Labo Spring 3
- *Formulaire* -

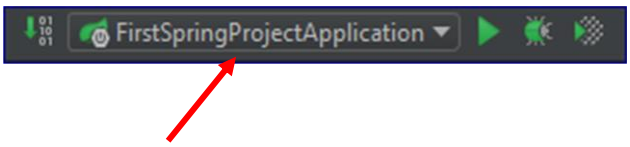
Objectif
- Créer des formulaires et les valider

Un formulaire va être ajouté dans la page *welcome.jsp* afin de demander à l'utilisateur une clé (un code) permettant d'accéder à la suite de l'application Web.

Maven Build

Si vous ne l'avez pas encore fait, créez un script constitué des étapes *clean* et *install*. Il suffira ensuite de lancer ce script chaque fois que des modifications seront apportées (par exemple, lorsqu'une nouvelle dépendance aura été ajoutée dans le pom.xml).

Créez une "*launch configuration*". Celle-ci permettra de facilement faire un clean du projet suivi d'une installation.

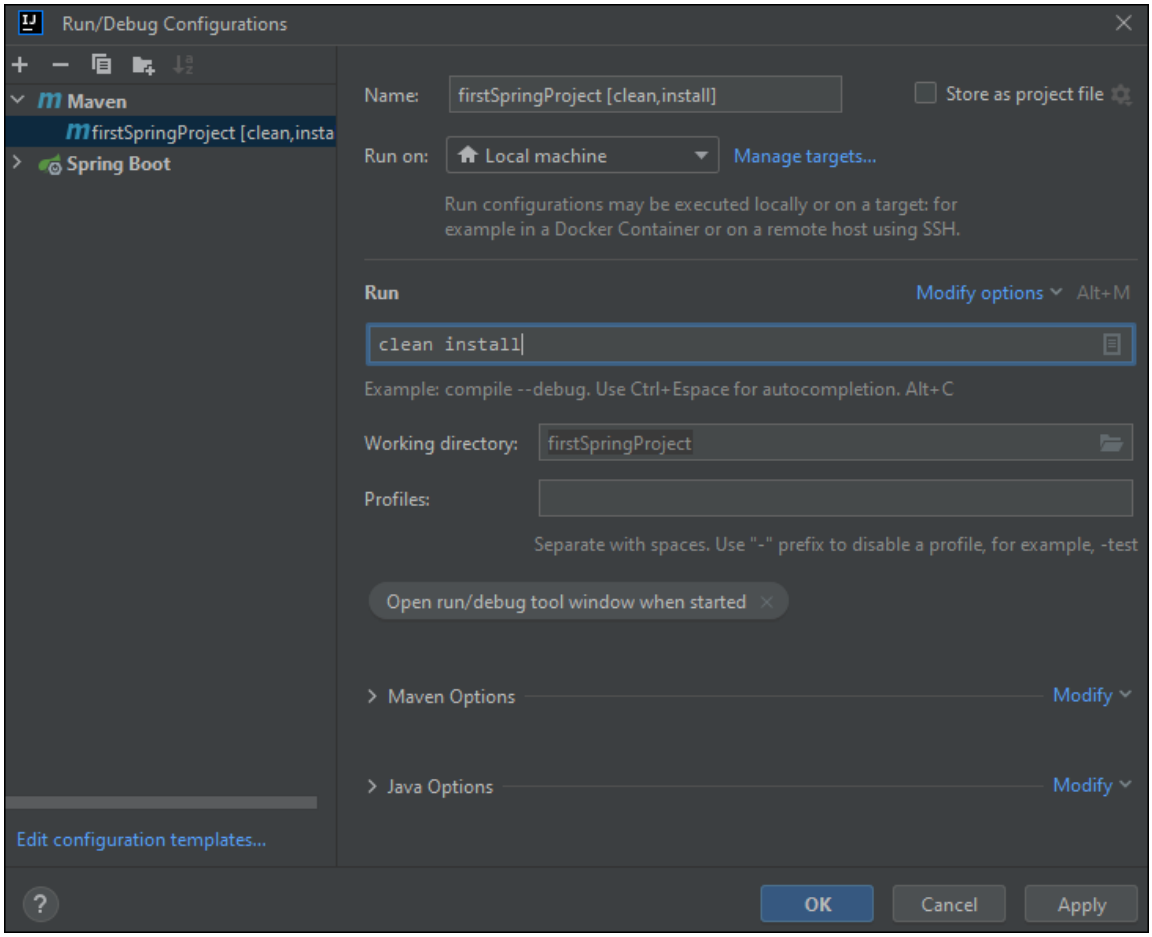


Sélectionnez le projet dans la barre de navigation

⇨ Edit Configurations ⇨ Clic sur le symbole + ⇨ Maven

Name : donner un nom Par exemple : *firstSpringProject[clean,install]*

⇨ Run: clean install



⇨ Apply ⇨ OK

Etape 1 – Classe modèle *MagicKeyForm*

Data Binding entre une page Web et une classe modèle

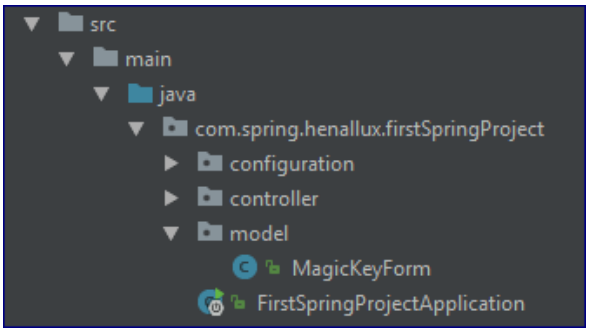
Pour chaque formulaire de saisie de données prévu dans une page jsp doit être créée une classe modèle dont chaque variable d'instance (privée) correspondra à un champ du formulaire (+ getters/setters publiques et au moins un constructeur sans argument).

Le but est que, par **Data Binding**, Spring affecte à chaque variable d'instance (de la classe modèle) la valeur introduite par l'utilisateur dans le champ du formulaire correspondant.

Créez le package *model* dans *com.spring.henallux.firstSpringProject.*

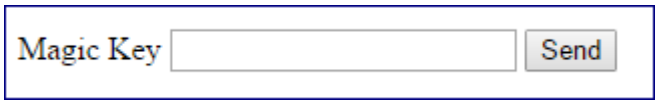
Créez-y la classe *MagicKeyForm* qui contient une variable d'instance privée de type String appelée *magicKey*.

Prévoyez le getter et le setter publiques pour cette variable d'instance ainsi que, au minimum, le constructeur sans argument.



Etape 2 – Formulaire dans la page *welcome.jsp*

Ajoutez dans la page *welcome.jsp* le formulaire ci-dessous de saisie d'une clé permettant de continuer à naviguer sur le site. Placez dans le formulaire un bouton appelé *Send*.



Attention, vous devez utiliser la balise ***<form:form>*** pour créer le formulaire ainsi que les balises ***<form:label>*** pour le label, ***<form:input>*** pour la zone de saisie et ***<form:button>*** pour le bouton.

Référez-vous au cours pour construire ce formulaire (Cf. Module 6 – Controller ⇨ Voir Form Controller – *Jsp Page*).

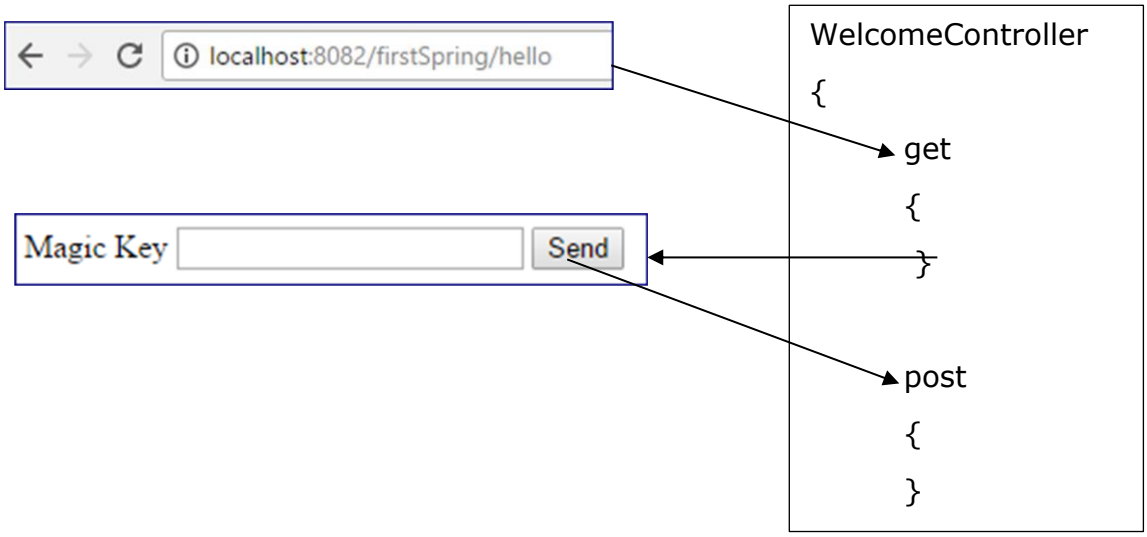
Etape 3 – Classe *WelcomeController*

Soumettre un formulaire (post)

Un clic sur un bouton de soumission d'un formulaire a pour effet d'appeler une méthode post du *controller*. La méthode correspondante doit être déclarée de type *post*.

L'url correspondant à l'appel de cette méthode est précisé dans le paramètre *action* de la balise *<form:form>* du formulaire.

Un clic sur le bouton *Send* devra avoir pour effet d'appeler la méthode *post* de la classe *WelcomeController* :



Ce controller doit créer un objet de la classe *MagicKeyForm* et être implémenté de sorte que la valeur introduite par l'utilisateur soit placée dans la variable *magicKey* de cet objet.

Pour ce faire, prévoyez l'instruction de création et d'ajout de cet objet de type *MagicKeyForm* au dictionnaire *Model* dans la méthode ***get*** (méthode qui a pour effet d'afficher la page contenant le formulaire vierge), dont la clé sera la même que la valeur de l'attribut *modelAttribute* de la balise <*form:form*> :

```
model.addAttribute("magicKeyForm", new MagicKeyForm());
```

Récupérez ensuite cet objet du *Model* comme argument de la méthode ***post*** :

```
import org.springframework.web.bind.annotation.ModelAttribute;
```

```
@RequestMapping(value="/send", method=RequestMethod.POST)
public String getFormData(@ModelAttribute(value="magicKeyForm") MagicKeyForm form) {
```

La variable *magicKey* de l'objet *form* sera remplie avec la valeur introduite par **l'utilisateur à condition que**, dans la balise <*form:form*> de la page jsp, les bonnes valeurs aient été placées dans les attributs *action* (path pour appeler la méthode *post* du controller) et *modelAttribute* (lien vers l'objet modèle).

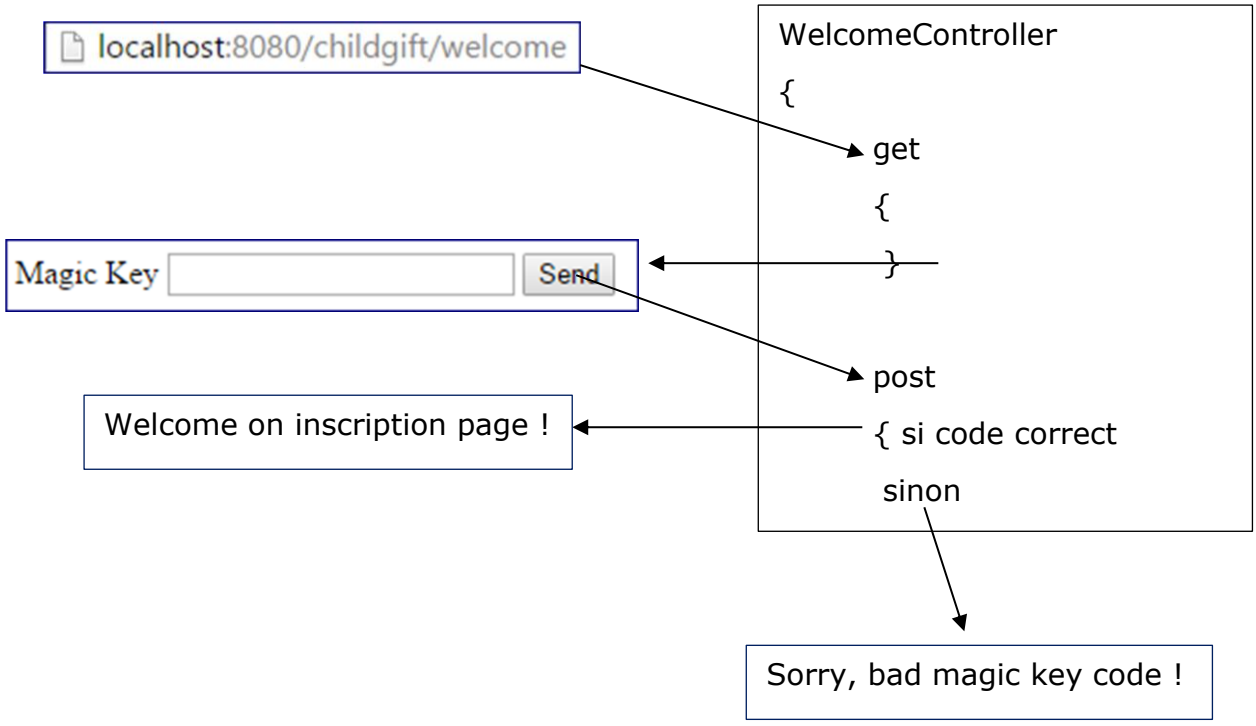
```
<form:form id="form"
           method="POST"
           action ="/firstSpring/hello/send"
           modelAttribute="magicKeyForm">
```

De même, il faut que les champs du formulaire soient reliés aux variables d'instance de l'objet modèle via l'attribut *path (*pour le Data Binding*)*.

```
<form:input path="magicKey"/>
```

Nom de la variable d'instance dans la classe modèle

Dans notre exercice, la méthode *post* doit ensuite vérifier que la valeur introduite par l'utilisateur (qui se trouve donc dans la variable *magicKey* de l'objet *form*) est une valeur correcte. Si oui, une nouvelle page *userInscription.jsp* doit être affichée. Cette page contient pour l'instant seulement un message de bienvenue. Si non, une page d'erreur (*keyError.jsp*) doit être affichée prévenant l'utilisateur qu'il a entré une mauvaise clé.



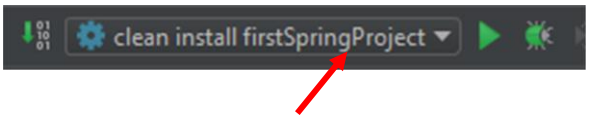
Créez ces deux nouvelles pages *userInscription.jsp* et *keyError.jsp.*

La page *userInscription.jsp* affiche pour l'instant : "*Welcome on inscription page !*" et la page *keyError.jsp* affiche le message : "*Sorry, bad magic key code !*".

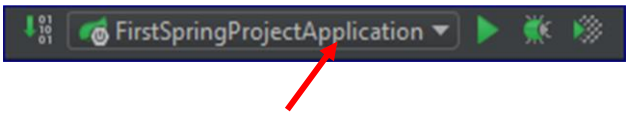
Idéalement, la valeur introduite par l'utilisateur devrait être comparée à des valeurs stockées dans une base de données (cf. exercice ultérieur). Pour l'instant, la valeur introduite est juste comparée à une liste de valeurs hardcodée dans le controller.

Faites un clean+install de l'application puis déployez-la.

Build (clean + install) de l'application

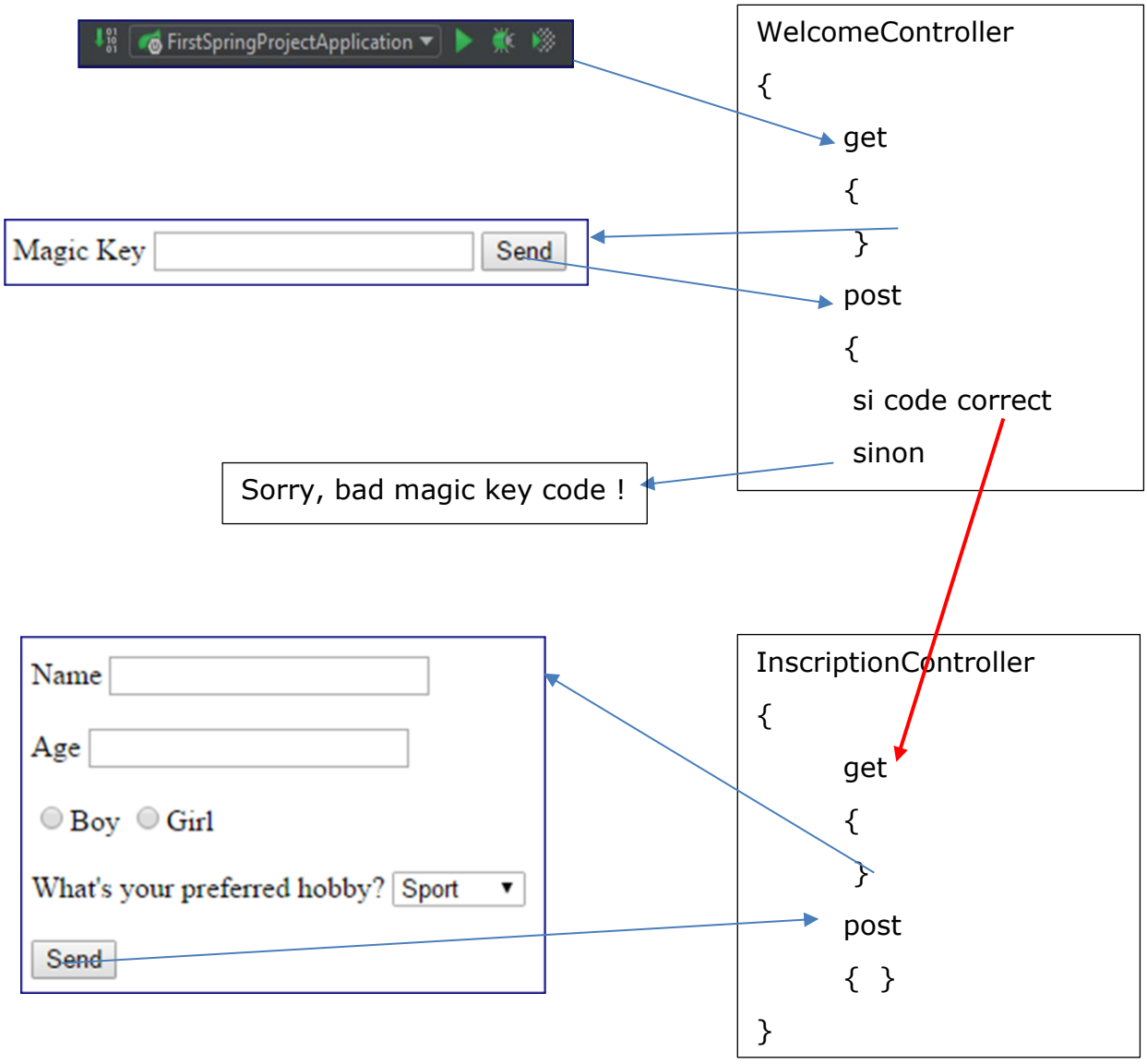


Déploiement de l'application sur le serveur Tomcat



Dans la suite de l'exercice, la page *userInscripton* va contenir un formulaire qui permettra de saisir les coordonnées de l'enfant (étapes 4 et 5). Par conséquent, un nouveau controller va gérer la page *userInscription*, soit la classe *InscriptionController* (étape 6).

Attention, la méthode *post* de *WelcomeController* ne devra plus retourner directement la page *userInscription*, mais faire appel à la méthode *get* de *InscriptionController* (étape 7).



Etape 4 – Classe modèle *User*

Un formulaire de création d'un utilisateur va être affiché. Il faut donc créer la classe modèle *User* qui va récupérer par Data Binding dans ses variables d'instance les valeurs introduites dans les champs du formulaire

Dans le package *model*, créez la classe *User* qui contient les variables d'instance privées suivantes :

name : de type String

age : de type **Integer**

male : de type **B**oolean

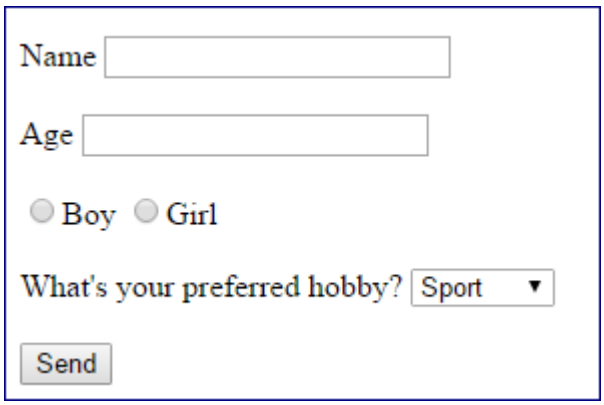
hobby : de type String

Prévoyez les getters/setters publiques ainsi qu'au moins le constructeur sans argument.

Etape 5– Page userInscription

Adaptez la page *userInscription.jsp.*

Celle-ci doit contenir le formulaire suivant :



Pour les boutons radio, utilisez la balise *<form:radiobutton>* avec les attributs *path* (cf. Data Binding : lien avec la variable d'instance correspondante), *value* ("true" ou "false") et *label* ("Boy" ou "Girl").

```
<form:radiobutton path="male" value="true" label="Boy" />
```

Pour la liste, utilisez la balise *<form:select>* avec l'attribut *path* (cf Data Binding) et *<form:option>* avec les attributs ***value*** **(valeur réelle qui sera récupérée dans la variable d'instance de l'objet modèle)** et *label* (valeur affichée dans la page). Placez-y au moins les hobbies suivants : *Sport, Nature, Reading* et *Music.*

```
<form:select path="...">
        <form:option value="..." label="..." />
        <form:option value="..." label="..." />
        <form:option value="..." label="..." />
</form:select>
```

Etape 6 – Classe InscriptionController

Une nouvelle classe de type Controller va se charger d'afficher la page contenant ce formulaire de création d'un nouvel utilisateur, soit la page *userInscription.jsp*.

La classe *InscriptionController* contient donc une méthode *get* qui sera appelée pour afficher la page contenant le formulaire vierge et une méthode *post* qui sera appelée lors du clic sur le bouton *send*.

Créez la classe *InscriptionController* dans le package *controller*.

Dans la méthode *get*, créez un objet de la classe *User*, placez-le dans le dictionnaire *Model* et retournez la page *userInscription.jsp*.

Implémentez la méthode *post* de sorte que les valeurs introduites par l'utilisateur dans les champs du formulaire soient placées dans les variables d'instance de l'objet modèle de type *User* (quand clic sur le bouton *send*). Créez et retournez une page *gift.jsp* qui ne contient pour l'instant qu'un message de félicitations.

N.B. Faire le point 7 avant de tester !

Etape 7 - Redirection

Redirection vers une méthode *get* d'un controller

Parfois, une méthode *post* ne doit pas se contenter de retourner directement une page Web. En effet, dans certains cas, une "préparation" de la page doit avoir lieu avant de l'afficher. C'est le cas notamment si la page contient un formulaire ; il faut créer un objet de type modèle au préalable et passer cet objet modèle via le dictionnaire pour que la page Web y ait accès.

Le type de retour de la méthode *post* est alors une **redirection** vers une méthode *get* d'un autre controller qui a pour effet de préparer la page Web et de l'afficher.

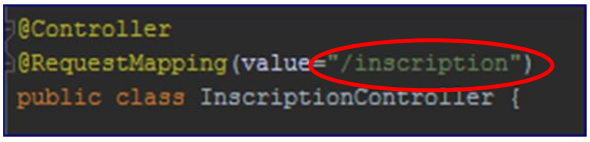
Dans notre application, la page *userInscription.jsp* contient désormais un formulaire qui attend un objet modèle.

Il faut donc que la méthode *get* du controller correspondant soit appelée avant chaque affichage de la page *userInscription* pour que l'objet modèle soit créé !

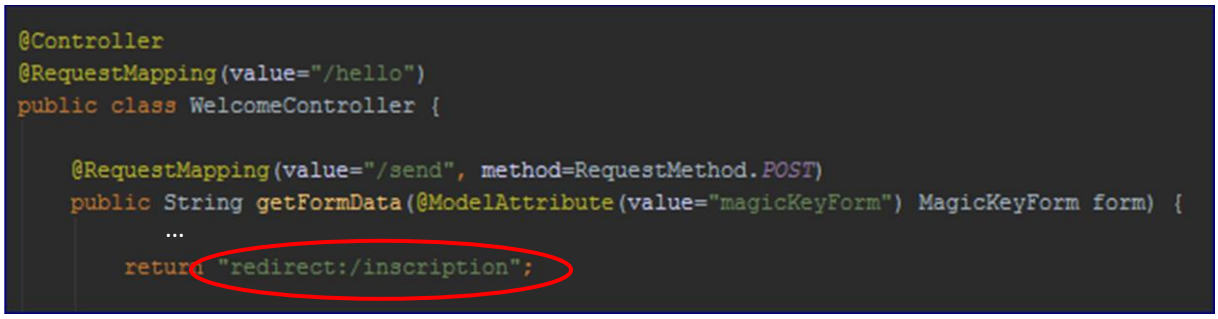
Il faut donc **modifier la méthode *post* de *WelcomeController*** !!! La méthode *post* ne retourne plus directement une page jsp (*return "integrated:userInscription*") mais doit appeler le controller de cette page.

L'instruction à écrire est donc une **redirection** vers un controller.

Si le *path* pour appeler *InscriptionController* est *"/inscription*",



alors, le retour de la méthode post de *WelcomeController* doit être "***redirect:/inscription***" et non plus "***integrated:userInscription***".



Testez que les valeurs introduites par l'utilisateur dans les champs du formulaire sont bien placées dans les variables d'instance de l'objet modèle de type *User* après que l'utilisateur a cliqué sur le bouton *send*. Par exemple, en affichant à la console (juste le temps du test, bien sûr !) les valeurs des variables d'instance de l'objet modèle de type *User* (placez cette instruction d'affichage à la console dans la méthode *post* de *InscriptionController* juste avant de faire la redirection).

Etape 8 – Service *HoobiesService*

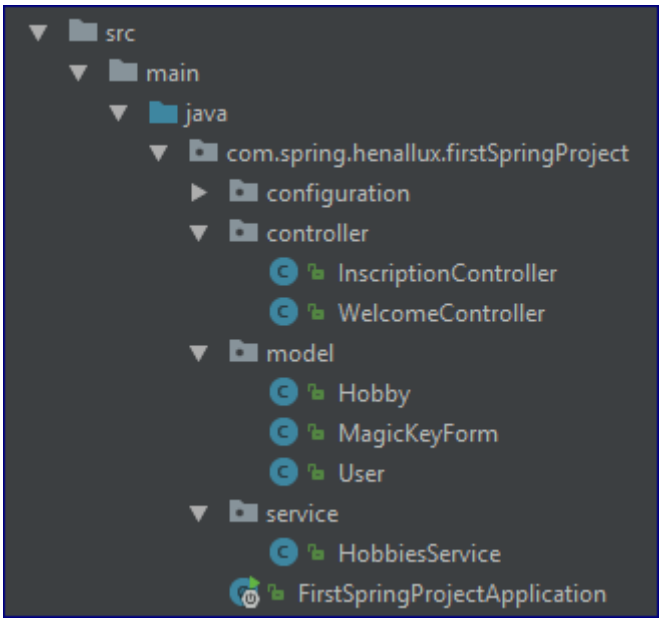
La liste des hobbies peut être remplie avec des valeurs fournies par un service (ultérieurement provenant d'une base de données).

Créez la classe *Hobby* dans le package *model*.

Cette classe contient les deux variables d'instance privées de type String : *id* et *name*.

Créez le package *service* dans *com.spring.henallux.firstSpringProject*.

Créez-y la classe *HobbiesService* qui contient une variable d'instance privée de type *ArrayList<Hobby>.* Prévoyez le getter et le setter publiques pour cette variable d'instance ainsi qu'au moins le constructeur sans argument. Garnissez cette liste dans le constructeur sans argument de *HobbiesServices* (ajoutez au moins 4 hobbies).



Inversion de contrôle et injection de dépendances

Pour rappel, le framework Spring supporte l'inversion de contrôle (le programmeur ne crée plus les objets) par injection de dépendance (création automatique par Spring d'objets appelés beans). Les classes annotées *@Service* permettent l'inversion de contrôle ; Spring créera un objet d'une telle classe automatiquement dès que nécessaire.

La classe *HobbiesService* doit être annotée *@Service*. Ceci permettra d'y avoir accès dans d'autres classes par injection de dépendance.

```
import org.springframework.stereotype.Service;

@Service
public class HobbiesService {
    private ArrayList<Hobby> hobbies;
```

Etape 9 – Accès à *HoobiesService* par injection

Récupération de la référence vers un bean créé par Spring

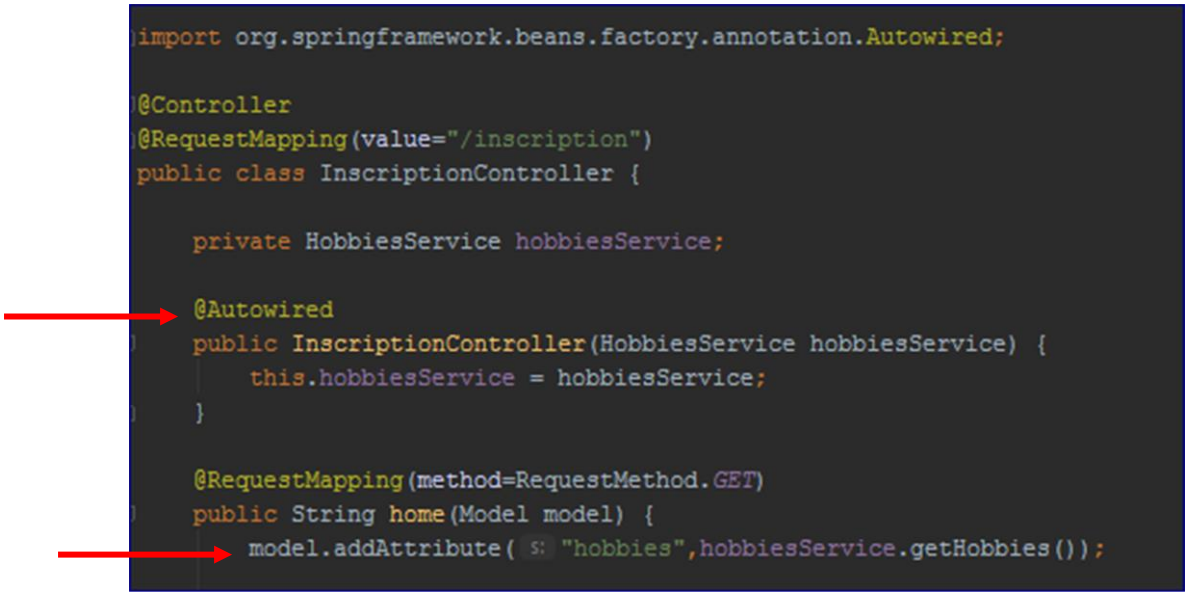
En programmation, il est possible de récupérer la référence d'un bean qui aurait été créé par inversion de contrôle par Spring.

L'annotation ***@Autowired*** permet de récupérer les références vers de tels beans.

Si l'annotation est placée devant un constructeur, les arguments du constructeur peuvent être récupérés par injection.

Dans *InscriptionController*, déclarez une variable d'instance de type *HobbiesService* et récupérez par injection de dépendance la référence vers l'objet de cette classe créé automatiquement par Spring (via @*Autowired* sur le constructeur - cf cours).

Ajoutez la liste des hobbies fournie par ce service comme attribut dans le dictionnaire *model*, afin de pouvoir avoir accès à la liste des hobbies dans la page *userInscription.jsp.*



```
import org.springframework.beans.factory.annotation.Autowired;

@Controller
@RequestMapping(value="/inscription")
public class InscriptionController {

    private HobbiesService hobbiesService;

    @Autowired
    public InscriptionController(HobbiesService hobbiesService) {
        this.hobbiesService = hobbiesService;
    }

    @RequestMapping(method=RequestMethod.GET)
    public String home(Model model) {
        model.addAttribute( s: "hobbies",hobbiesService.getHobbies());
```

Afin d'afficher les hobbies fournis par le service *HobbiesService*, supprimez les balises <form:option> du formulaire et remplassez-les par :

```
<form:options items="${hobbies}" itemValue="id" itemLabel="name" />
```

L'attribut *itemValue* précise la valeur réelle qui sera récupérée dans la variable d'instance de l'objet modèle.

Faites un clean + install puis redéployez l'application et testez-la.

Etape 10 – Validation du formulaire

Les champs du formulaire peuvent être validés automatiquement.

Validations à imposer au formulaire de la page *userInscription* :

Entre 4 et 15 caractères pour le nom

Age obligatoire et compris entre 1 et 12 ans

Ajoutez une dépendance dans le pom.xml :

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-validation</artifactId>
</dependency>
```

Faites un clean + install.

Annotez les variables d'instance de la classe modèle *User* que vous souhaitez valider. Utilisez *@Size* (avec les attributs *min* et *max*) pour les chaînes de caractères, *@NotNull, @Min et @Max* (avec l'attribut *value*) pour les nombres… *(*cf. Module 6 – Controller ⇨ Form Validation). Importez le package correspondant à ces annotations (javax.validation.constraints).

Dans la méthode *post* de *InscriptionController*, annoter l'attribut modèle de type *User* avec *@Valid*.

```
import javax.validation.Valid;
import org.springframework.validation.BindingResult;
```

```
@RequestMapping(value="/send", method=RequestMethod.POST)
public String getFormData(Model model,
                          @Valid @ModelAttribute(value="user") User user,
                          final BindingResult errors) {

    if (!errors.hasErrors())
    {
```

Déclarez un argument de type *BindingResult* dans la méthode *post*. Testez ce dernier. S'il n'y a pas d'erreur, la page *gift.jsp* est affichée. Cette page contient pour l'instant seulement un message de félicitation. S'il y a des erreurs, réaffichez la page *userInscription*, en prenant soin de replacer la liste des hobbies dans le dictionnaire *model* avant de réafficher la page.

Faites-en sorte que le message d'erreur éventuel relatif au nom soit affiché à côté du champ nom (via utilisation de <*form:errors*> cf. Module 6 – Controller ⇨ Form Validation – *Jsp Page*).

Faites de même pour le message d'erreur éventuel relatif à l'âge.