Understanding Form class to pass data

Presented by ARMS（THAILAND）Co., Ltd.

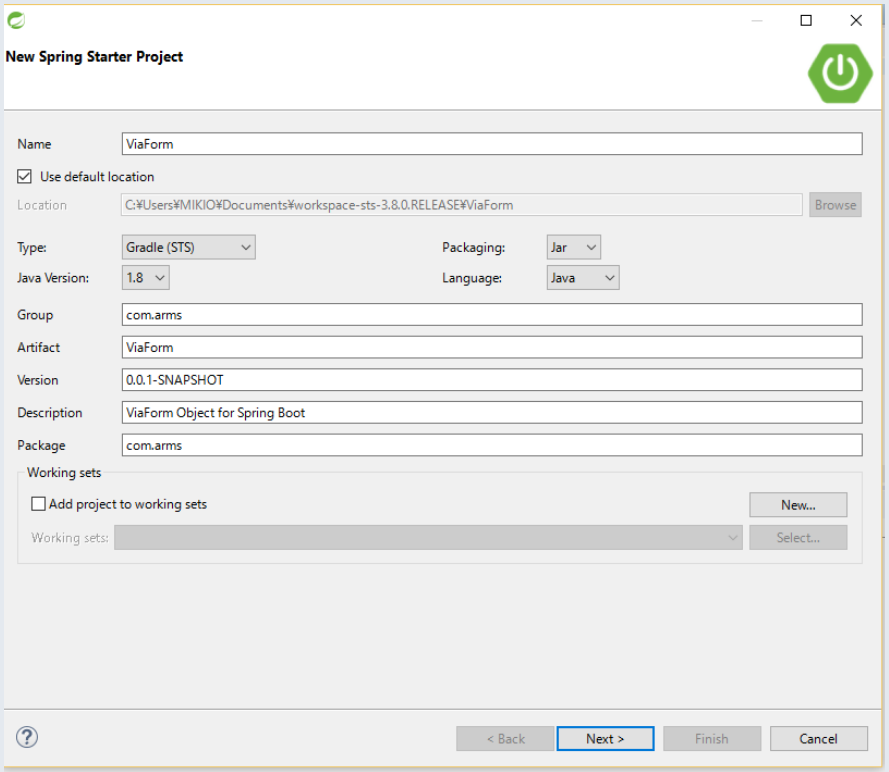
Index

Index

1. Start creating a project

1-1. Start Project

- Open STS and File – New – Spring Starter Project

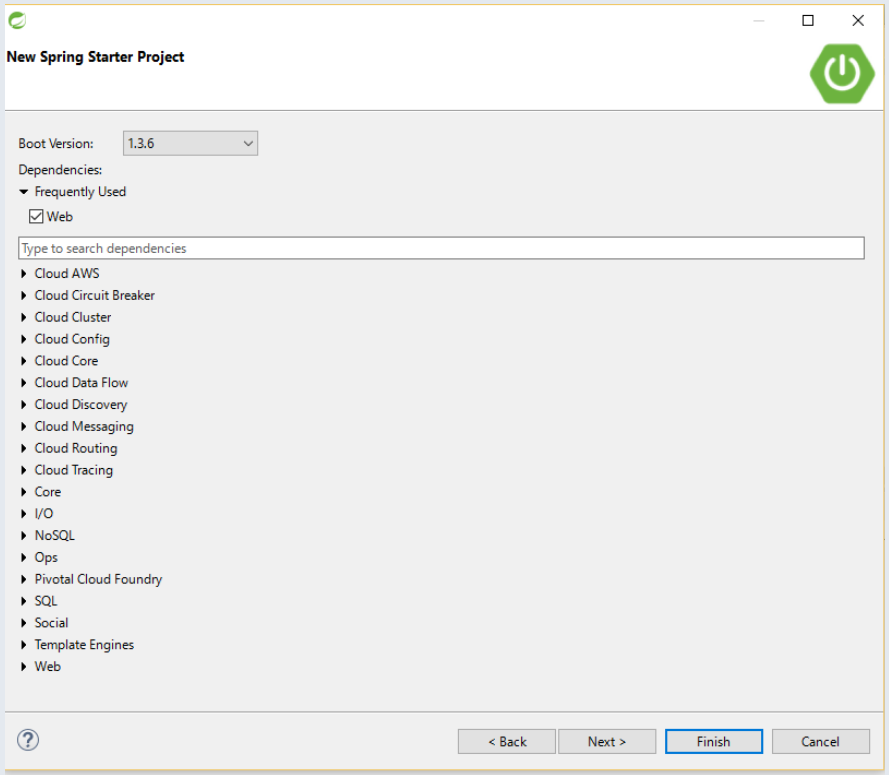


Key in necessary fields and click Next>

1. Start creating a project

1-1. Start Project

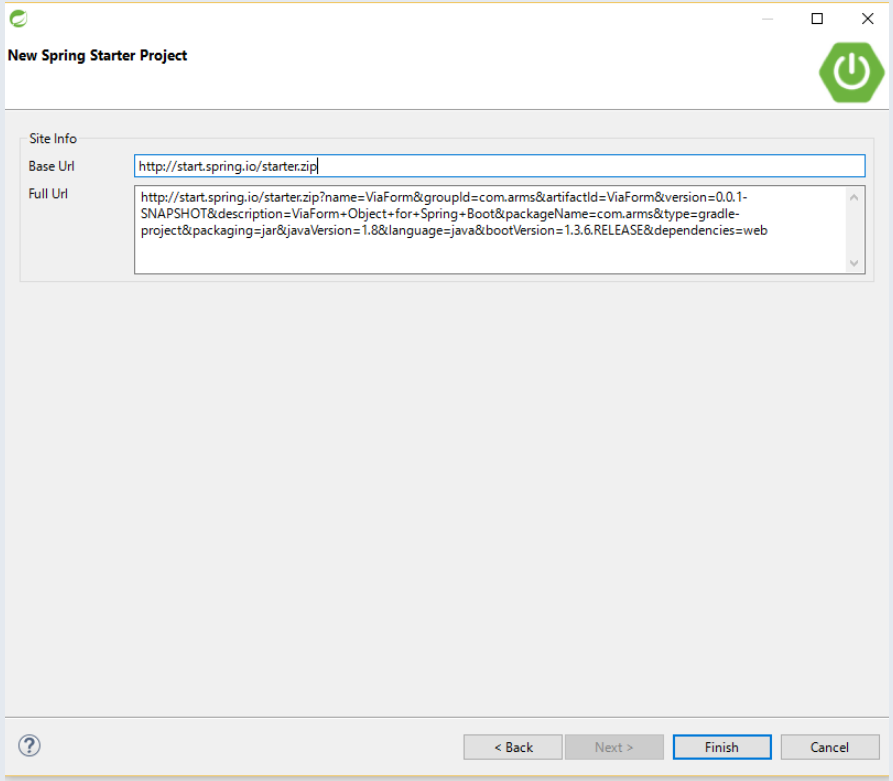
- Check Web and click Next>



1. Start creating a project

1-1. Start Project

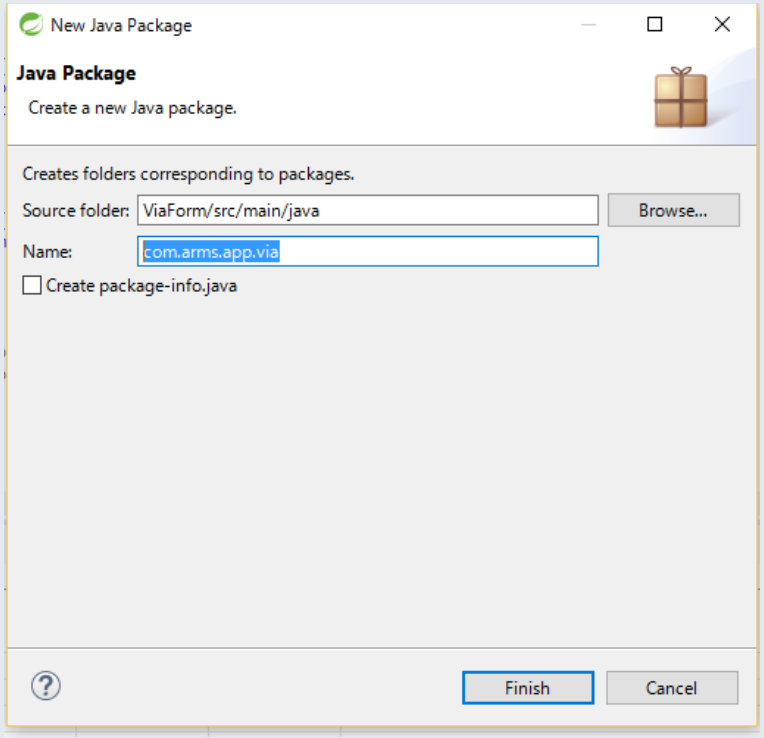
- Click Finish

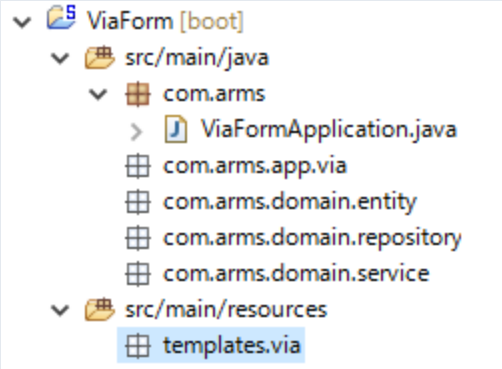


1. Start creating a project

1-1. Start Project

- Create packages as below.





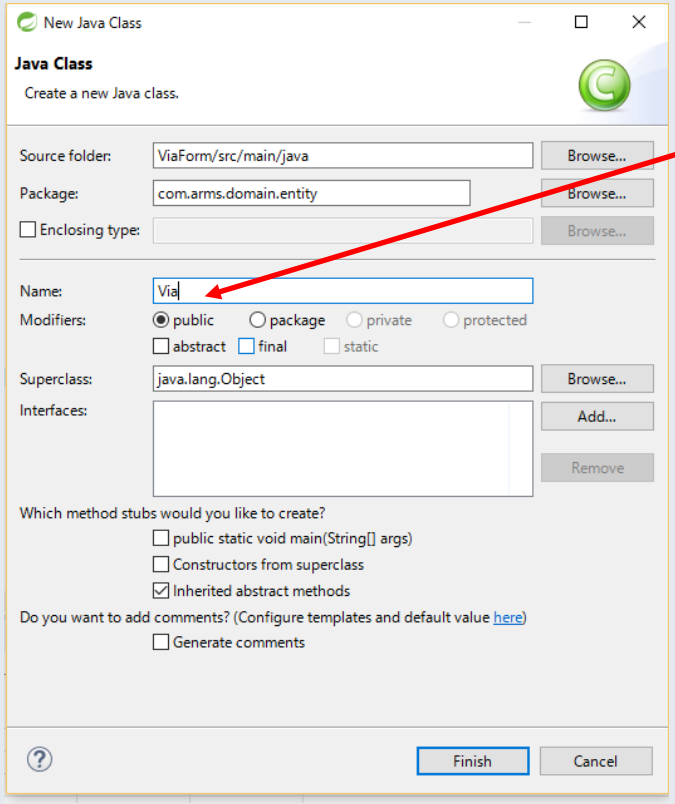
<u>src/main/java</u>
com.arms.app.via
com.arms.domain.entity
com.arms.domain.repository
com.arms.domain.service

<u>src/main/resources</u>
templates.via

2. Create Entity and Repository

2-1. Create entity class

- Right-click on com.arms.domain.entity New - Class



Name: Via

and click Finish

2. Create Entity and Repository

2-1. Create entity class

- Resolve dependencies to create entity class

Modify build.gralde as below.

```
dependencies {
        compile('org.springframework.boot:spring-boot-starter-data-jpa')
        compile('org.springframework.boot:spring-boot-starter-thymeleaf')
        compile('org.springframework.boot:spring-boot-starter-web')
        compile('net.sourceforge.nekohtml:nekohtml')
        runtime('org.hsqldb:hsqldb')
        testCompile('org.springframework.boot:spring-boot-starter-test')
}
```

Modify application.properties as below.

```
spring.thymeleaf.mode=LEGACYHTML5
spring.thymeleaf.cache=false
spring.thymeleaf.encoding=UTF-8
```

Right-click on your project, and Gradle(STS) – Refresh Dependencies

2. Create Entity and Repository

2-1. Create entity class

- Add the following code into Via.java

```
package com.arms.domain.entity;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;

@Entity
public class Via {

@Id @GeneratedValue
private Integer id;

private String name;

}
```

If you forget those dependencies to be resolved, you can't use the annotations in the code.

```
dependencies {
        compile('org.springframework.boot:spring-boot-starter-data-jpa')
        runtime( 'org.hsqldb:hsqldb' )
……..
}
```

2. Create Entity and Repository

2-1. Create entity class

- Since you have private fields in Via.java, create getter/setter for them.

```
.........................
private Integer id;

private String name;

}
```

Right-click on your code and display context menu.
Source – Generate getters and setters….

Eclipse(STS) has this useful function.

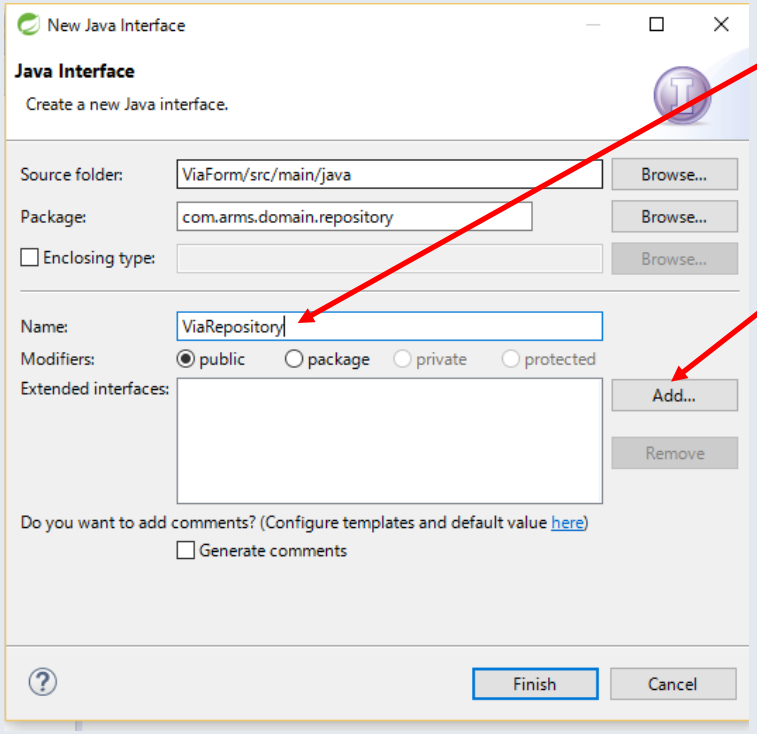
But in our case, let's use lombok to auto-generate getters and setters.
Add @Data annotation and import.

```
import lombok.Data;

@Entity
@Data
public class Via {
```

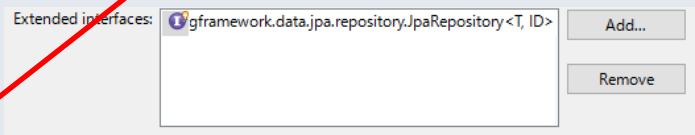
2. Create Entity and Repository

2-2. Create repository interface

- Right-click on com.arms.domain.repository, New- Interface



Key in ViaRepository in the Name, and press "Add" button to choose interface to be extended.



After adding the extended interface Press "Finish" button

2. Create Entity and Repository

2-2. Create repository interface

- Add the following code into ViaRepository

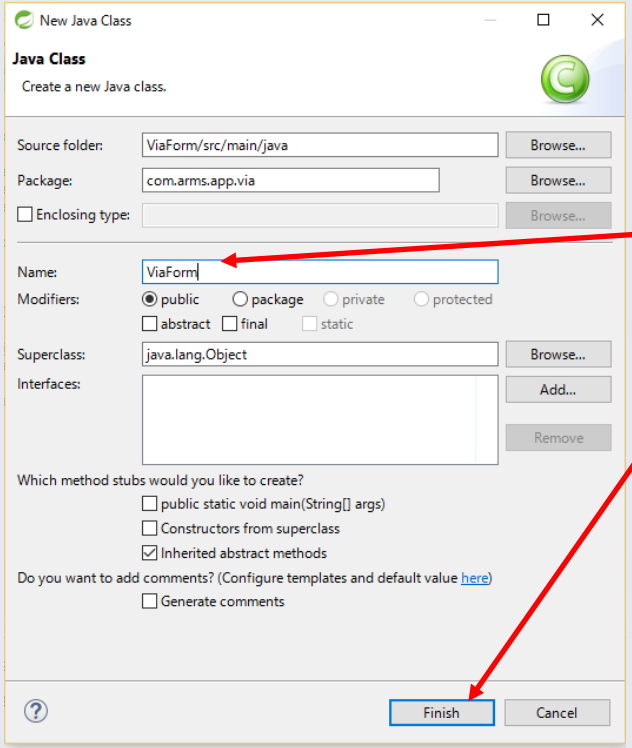
```
package com.arms.domain.repository;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import com.arms.domain.entity.Via;

@Repository
public interface ViaRepository extends JpaRepository<Via, Integer> {

}
```

Interface has to extend Repository and be typed to the domain class and its PK type.

3. Create Form

3-1. Create form object

- Create a form object to receive values from text. Right-click on com.arms.app.via New - Class



Key in ViaForm in Name, and Press "Finish" button.

3. Create Form

3-1. Create form object

- Add the following code into ViaForm.java

```
package com.arms.app.via;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@AllArgsConstructor
@NoArgsConstructor
public class ViaForm {

private Integer id;
private String name;
}
```

3. Create Form

3-1. Create form object

- @All and NoArgsConstructor annotations

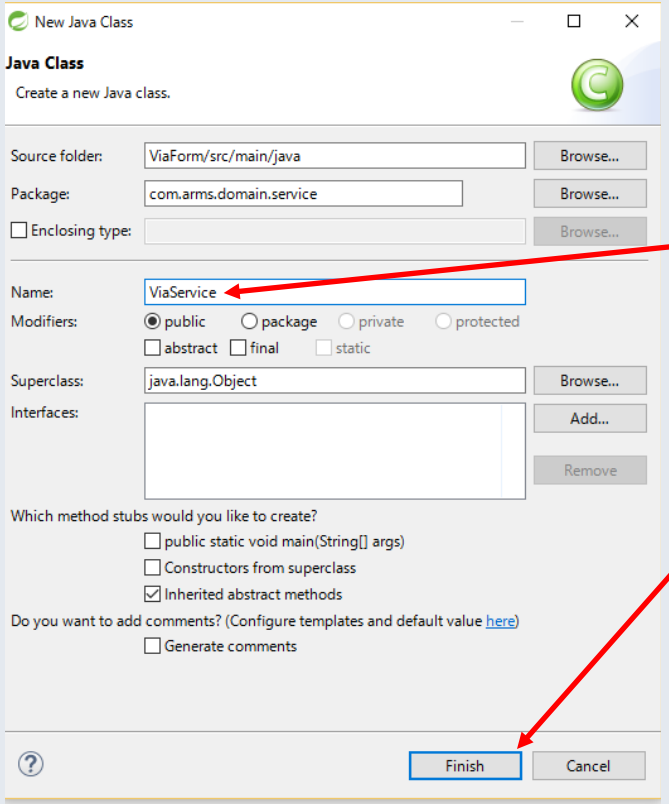
```
@AllArgsConstructor
@NoArgsConstructor
public class ViaForm {

private Integer id;
private String name;

        public ViaForm(Integer id, String name) {
                super();
                this.id = id;
                this.name = name;
        }

        public ViaForm() {
        }

}
```

You already have the above constructors.

4. Create Service and Controller

4-1. Create service

- Right-click on com.arms.domain.service. New - Class



Key in ViaService in Name, and press “Finish” button.

4. Create Service and Controller

4-1. Create service

- Add the following code into ViaService.java

```
package com.arms.domain.service;

import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import com.arms.app.via.ViaForm;
import com.arms.domain.entity.Via;
import com.arms.domain.repository.ViaRepository;

@Service
public class ViaService {

	@Autowired
	ViaRepository viaRepository;

	public List<Via> findAllVia(){
		return viaRepository.findAll();
	}

	public void save(ViaForm viaForm){
		Via via = new Via();
		via.setName(viaForm.getName());
		viaRepository.save(via);
	}
}
```

@Service annotation

@Autowired and access methods in Repository.

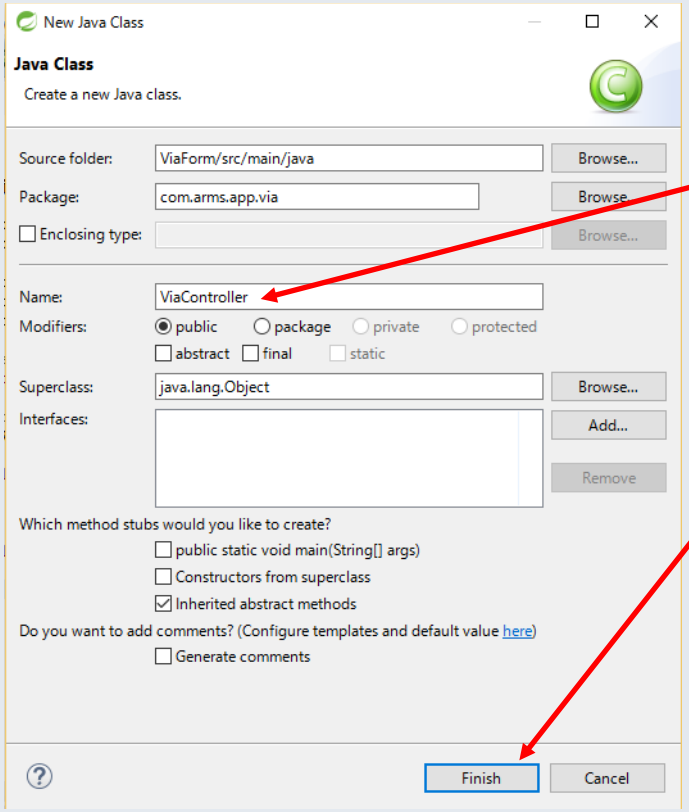
Get name value from viaForm and set it in via.

via.name = viaForm.getName();

4. Create Service and Controller

4-2. Create controller

- Right-click on com.arms.app.via New - Class



Key in ViaController in Name, and press “Finish” button

4. Create Service and Controller

4-2. Create controller

- Add the following code into ViaController.java

```
package com.arms.app.via;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

import com.arms.domain.entity.Via;
import com.arms.domain.service.ViaService;

@Controller
public class ViaController {

	@Autowired
	ViaService viaService;

		@RequestMapping("/")
		public String index(Model model){
			List<Via> viaList = viaService.findAllVia();
			model.addAttribute("viaList", viaList);
			return "via/index";
		}
```

Continue on next page

4. Create Service and Controller

4-2. Create controller

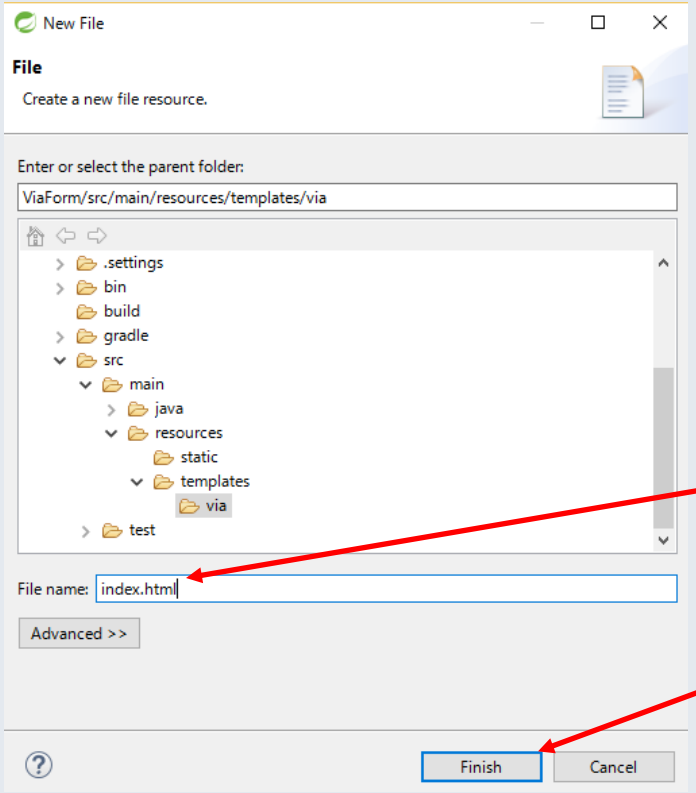
- Add the following code into ViaController.java

```
@RequestMapping(value = "/via", method = RequestMethod.GET)
public String via(Model model){
        model.addAttribute("viaForm", new ViaForm());
        return "via/via";
}

@RequestMapping(value = "/via", method = RequestMethod.POST)
public String create(@ModelAttribute ViaForm viaForm){
        viaService.save(viaForm);
        return "redirect:/via";
}


}
```

5. Create View

5-1. Create view

- Right-click on templates.via New - File



Key in index.html in File name and press “Finish” button.

5. Create View

5-1. Create view

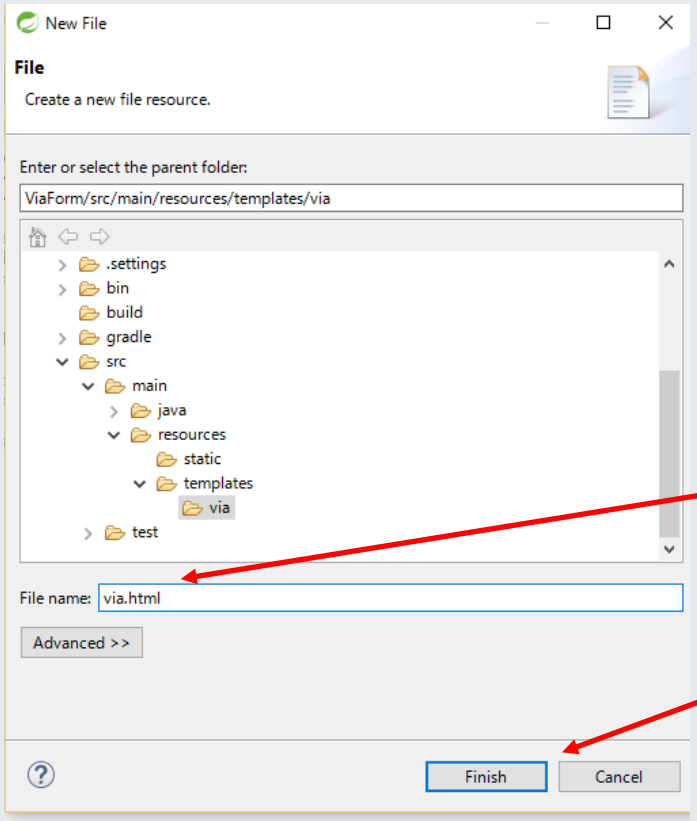
- Add the following code into index.html

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:th="http://www.thymeleaf.org">
<head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Input Form</title>
</head>

<body>
        <div th:each="via: ${viaList}">
                <p th:text="${via.name}">name</p>
        </div>
</body>

</html>
```

5. Create View

5-1. Create view

- Right-click on templates.via New - File



Key in via.html in File name and press “Finish” button.

5. Create View

5-1. Create view

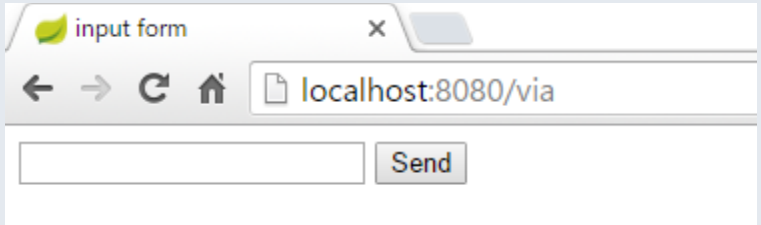
- Add the following code into via.html

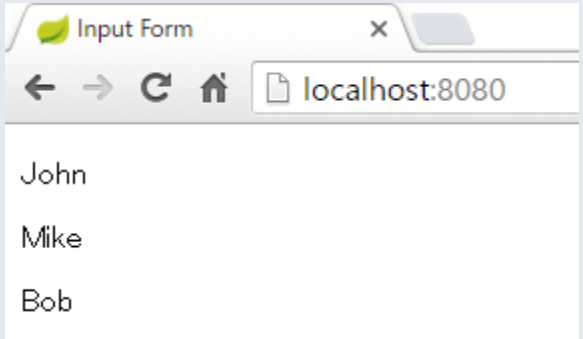
```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:th="http://www.thymeleaf.org">
<head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>input form</title>
</head>
<body>
        <form th:action="@{/via}" th:object="${viaForm}" method="POST">
                <input th:field="*{name}" id="name" type="text" name="name">
                <button type="submit">Send</button>
        </form>
</body>

</html>
```

5. Create View

5-1. Create view

- Start your project and access localhost:8080/via





Send a few records and access localhost:8080

6. Code Explanation

6-1. form object

- Let's see how form object is used.

access via.html

```
@RequestMapping(value = "/via", method = RequestMethod.GET)
public String via(Model model){
        model.addAttribute("viaForm", new ViaForm());
        return "via/via";
}
```

viaForm object goes to via.html

```
<form th:action="@{/via}" th:object="${viaForm}" method="POST">
        <input th:field="*{name}" id="name" type="text" name="name">
        <button type="submit">Send</button>
</form>
```

viaForm.name is received as args

```
@RequestMapping(value = "/via", method = RequestMethod.POST)
public String create(@ModelAttribute ViaForm viaForm){
        viaService.save(viaForm);
        return "redirect:/via";
}
```

Saved by save(viaForm) method

6. Code Explanation

6-1. form object

- Thymeleaf syntax

via.html

```
<form th:action="@{/via}" th:object="${viaForm}" method="POST">
        <input th:field="*{name}" id="name" type="text" name="name">
        <button type="submit">Send</button>
</form>
```

With th:object, it can abbreviate model object name part

```
<form th:action="@{/via}" th:object="${viaForm}" method="POST">
        <input th:field="${viaForm.name}" id="name" type="text" name="name">
        <button type="submit">Send</button>
</form>
```

index.html

```
<div th:each="via: ${viaList}">
        <p th:text="${via.name}">name</p>
</div>
```

This has the same result, but useful when there are many fields

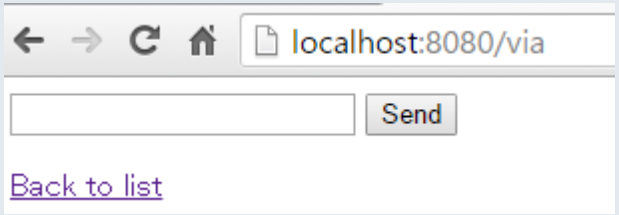
```
<div th:each="via: ${viaList}" th:object=${via}>
        <p th:text="*{name}">name</p>
</div>
```

7. Practice

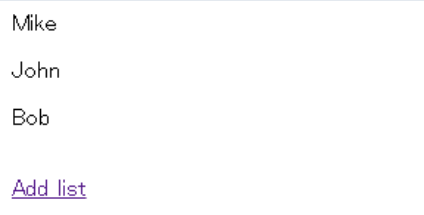
7-1. Create link button

- Create link button on via.html and index.html

via.html : Back to list button to go to index.html



index.html : Add list button to go to via.html



ARMS

Your Idea Leads Your Ideals

homepage: http://arms-asia.com/
facebook: https://www.facebook.com/arms.asia?fref=ts