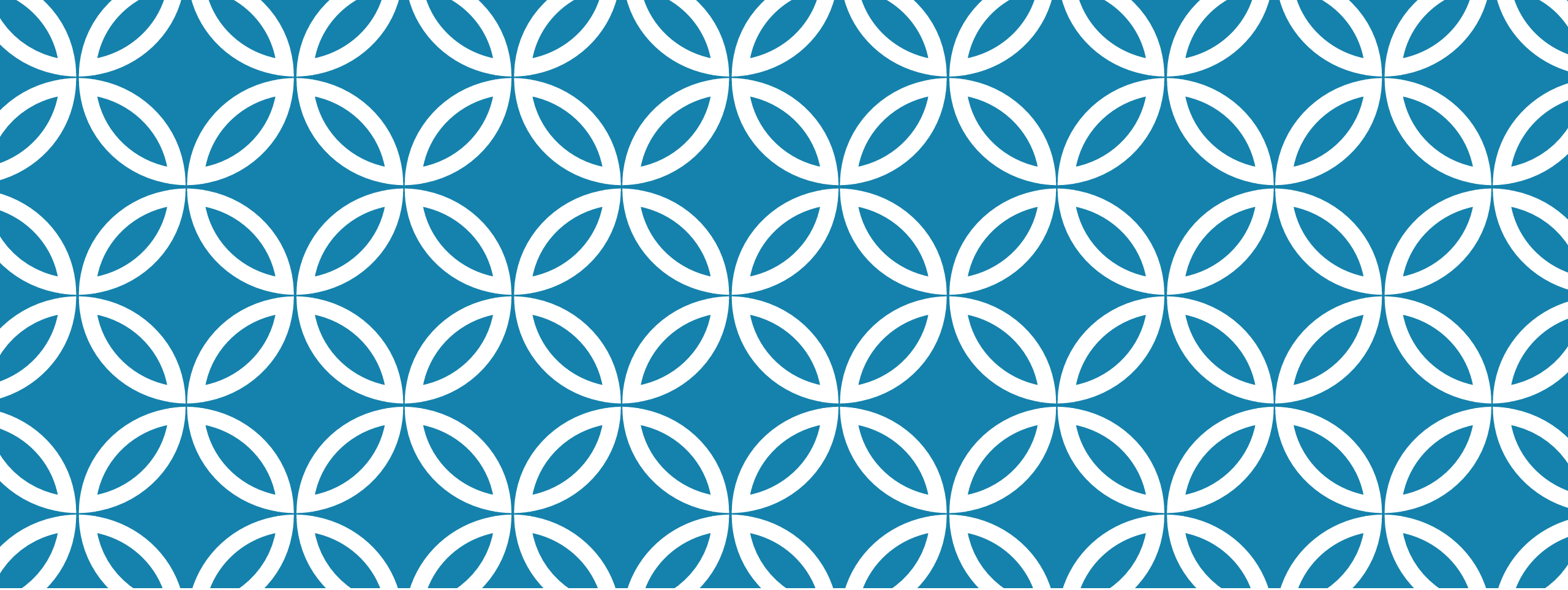
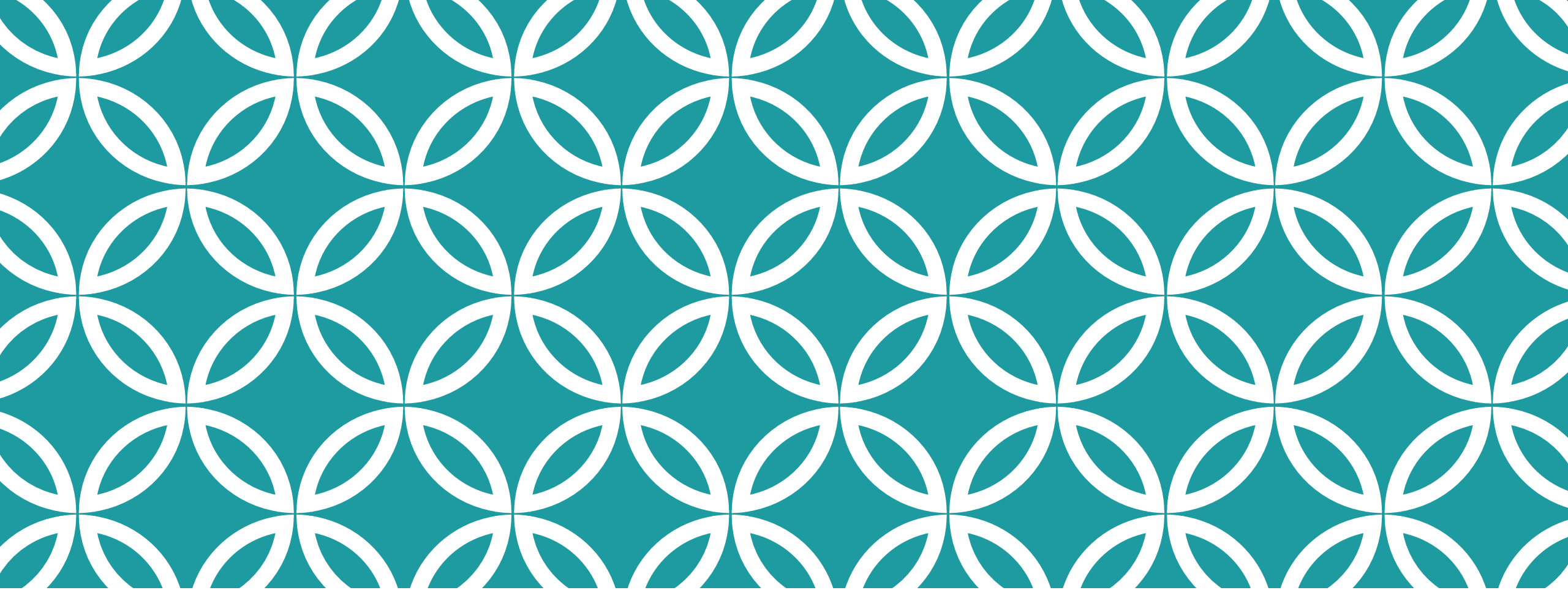


SPRING

Jérémy PERROUAULT



SPRING WEB MVC

Introduction à
Spring Web MVC

PRÉSENTATION DE SPRING MVC

Une Servlet principale : **DispatcherServlet**

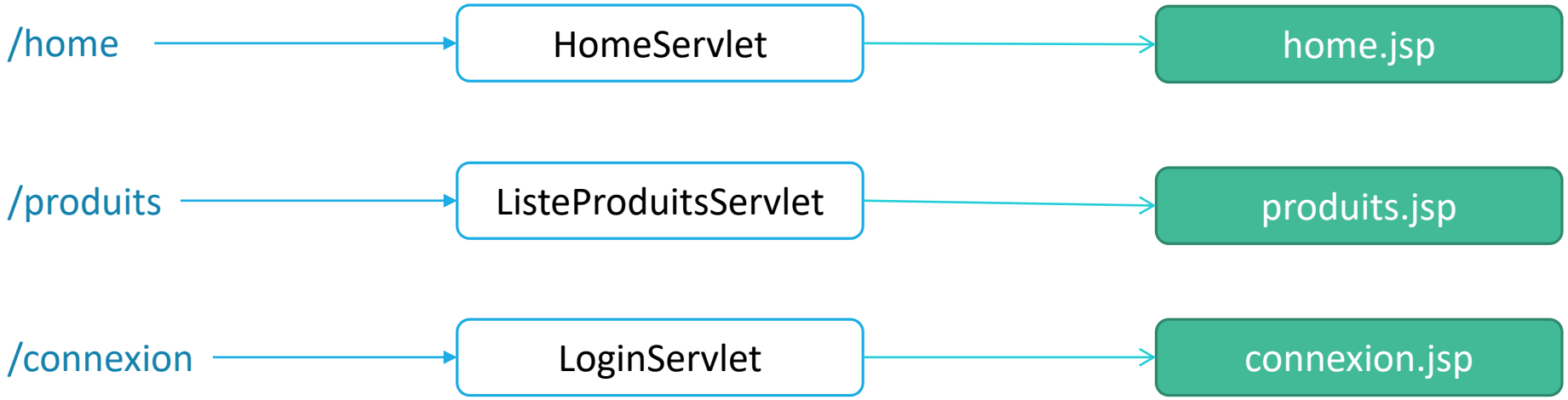
- Délègue les requêtes à des contrôleurs (classes annotées @**Controller**)
  - Selon le point d'accès (la ressource URL)

Un controller

- Fabrique un modèle sous la forme d'une Map qui contient les éléments de la réponse
  - Clé / Valeur
- Utilise une View pour afficher la vue (la page HTML)

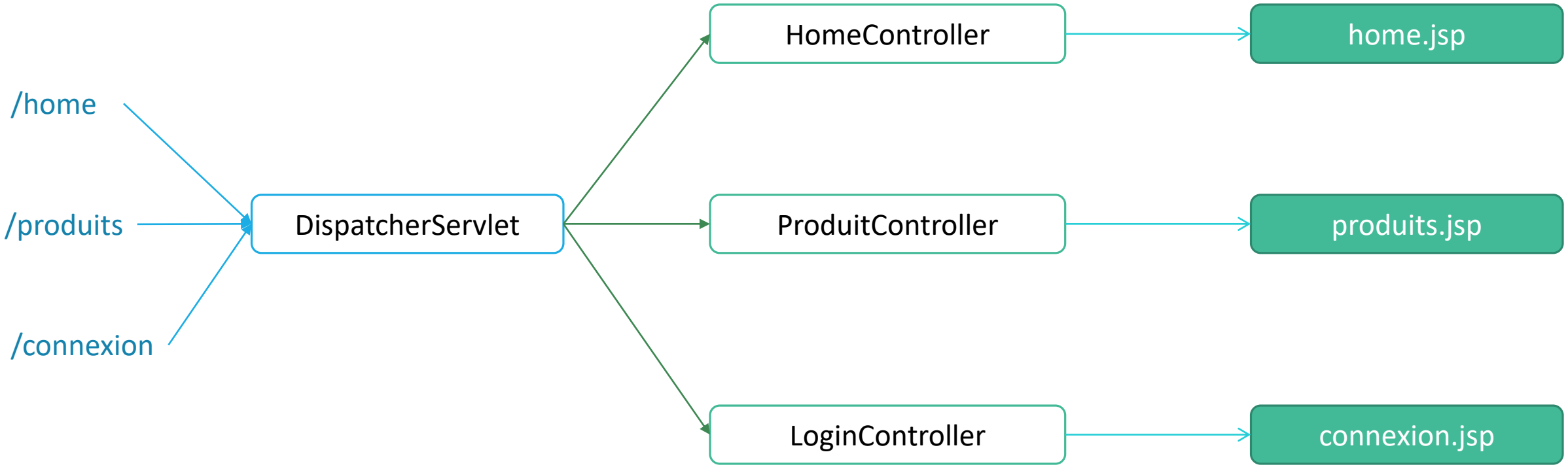
PRÉSENTATION DE SPRING MVC

JEE Servlets classiques



PRÉSENTATION DE SPRING MVC

Avec Spring MVC



PRÉSENTATION DE SPRING MVC

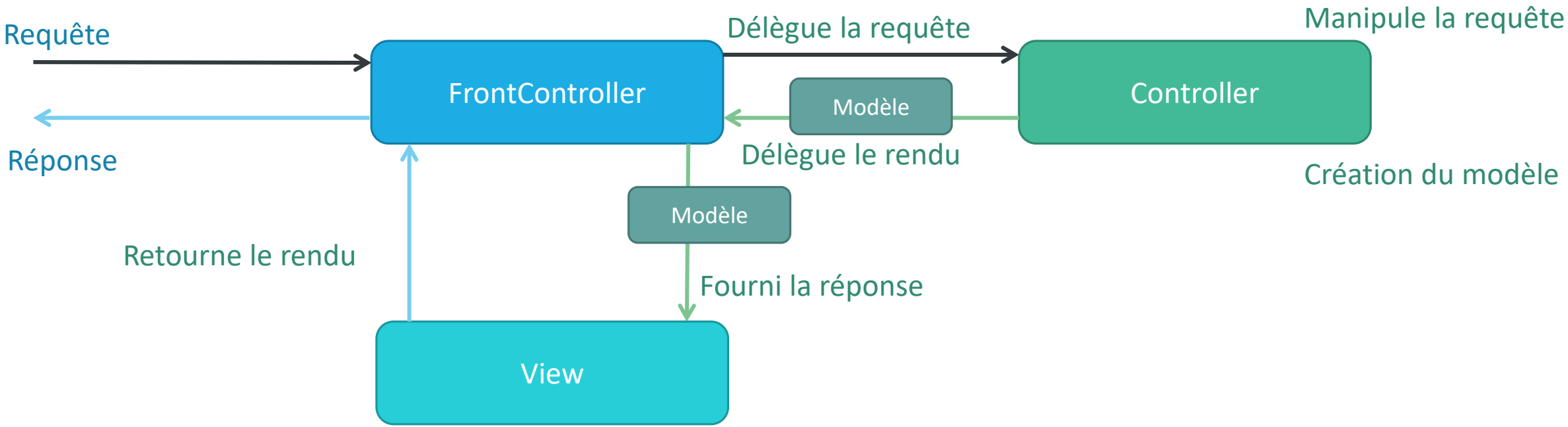
La Servlet **DispatcherServlet** est mappée sur toutes les ressources

- Par exemple « / »
- On l'appelle « FrontController »

Il n'y a plus de Servlet JEE

- Mais si elles existent, leur mapping prend le pas sur le mapping des @Controller !

TRAITEMENT D'UNE REQUÊTE



CONFIGURATION (FRONT CONTROLLER)

Déclaration de la Servlet unique dans le web.xml

```
<servlet>
  <servlet-name>dispatcher</servlet-name>
  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

  <init-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/dispatcher-context.xml</param-value>
  </init-param>

  <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
  <servlet-name>dispatcher</servlet-name>
  <url-pattern>/</url-pattern>
</servlet-mapping>
```

DispatcherServlet utilise son propre fichier de configuration.
Si vous définissez un fichier unique (application-context.xml)
Vous aurez 2 instances que chaque bean Spring !

CONFIGURATION (FRONT CONTROLLER)

Configuration XML avec configuration Spring par classe

```
<servlet>
  <servlet-name>dispatcher</servlet-name>
  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

  <init-param>
    <param-name>contextClass</param-name>
    <param-value>org.springframework.web.context.support.AnnotationConfigWebApplicationContext</param-value>
  </init-param>

  <init-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>fr.formation.config.WebConfig</param-value>
  </init-param>

  <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
  <servlet-name>dispatcher</servlet-name>
  <url-pattern>/</url-pattern>
</servlet-mapping>
```

CONFIGURATION (FRONT CONTROLLER)

Si on veut ajouter la configuration générale Spring (JPA par exemple)

- Ne pas oublier de charger le contexte de Spring

```
<listener>
  <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>
```

- En plus du chargement de la configuration de Spring dans ce fichier **web.xml**

CONFIGURATION (FRONT CONTROLLER)

Dans le fichier de configuration de **DispatcherServlet**

- ➔ dispatcher-context.xml
- Activer le contexte de Spring pour déléguer les requêtes aux contrôleurs
  - Cette balise crée deux beans
    - DefaultAnnotationHandlerMapping
    - AnnotationMethodHandlerAdapter

- Configuration XML

```
<mvc:annotation-driven />
```

- Configuration par classe

```
@EnableWebMvc
```

CONFIGURATION (FRONT CONTROLLER)

DispatcherServlet est mappée sur toutes les ressources ...

- Comment accéder aux ressources CSS, JS, Images, … ?
- Comment distribuer une ressource statique ?

Utilisation de la balise mvc:resources prévue à cet effet

- Dans la configuration de DispatcherServlet (dispatcher-context.xml)

```
<mvc:resources mapping="/css/**" location="/css/" />
<mvc:resources mapping="/images/**" location="/images/" />
```

CONFIGURATION (FRONT CONTROLLER)

En configuration par classe

- La classe de configuration doit implémenter **WebMvcConfigurer**
- Vous devez surcharger la méthode addResourceHandlers()

```
@Configuration
@EnableWebMvc
public class WebConfig implements WebMvcConfigurer {
  public void addResourceHandlers(ResourceHandlerRegistry registry) {
    registry.addResourceHandler("/css/**").addResourceLocations("/css/");
    registry.addResourceHandler("/images/**").addResourceLocations("/images/");
  }
}
```

CONFIGURATION (CONTROLLERS)

Annotation de classes POJO de **@Controller**

Mapping ressource

```
@Controller
public class HomeController {
  @RequestMapping("/home")
  public String home(Model model) {
    model.addAttribute("message", "Allô le monde ?!");

    return "home";
  }
}
```

- « home » fait référence à une vue JSP « home.jsp »
- "message" pourra être lu depuis la vue JSP avec les EL ${ message }

CONFIGURATION (VIEW)

Choisir sa technologie

- Par exemple JSP/JSTL (on se base sur **JSTL**, on a besoin de la dépendance !)

Paramétrer les vues dans dispatcher-context.xml avec un **ViewResolver**

- Nos vues JSP sont dans le répertoire "/WEB-INF/views/"
- Le nom des fichiers se terminent par ".jsp"
- Permettra de retourner "home" au lieu de "/WEB-INF/views/home.jsp" dans le contrôleur

```
<bean id="viewResolver" class="org.springframework.web.servlet.view.UrlBasedViewResolver">
  <property name="viewClass" value="org.springframework.web.servlet.view.JstlView" />
  <property name="prefix" value="/WEB-INF/views/" />
  <property name="suffix" value=".jsp" />
</bean>
```

CONFIGURATION (VIEW)

Définition d'un **@Bean**

```
@Bean
public UrlBasedViewResolver viewResolver() {
  UrlBasedViewResolver viewResolver = new UrlBasedViewResolver();

  viewResolver.setViewClass(JstlView.class);
  viewResolver.setPrefix("/WEB-INF/views/");
  viewResolver.setSuffix(".jsp");

  return viewResolver;
}
```

EXERCICE

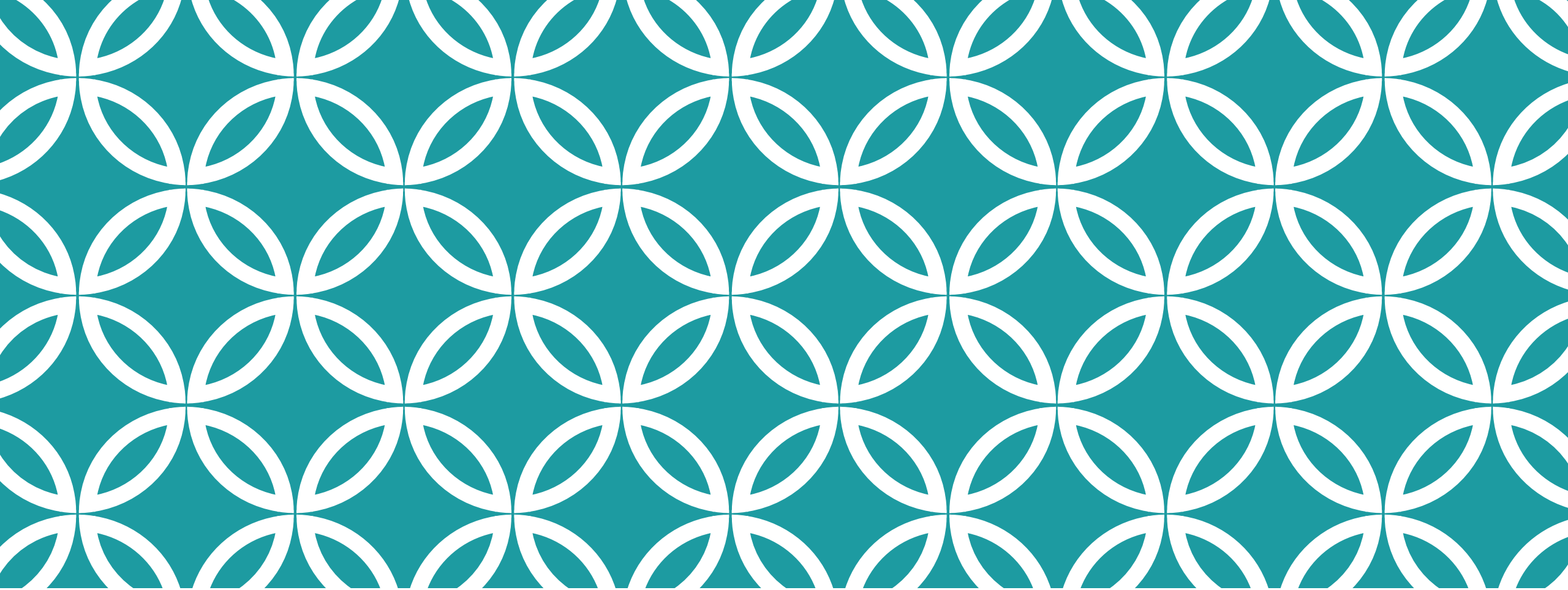
Créer un nouveau projet (Maven)

Configurer Spring MVC

- Dépendance **spring-webmvc**

Implémenter un @**Controller HomeController**

- Mapper une ressource /home
- Passer un attribut « nomUtilisateur » au modèle
- Afficher une JSP « home.jsp » qui affiche « Bonjour nomUtilisateur ! »



CONTROLLER

Le contrôleur

LE CONTROLLER

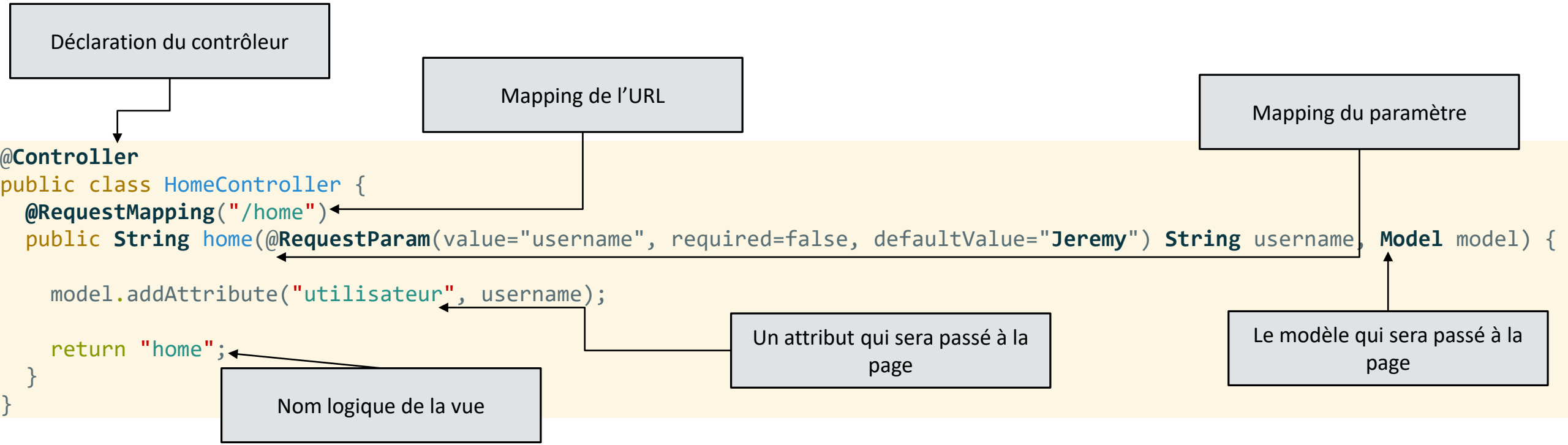
Le contrôleur est utilisé pour traiter une requête

Spring permet

- De mapper l'URL de la requête à une méthode d'un contrôleur
- De mapper les paramètres de la requête aux paramètres de la méthode
- De mapper des objets du contexte de la requête avec les paramètres de la méthode

LE CONTROLLER

- http://.../home affichera « Bonjour Jérémy ! »
- http://.../home?username=Toto affichera « Bonjour Toto ! »



Note : on peut rediriger en utilisant "redirect:url" (exemple : "redirect:/account/subscribe")

LE CONTROLLER

La méthode mappée retourne une chaîne de caractères

- Utilisée pour retourner le nom de la vue
- Peut être utilisée pour rediriger l'utilisateur vers une autre adresse URL
  - Avec "redirect:/url"

```
@Controller
public class HomeController {
  @RequestMapping("/home")
  public String home(@RequestParam("username") String username, Model model) {
    return "redirect:/login";
  }
}
```

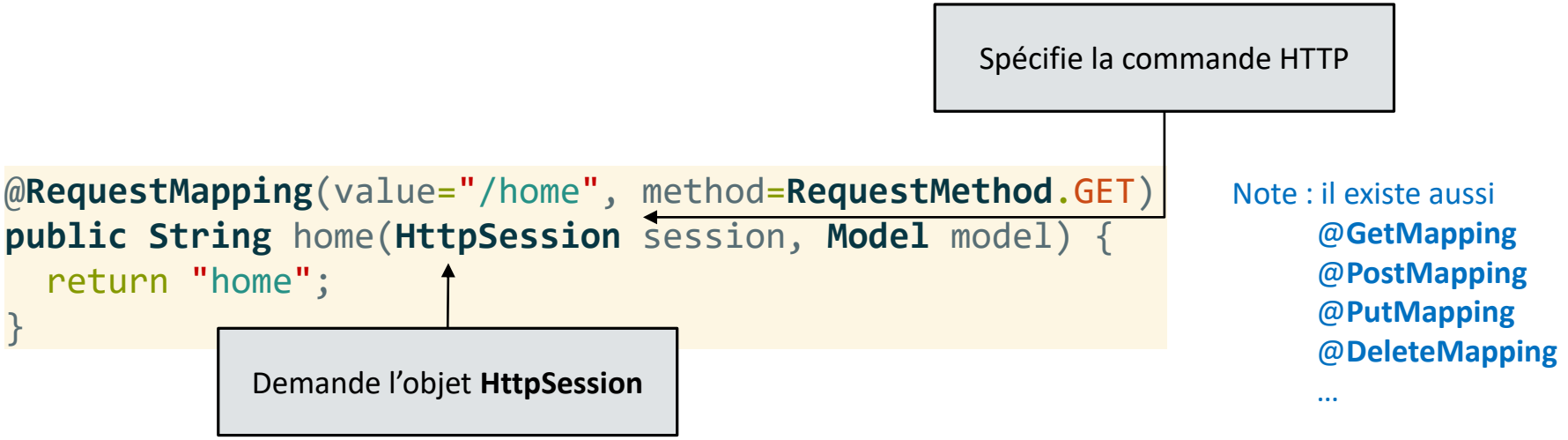
LE CONTROLLER

Spécifier le type de commande HTTP (GET, POST, PUT, DELETE, …)

- Par défaut, toutes les commandes HTTP sont mappées sur l'URL avec **@RequestMapping**

Demander l'objet **HttpSession** (qui nous permettra de manipuler la session)

- Principe d'injection de dépendance basé sur le type



Note : il existe aussi
@GetMapping
@PostMapping
@PutMapping
@DeleteMapping
…

LE CONTROLLER

Variables dans la ressource et multiple mapping d'URL

- http://.../home affichera « Bonjour ! »
- http://.../home/Toto affichera « Bonjour Toto ! »

```
@GetMapping(value={ "/home", "/home/{username}" })
public String home(@PathVariable(value="username", required=false) String username, Model model) {
  model.addAttribute("utilisateur", username);

  return "home";
}
```

LE CONTROLLER

Il est possible de mapper toutes les méthodes d'un contrôleur

- En annotant le contrôleur de **@RequestMapping**

```
@Controller
@RequestMapping("/produit")
public class ProduitController {
  @GetMapping("/liste")
  public String getProduits() {
    return "produits";
  }

  @GetMapping("/{nomProduit}")
  public String getProduit(@PathVariable(value="nomProduit") String produit) {
    return "produit";
  }
}
```

http://.../produit/liste ➔ produits.jsp
http://.../produit/gopro ➔ produit.jsp
http://.../produit/iphone ➔ produit.jsp

EXERCICE

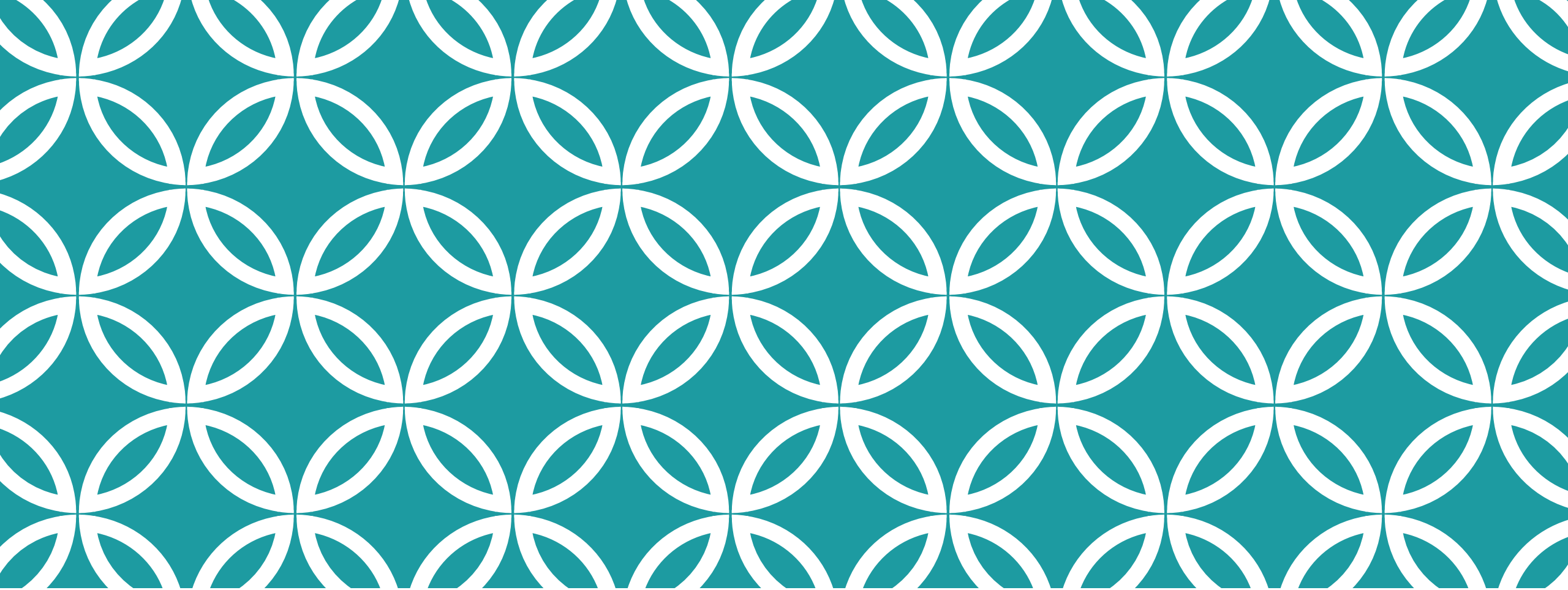
Modifier la méthode « home » du contrôleur

- Doit attendre un paramètre « idProduit »
- Doit attendre une variable de chemin dans l'URL « username »
- Ne fonctionne qu'en méthode GET

Afficher sur la JSP la valeur de « idProduit » et de « username »

RAPPEL : pour utiliser plusieurs paramètres dans l'URL :

- url?param1=valeur1&param2=valeur2



THYMELEAF

Configurer Thymeleaf

THYMELEAF

Thymeleaf est un moteur de rendu pleinement compatible avec Spring

- Dépendances **thymeleaf-spring5** et **thymeleaf-layout-dialect**

Configuration

- D'un **ViewResolver**
- D'un **TemplateEngine**
- D'un **TemplateResolver**

THYMELEAF

Coniguration du Bean du **TemplateResolver**

```
<bean id="templateResolver" class="org.thymeleaf.spring5.templateresolver.SpringResourceTemplateResolver">
  <property name="prefix" value="/WEB-INF/templates/" />
  <property name="suffix" value=".html" />
  <property name="templateMode" value="HTML" />
</bean>
```

THYMELEAF

Bean du **TemplateEngine** à configurer

- Pour lequel on donnera la référence du bean **TemplateResolver**
- On lui ajoutera également le dialect **LayoutDialect**

```
<bean id="templateEngine" class="org.thymeleaf.spring5.SpringTemplateEngine">
  <property name="templateResolver" ref="templateResolver" />
  <property name="enableSpringELCompiler" value="true" />
  <property name="additionalDialects">
    <set>
      <bean class="nz.net.ultraq.thymeleaf.LayoutDialect"/>
    </set>
  </property>
</bean>
```

THYMELEAF

Bean du **ViewResolver** dans Spring

- Pour lequel on donnera la référence du bean **TemplateEngine**

```
<bean id="viewResolver" class="org.thymeleaf.spring5.view.ThymeleafViewResolver">
  <property name="templateEngine" ref="templateEngine" />
</bean>
```

THYMELEAF

Définition du @Bean **TemplateResolver**

```
@Bean
public SpringResourceTemplateResolver templateResolver() {
  SpringResourceTemplateResolver templateResolver = new SpringResourceTemplateResolver();

  templateResolver.setPrefix("/WEB-INF/templates/");
  templateResolver.setSuffix(".html");

  return templateResolver;
}
```

THYMELEAF

Définition du @Bean **TemplateEngine**

```
@Bean
public SpringTemplateEngine templateEngine(SpringResourceTemplateResolver templateResolver) {
  SpringTemplateEngine templateEngine = new SpringTemplateEngine();

  templateEngine.setTemplateResolver(templateResolver);
  templateEngine.setEnableSpringELCompiler(true);
  templateEngine.addDialect(new LayoutDialect());

  return templateEngine;
}
```

THYMELEAF

Définition du @Bean **ViewResolver**

```
@Bean
public ThymeleafViewResolver viewResolver(SpringTemplateEngine templateEngine) {
  ThymeleafViewResolver viewResolver = new ThymeleafViewResolver();

  viewResolver.setTemplateEngine(templateEngine);
  return viewResolver;
}
```

EXERCICE

Implémenter Thymeleaf

Mettre en place le layout (la structure) du site

- Basé sur Bootstrap
- Titre
- Navigation (menu)
- Contenu dynamique

Réaliser le CRUD de « produit »