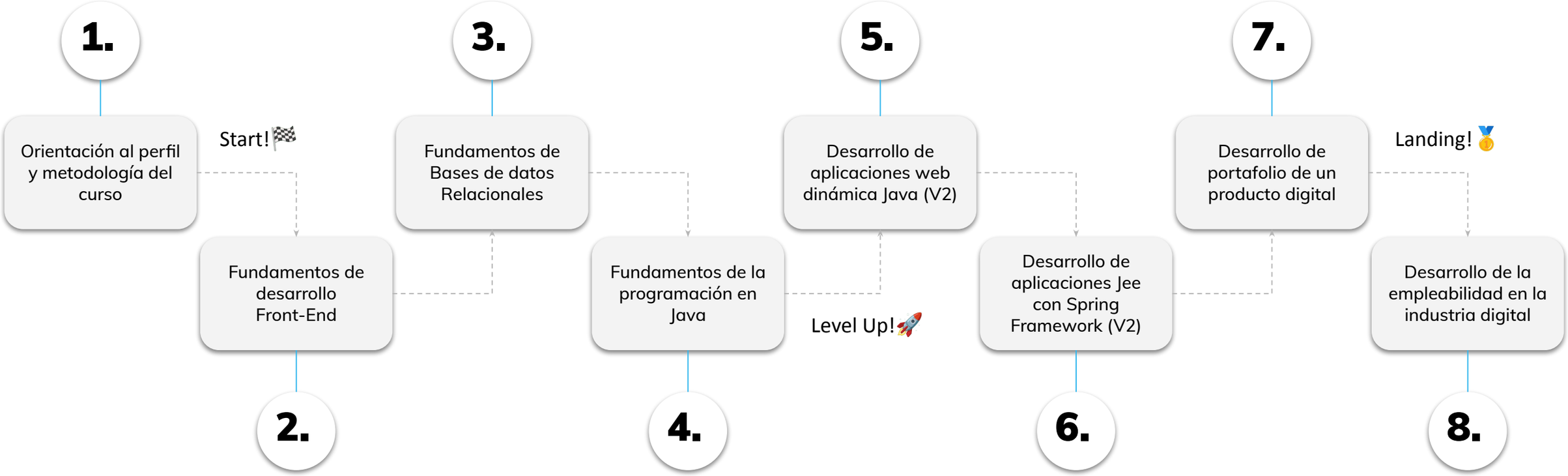
El Framework Spring MVC

Plan formativo: Desarrollo de Aplicaciones Full Stack Java Trainee V2.0

English Always | alkemy

HOJA DE RUTA

¿Cuáles **skill** conforman el programa?



English Always | alkemy

REPASO CLASE ANTERIOR

En la clase anterior trabajamos 📚:

- ✔ *Capa de vistas y controladores*
- ✔ *Configuración de las peticiones*
- ✔ *Controladores multiacción*

English Always | alkemy

LEARNING PATHWAY

¿Sobre qué temas trabajaremos?

6.2

Start! 🏁

El Framework Spring MVC

Envío de datos Vista-Controlador

Renderizando

Recibiendo datos en el controlador
Entregando datos a la vista
Desplegando una vista JSP con datos entregados desde el controlador

English Always | alkemy

OBJETIVOS DE APRENDIZAJE

¿Qué aprenderemos?

- *Aprender a recibir datos en el controlador y a enviar datos a la vista*
- *Realizar el despliegue de una vista JSP con datos entregados en el Controlador*

Recibiendo datos en el controlador

Recibiendo datos

Una de las tareas principales de un controlador Spring es recibir datos provenientes del cliente, generalmente a través de una solicitud HTTP. Esto puede incluir parámetros, rutas, encabezados u otros datos adjuntos a la solicitud. Spring proporciona varias formas de recibir estos datos en el controlador.

Recibiendo datos

1- **Parámetros de Solicitud**: Los controladores Spring pueden recibir parámetros de solicitud que se envían en la URL o en el cuerpo de una solicitud HTTP. Esto se logra mediante la anotación @RequestParam. Por ejemplo, supongamos que tienes una solicitud HTTP que envía un parámetro llamado "nombre":

```
@GetMapping("/saludo")
public String saludar(@RequestParam("nombre") String nombre) {
    // Procesar el parámetro "nombre"
    return "saludo";
}
```

Recibiendo datos

2- **Variables de Ruta**: Los controladores también pueden recibir datos a través de variables de ruta, las cuales se identifican con la anotación @PathVariable. Esto es útil para extraer valores de la URL de la solicitud. Por ejemplo, si tienes una URL como "/usuario/123", puedes recibir el valor "123" como una variable de ruta:

```
@GetMapping("/usuario/{id}")
public String verUsuario(@PathVariable("id") Long id) {
    // Procesar el ID del usuario
    return "perfil";
}
```

Recibiendo datos

3- **Objetos de Solicitud**: Spring permite recibir un objeto completo de solicitud, que incluye todos los datos de la solicitud, como parámetros, encabezados y más. Esto se hace utilizando la anotación @ModelAttribute. Por ejemplo:

```
@PostMapping("/nuevo-usuario")
public String crearUsuario(@ModelAttribute Usuario usuario) {
    // Procesar el objeto de solicitud (Usuario)
    return "confirmacion";
}
```

Recibiendo datos

4- **Objeto HttpServletRequest:** proporciona acceso a todos los detalles de una solicitud HTTP, incluyendo parámetros, encabezados, cuerpo de la solicitud y otros atributos relacionados con la solicitud.

```java
@GetMapping("/ejemplo")
public String recibirDatos(HttpServletRequest request) {
    // Recibir datos de la solicitud utilizando HttpServletRequest
    String parametroNombre = request.getParameter("nombre");
    String parametroEdad = request.getParameter("edad");

    // Procesar los datos
    // ...

    return "vista";
}
```

Recibiendo datos

Sin embargo, es importante destacar que el uso de HttpServletRequest a menudo implica una codificación más detallada y manual en comparación con el uso de anotaciones como @RequestParam o @PathVariable, que simplifican la extracción de datos de la solicitud. Por lo tanto, se recomienda utilizar HttpServletRequest **cuando sea necesario**, **pero considera usar anotaciones cuando sea posible** para una implementación más concisa y mantenible.

Evaluación Integradora ✨

¿Listos para un nuevo desafío? En esta clase comenzamos a construir nuestro....

Trabajo Integrador del Módulo 💪🏽

Iremos completándolo progresivamente clase a clase.

English Always | alkemy

LIVE CODING

Ejemplo en vivo

WalletController:

Vamos a continuar el diseño de la AlkeWallet. En este caso aplicaremos seguridad creando una contraseña vinculada a un usuario para el ingreso a la aplicación.

Además, vamos a crear un controlador que reciba los datos de acceso del usuario y nos devuelva un mensaje de bienvenida personalizado.

Tiempo: 15 minutos

LIVE CODING

Ejemplo en vivo

1. *Agregar al paquete de entidades la entidad Usuario*
2. *Agregar los atributos String nombreUsuario, String email, Integer id, String password. Por el momento no vincularemos otra cuenta.*
3. *Crear una clase UsuarioController en el paquete de controladores*
4. *Crear un método que reciba el email y la contraseña del usuario a través de @RequestParam y devuelva un saludo a través de un modelo*

Entregando datos a la vista

Entregando datos a la vista

Una vez que el controlador ha procesado los datos de la solicitud, el siguiente paso es entregar esos datos a la vista para que se muestren al usuario final.

En Spring, este proceso se realiza mediante la devolución de un modelo de vista, que contiene los datos necesarios para la vista. Spring ofrece varias formas de hacerlo:

Entregando datos a la vista

1- **ModelAndView**:. Ya hemos visto ejemplos de cómo se utiliza en la explicación anterior. En este enfoque, el objeto ModelAndView se utiliza para combinar tanto la vista a la que se debe redirigir como el modelo de datos que se desea entregar a esa vista.

```
@GetMapping("/perfil")
public ModelAndView verPerfil() {
    ModelAndView modelAndView = new ModelAndView("perfil");
    modelAndView.addObject("nombre", "Juan");
    modelAndView.addObject("edad", 30);
    return modelAndView;
}
```

En este ejemplo, "perfil" es el nombre de la vista a la que se redirigirá, y se agregan atributos como "nombre" y "edad" al modelo de datos. Estos atributos estarán disponibles para su uso en la vista "perfil".

Entregando datos a la vista

2- **La interfaz Model** se utiliza para agregar atributos al modelo de datos en Spring. La principal ventaja de este enfoque es que no es necesario crear un objeto ModelAndView. Spring se encarga de seleccionar automáticamente la vista basándose en el nombre del controlador.

```
@GetMapping("/perfil")
public String verPerfil(Model model) {
    model.addAttribute("nombre", "Juan");
    model.addAttribute("edad", 30);
    return "perfil";
}
```

En este caso, simplemente devolvemos el nombre de la vista, "perfil". Spring sabe que debe buscar una vista llamada "perfil" para renderizar.

Entregando datos a la vista

3- **Map**: Aunque es menos común que Model o ModelAndView, es útil cuando se necesita una flexibilidad extrema en la construcción del modelo.

```
@GetMapping("/perfil")
public String verPerfil(Map<String, Object> model) {
    model.put("nombre", "Juan");
    model.put("edad", 30);
    return "perfil";
}
```

Este enfoque permite que cualquier objeto sea colocado en el modelo utilizando su nombre como clave en el mapa.

Entregando datos a la vista

4- . **JSON o XML (ResponseBody)**: Si necesitas entregar datos en formato JSON o XML, Spring facilita la conversión automática de objetos a estos formatos utilizando la anotación @ResponseBody. Esto es útil cuando se desarrollan servicios web RESTful y se necesita transmitir datos en lugar de vistas.

```
@GetMapping("/datos-json")
@ResponseBody
public Persona obtenerDatosEnJSON() {
    Persona persona = new Persona("Juan", 30);
    return persona;
}
```

En este ejemplo, la anotación @ResponseBody indica que el método del controlador debe devolver el objeto Persona como una respuesta en formato JSON. Spring se encarga de la serialización y envío de datos al cliente.

Entregando datos a la vista

5- **Uso de Thymeleaf y JSP**:. Estos motores de plantillas permiten que los datos entregados desde el controlador se integren de manera dinámica en el contenido HTML. Por ejemplo, en una plantilla Thymeleaf:

```
<p>Nombre: [[${nombre}]]</p>
<p>Edad: [[${edad}]]</p>
```

Aquí, los atributos "nombre" y "edad" se insertan en el HTML utilizando la sintaxis de Thymeleaf, lo que permite una presentación dinámica de los datos en la vista.

LIVE CODING

Ejemplo en vivo

WalletController:

A continuar el diseño de la AlkeWallet! En este caso vamos a crear un formulario de acceso a la aplicación en un archivo llamado login.jsp.

Tiempo: 15 minutos

LIVE CODING

Ejemplo en vivo

1. *Agregar la vista login.jsp al paquete templates*
2. *Agregar un formulario de acceso que pida ingresar usuario (email) y contraseña (password)*
3. *Crear una vista bienvenida.jsp que muestre por pantalla un saludo personalizado con el email del usuario recibido desde el controlador.*

Ejercicio

Renderizando

Controlando

Consigna: ✍️

Vamos a hacer un pequeño ejercicio de prueba para aplicar lo aprendido en clase

Crea una vista jsp que muestre por pantalla un saludo personalizado devuelto por el controlador realizado en la clase anterior!

En este nuevo paquete debes:

1- Crear una vista index.jsp que se renderice al ejecutarse el controlador que responde a la URL “/index”

2- Crear un saludo que muestre por pantalla el nombre enviado por el controlador en el String “nombre”

Tiempo🕐: *15 minutos*

¿Alguna consulta?

RESUMEN

¿Qué logramos en esta clase?

- ✓ *Aprender a recibir y enviar datos entre un controlador y una vista*
- ✓ *Desplegar una vista con datos entregados desde el controlador*

English Always | alkemy

#WorkingTime

Continuemos ejercitando

¡Antes de cerrar la clase! Te invitamos a: 👇👇👇

1. Repasar nuevamente la grabación de esta clase
2. Revisar el material compartido en la plataforma de Moodle (lo que se vio en clase y algún ejercicio adicional)
   a. *Lectura Modulo 6, Lección 2: páginas 26 - 30*
3. Traer al próximo encuentro, todas tus dudas y consultas para verlas antes de iniciar nuevo tema.

Momento:

Time-out!

⏰5 min.