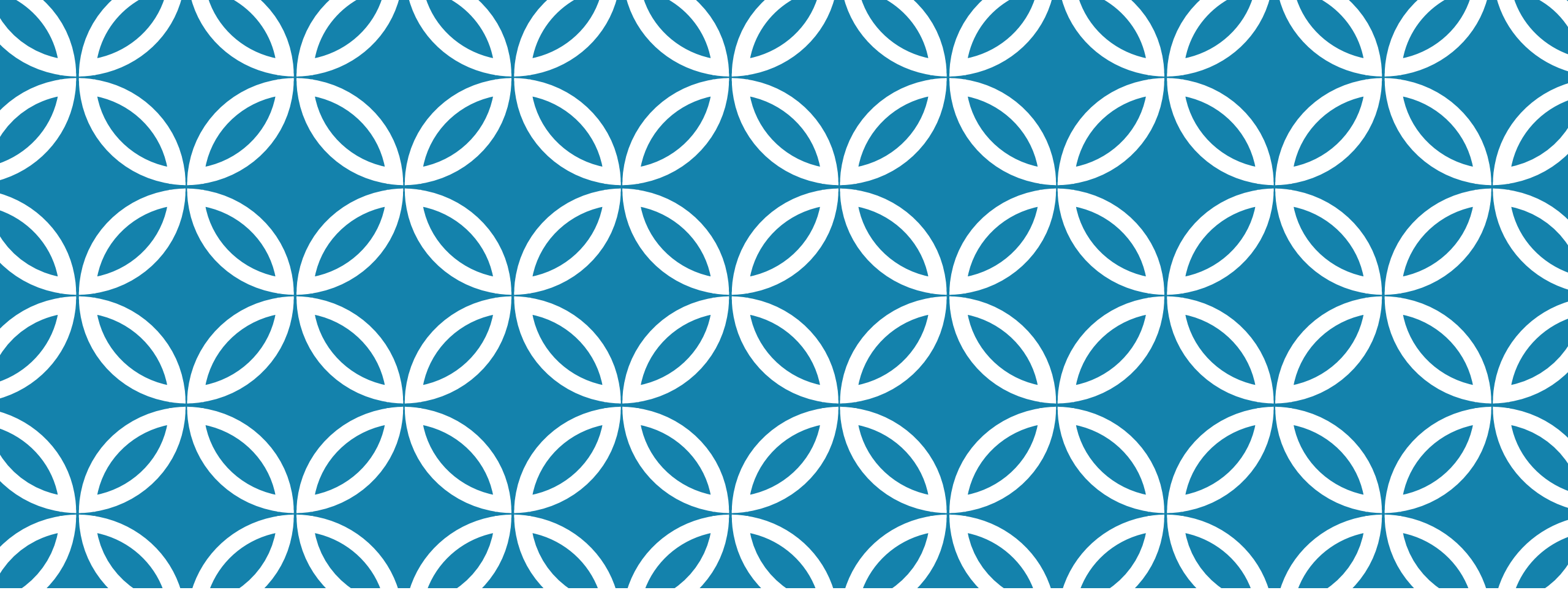
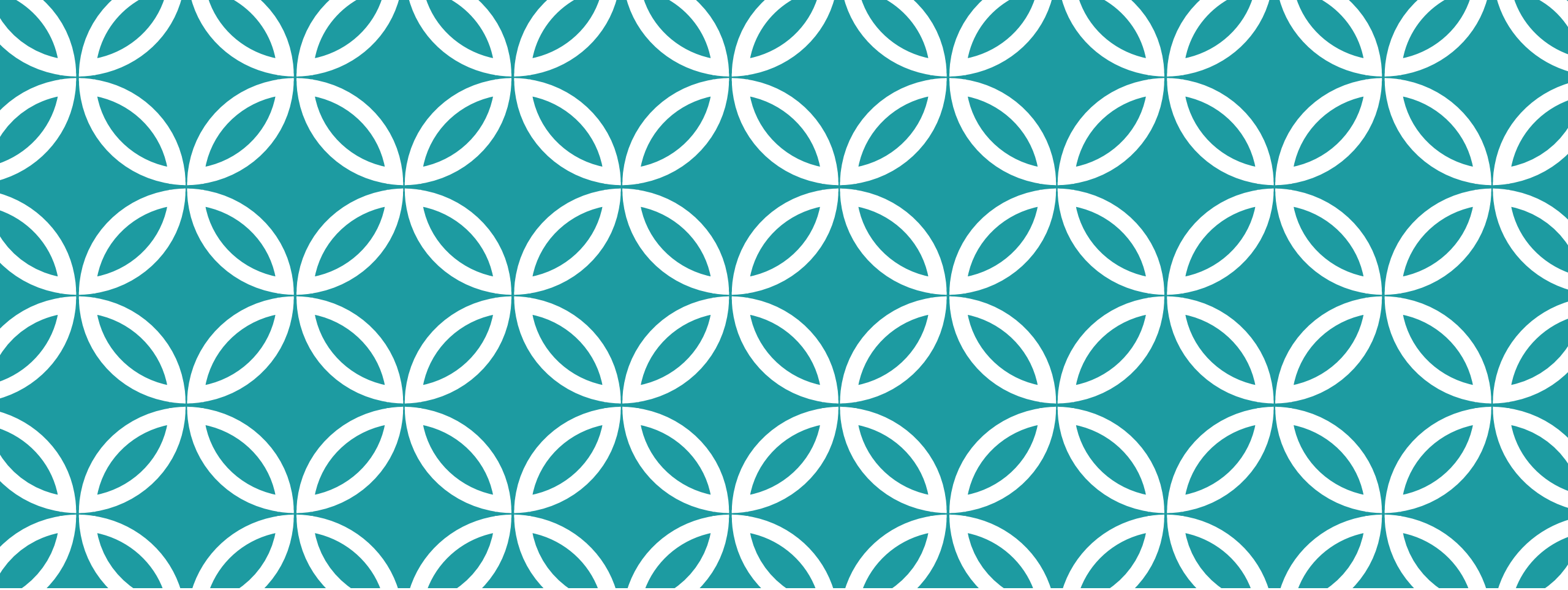


# SPRING



# DATA BINDING

Data Binding

# DATA BINDING

Des données issues d'un formulaire

- Il faut mapper les propriétés de la classe une à une
  - setNom(request.getParameter("nom"))
  - setPrix(request.getParameter("prix"))
  - … même chose en utilisant @**RequestParam**

Le Data Binding va nous éviter tout ça !

Utilisation de l'annotation @**ModelAttribute**("nom_model")

# DATA BINDING

**@ModelAttribute**("produit") associe les paramètres du formulaire à l'objet

- C'est Spring qui se chargera d'utiliser les setters de Produit, à notre place

```java
@PostMapping("/produit/nouveau")
public String ajouterProduit(@ModelAttribute("produit") Produit produit, Model model) {
  System.out.println("Nom : " + produit.getNom() + " Prix : " + produit.getPrix());
  //...

  return "redirect:/produits";
}
```

Récupérer l'objet bindé via l'attribut "user" de Model

Note : **@ModelAttribute** n'est pas obligatoire
Spring cherchera un attribut du même nom que le paramètre
Ou, le cas échéant, cherchera un attribut du même type

# DATA BINDING

```
<form method="POST">
  <table>
    <tr>
      <td><label for="nom">Nom</label></td>
      <td><input id="nom" name="nom" type="text" th:value="${produit.nom}" /></td>
    </tr>

    <tr>
      <td><label for="prix">Prix</label></td>
      <td><input id="prix" name="prix" type="number" /></td>
    </tr>

    <tr>
      <td colspan="2"><input type="submit" value="Ajouter"></td>
    </tr>
  </table>
</form>
```

Objet du modèle et la valeur de sa propriété

Nom des propriétés

# DATA BINDING

L'attribut *modelAttribute* du formulaire Spring nécessite un attribut par défaut

- Il faut donc le créer et l'ajouter au Model
  - Soit à l'affichage du formulaire

```
@GetMapping("/produit/nouveau")
public String ajouterProduit(Model model) {
  model.addAttribute("produit", new Produit());
  return "form-produit";
}
```

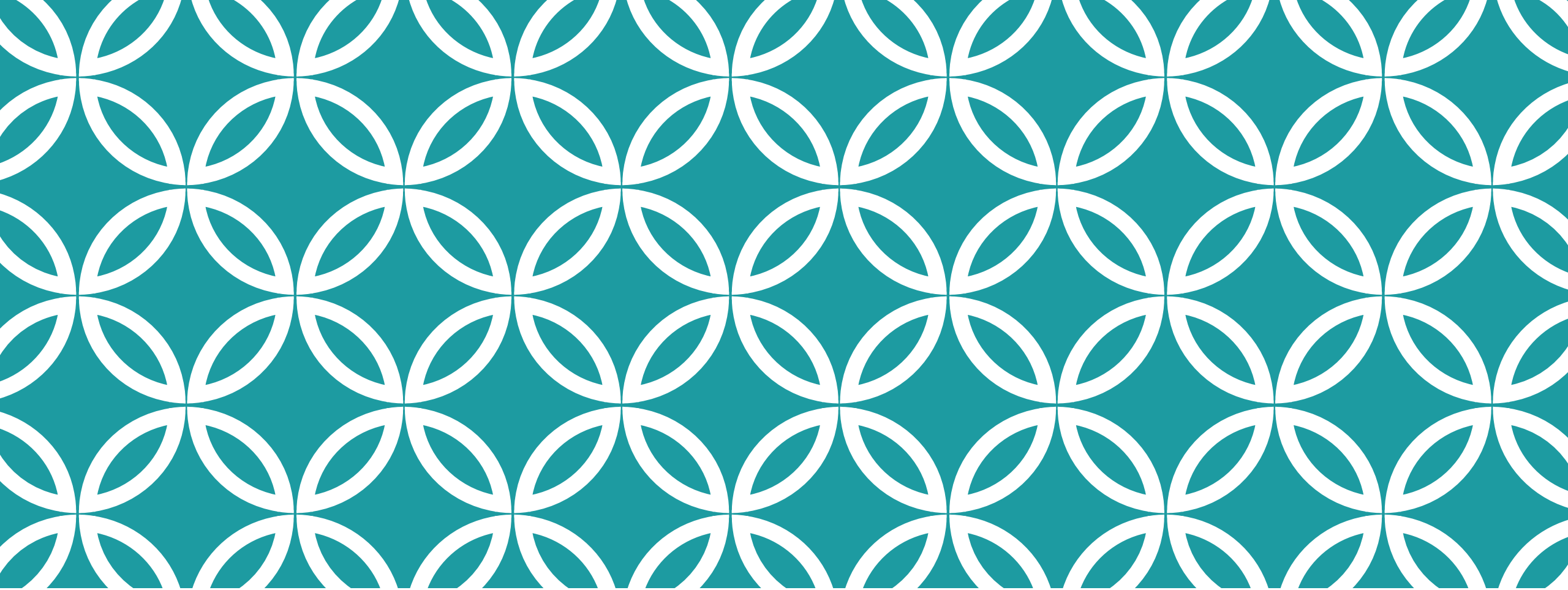
  - Soit via une méthode annotée de @**ModelAttribute** (directement dans le contrôleur)

```
@ModelAttribute("produit")
public Produit initProduit() {
  return new Produit();
}
```

# EXERCICE

Modifier le CRUD « produit » en conséquence !

- Retirer les @**RequestParam**



# VALIDATION

Validation

# VALIDATION

Le principe de la validation

- Vérifier si les champs obligatoires sont remplis
- Vérifier si une valeur est comprise entre x et y
- Ces validations sont à faire, une à une, dans la méthode POST

Spring MVC et l'API de validation vont nous éviter tout ça !

Utilisation de l'annotation @**Valid**

# VALIDATION

Ajouter @**Valid** devant @**ModelAttribute**

Active la validation api-validator (hibernate-validator)

```
@PostMapping("/produit/nouveau")
public String ajouterProduit(@Valid @ModelAttribute("produti") Produit produit, BindingResult result, Model model) {
  if (result.hasErrors()) {
    System.out.println("Le produit n'a pas été validé ...");
    return "form-produit";
  }

  return "redirect:/produits";
}
```

Permet de connaître les erreurs lors du Bind, si erreur il y a

Si utilisation de plusieurs @**ModelAttribute**

- Il faut placer un BindingResult juste après un @**ModelAttribute**
  - Celui qui suit @**ModelAttribute** lui correspond

# VALIDATION

2 options sont possibles pour la validation

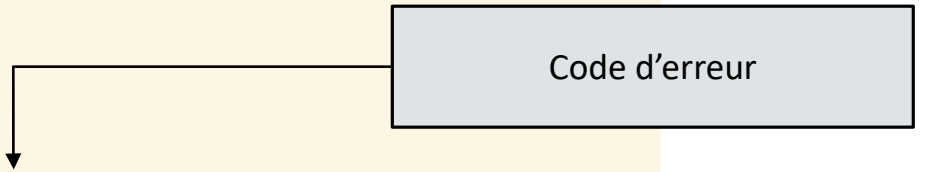
- Utiliser une classe qui implémente l’interface "Validator"
- Utiliser Hibernate-Validation

# VALIDATION – VALIDATOR

Implémenter une classe qui implémente *Validator* (SpringFramework)

```
public class ProduitValidator implements Validator {
  @Override public boolean supports(Class<?> cls) {
    return Produit.class.equals(cls);
  }

  @Override public void validate(Object obj, Errors e) {
    ValidationUtils.rejectIfEmptyOrWhitespace(e, "nom", "nom.empty", "Le nom doit être saisi");
  }
}
```



# VALIDATION – VALIDATOR

La méthode @**RequestMapping** nécessite un Validator

- Soit en première instruction de la méthode (dans ce cas, @**Valid** n'est plus nécessaire)

```
@PostMapping("/produit/nouveau")
public String ajouterProduit(@Valid @ModelAttribute("produit") Produit produit, BindingResult result, Model model) {
  new ProduitValidator().validate(produit, result);
  //...
}
```

- Soit via une méthode annotée de @InitBinder (directement dans le contrôleur)

```
@InitBinder
protected void initBinder(WebDataBinder binder) {
  binder.addValidators(new ProduitValidator());
}
```

# VALIDATION – HIBERNATE-VALIDATOR

Il suffit d'annoter les propriétés de la classe

```
public class Produit {
  @NotEmpty(message = "Le nom est obligatoire")
  private String nom;
}
```

# VALIDATION

Pour afficher les messages d'erreur dans la page JSP

- <th:errors="*${nom_model.nom_propriete}*" />

```
<form method="POST">
  <table>
    <tr>
      <td><label for="nom">Nom</label></td>
      <td><input id="nom" name="nom" type="text" th:value="${produit.nom}" /></td>
      <td><span th:errors="${produit.nom}">Message d'erreur par défaut</span></td>
    </tr>
  </table>
</form>
```

# EXERCICE

Modifier le CRUD « produit »

- Utiliser @**ModelAttribute** et la validation hibernate-validation
- Message si le nom / prix n’est pas saisi ou mal saisi

# EXERCICE – ALLER PLUS LOIN

Mettre en place la validation pour une inscription Utilisateur

- Le nom, le prénom, le nom d'utilisateur et le mot de passe sont obligatoires
- Le mot de passe et le mot de passe de vérification doivent correspondre

  - Combinez la validation par Hibernate-Validator et la validation par une classe Validator
    - Utilisez e.rejectValue() dans ce Validator pour déclencher la non-validation