

Session 4

Dependency injection and controllers

Dependency injection is how Spring wires objects together. Controllers map URLs to methods and hand data to views. The session ends with a Spring Form for student admission, the form that Sessions 5 and 6 validate and persist.

Everything here goes into the `student-admission` project from Session 3. Q17 and Q18 are console programs in the `ioc` package; Q19 and Q20 add controllers and JSP views to the web application.

Objectives

✗ Do not copy. Read for understanding and the viva

- Complete questions 17 to 20 of the manual: dependency injection and controllers
- Prepare the deliverable before the lab and finish it during the session
- Be ready to explain every step in the viva

Questions Covered

✗ Do not copy. Read for understanding and the viva

Question	Requirement	Status
Q17	Write a class and implement it using dependency injection	Complete
Q18	Write the service interface with the getRandom() method, define 3 courses in an array,...	Complete
Q19	Write a programme using Spring Framework to create a controller and display response...	Complete
Q20	Create a Form to capture Student Admission information (make usual assumptions about...	Complete

Preparation

✖ Do not copy. Read for understanding and the viva

- Constructor injection with `@Autowired` is the form to show; be able to contrast it with setter injection.
- Decide the Student Admission attributes now (name, date of birth, gender, programme dropdown, address textarea, hostel required true/false, courses checkboxes) because Sessions 5, 6 and 7 reuse them.
- Spring form tags need the `spring-webmvc` dependency and the form taglib declaration on the JSP.

Question 17

Problem Statement

■ Write in lab record

Write a class and implement it using dependency injection.

Solution

■ Write in lab record

Steps

1. Create `DiApp.java` in `com.ignou.lab.admission.ioc`. It holds one interface (`FeeCalculator`), one implementation (`FlatFeeCalculator`) and the class that depends on it (`AdmissionDesk`), plus `main`.
2. Run it: Run As → Java Application, or `mvn -q compile exec:java -Dexec.mainClass=com.ignou.lab.admission.ioc.DiApp`.
3. To see the injection fail, comment out `@Component` on `FlatFeeCalculator` and run again: `UnsatisfiedDependencyException ... No qualifying bean of type FeeCalculator`.

Program

ioc/DiApp.java

```
1 package com.ignou.lab.admission.ioc;
2
3 import org.springframework.context.annotation.AnnotationConfigApplicationCon
4 import org.springframework.stereotype.Component;
5
6 // ponytail: three small classes in one file so the whole DI example is one
7
8 /** What AdmissionDesk depends on. It never knows which implementation it ge
9 interface FeeCalculator {
10     int feeFor(String programme);
11 }
12
13 @Component
14 class FlatFeeCalculator implements FeeCalculator {
15     @Override
16     public int feeFor(String programme) {
17         return programme.equals("MCA") ? 12000 : 8000;
18     }
19 }
20
21 /** Constructor injection: Spring passes the FeeCalculator in; the field is
22 @Component
23 class AdmissionDesk {
24     private final FeeCalculator fees;
25
26     AdmissionDesk(FeeCalculator fees) {
27         this.fees = fees;
28     }
29
30     String quote(String programme) {
31         return programme + " fee per semester: Rs " + fees.feeFor(programme)
32     }
33 }
34
35 public class DiApp {
36     public static void main(String[] args) {
37         try (var ctx = new AnnotationConfigApplicationContext(FlatFeeCalcula
38             AdmissionDesk desk = ctx.getBean(AdmissionDesk.class);
39             System.out.println(desk.quote("MCA"));
40             System.out.println(desk.quote("BCA"));
41         }
42     }
43 }
```

Output

Expected console output, checked by reading, not executed:

```
1 MCA fee per semester: Rs 12000
2 BCA fee per semester: Rs 8000
```

Explanation

- `AdmissionDesk` declares what it needs (a `FeeCalculator`) as a constructor parameter. It never creates one. That is dependency injection: the dependency is pushed in from outside.
- Spring sees a single constructor and calls it with the only `FeeCalculator` bean it knows, `FlatFeeCalculator`. Since Spring 4.3 a single constructor needs no `@Autowired`; add it when there is more than one.
- Constructor injection makes the field `final`, so the object is complete the moment it exists. Setter injection (`@Autowired` on `setFees`) allows an incomplete object and is used for optional dependencies.
- A unit test can call `new AdmissionDesk(p → 100)` with a lambda; no Spring needed. That is the practical payoff of injecting an interface.

Question 18

Problem Statement

■ Write in lab record

Write the service interface with the `getRandom()` method, define 3 courses in an array, and inject using dependency injection into the teacher Interface (exercise no 4 in session 3). Test the application and verify the retrieving of random courses.

Solution

■ Write in lab record

Steps

1. In the `ioc` package add `CourseService.java` and `RandomCourseService.java`.

2. Replace `JavaTeacher.java` with the version below: a constructor takes `CourseService` and `getFavouriteCourse()` delegates to it. `Teacher.java` from Session 3 is unchanged.
3. Replace `TeacherApp.java` so the context knows both classes and prints five calls.
4. Run `TeacherApp` three times; the sequence of courses differs each run.

Program

- Lab record: every tab is one file of the answer. Write all of them.

CourseService.java

ioc/CourseService.java

```
1 package com.ignou.lab.admission.ioc;
2
3 public interface CourseService {
4     String getRandom();
5 }
```

Output

Expected console output (one run; yours will differ because the choice is random), checked by reading, not executed:

```
1 1. Dr. Rao prefers MCS-219 Object Oriented Analysis and Design
2 2. Dr. Rao prefers MCS-220 Web Technologies
3 3. Dr. Rao prefers MCS-220 Web Technologies
4 4. Dr. Rao prefers MCS-218 Data Communication and Computer Networks
5 5. Dr. Rao prefers MCS-219 Object Oriented Analysis and Design
```

Verification: over five calls only the three array entries ever appear, and at least two different ones show up in almost every run.

Explanation

- `RandomCourseService` is a `@Service` (a `@Component` with a clearer name). The three courses sit in a `static final String[]`; `ThreadLocalRandom.current().nextInt(3)` picks an index.
- `JavaTeacher` now depends on the `CourseService` interface, injected through its constructor. Session 3's version hard-coded the answer; this one asks a collaborator.

- The container wires the graph: it must build `RandomCourseService` first, then pass it to `JavaTeacher`'s constructor. Order is worked out from the constructor signatures, not from the order the classes are listed.
- Because `JavaTeacher` is in the web application's scanned package, the same wiring happens inside Tomcat, which Q19 uses.

Question 19

Problem Statement

■ Write in lab record

Write a programme using Spring Framework to create a controller and display response in view using `@GetMapping()` annotation.

Solution

■ Write in lab record

Steps

1. Add `TeacherController.java` to `com.ignou.lab.admission.web` and `teacher.jsp` to `WEB-INF/views`.
2. Rebuild, redeploy, open <http://localhost:8080/student-admission/teacher>.
3. Press F5 several times; the course changes.

Program

- Lab record: every tab is one file of the answer. Write all of them.

`TeacherController.java`

web/TeacherController.java

```
1 package com.ignou.lab.admission.web;
2
3 import org.springframework.stereotype.Controller;
4 import org.springframework.ui.Model;
5 import org.springframework.web.bind.annotation.GetMapping;
6 import com.ignou.lab.admission.ioc.Teacher;
7
8 @Controller
9 public class TeacherController {
10
11     private final Teacher teacher; // JavaTeacher, found by the component sc
12
13     public TeacherController(Teacher teacher) {
14         this.teacher = teacher;
15     }
16
17     @GetMapping("/teacher")
18     public String show(Model model) {
19         model.addAttribute("teacherName", teacher.getName());
20         model.addAttribute("course", teacher.getFavouriteCourse());
21         return "teacher";
22     }
23 }
```

Output

Expected browser result, checked by reading, not executed:

```
1 Teacher of the day
2 Dr. Rao will teach MCS-220 Web Technologies this session.
3 Refresh for another course
```

Explanation

- `@Controller` makes the class a bean whose methods can be mapped to URLs.
`@GetMapping("/teacher")` binds HTTP GET on `/teacher` to `show()`.
- The `Model` parameter is a map that the view can read. `teacher.jsp` reads `teacherName` and `course` with `${...}` expressions.

- The returned string `"teacher"` is a view name; `InternalResourceViewResolver` (Session 3) turns it into `/WEB-INF/views/teacher.jsp` and forwards the request.
- The controller receives the `Teacher` bean through its constructor, exactly like `TeacherApp` did; a web request is just another caller of the same object graph.

Question 20

Problem Statement

■ Write in lab record

Create a Form to capture Student Admission information (make usual assumptions about the attributes) using Spring Form tags (must use Text, textarea, dropdown, date picker, true/false and checkbox) and write a controller using `@PostMapping()` to display the form information.

Solution

■ Write in lab record

Assumed attributes: name, email and mobile (text), date of birth (date picker), address (textarea), programme (dropdown), hostel required (true/false radio) and courses (checkboxes).

Steps

1. Add `AdmissionForm.java` and `AdmissionController.java` to `com.ignou.lab.admission.web`.
2. Add `admission-form.jsp` and `admission-result.jsp` to `WEB-INF/views`. The form page declares the Spring form taglib with prefix `form`.
3. Rebuild, redeploy, open <http://localhost:8080/student-admission/admission>, fill every field, submit.
4. View page source of the form before submitting: every `form:` tag became a plain HTML element with `id` and `name` equal to the path.

Program

■ Lab record: every tab is one file of the answer. Write all of them.

AdmissionForm.java

web/AdmissionForm.java

```
1 package com.ignou.lab.admission.web;
2
3 import java.time.LocalDate;
4 import java.util.ArrayList;
5 import java.util.List;
6 import org.springframework.format.annotation.DateTimeFormat;
7
8 /** Form-backing object for the Student Admission form. */
9 public class AdmissionForm {
10
11     private String name; // form:input
12     private String email; // form:input type="email"
13     private String mobile; // form:input
14     @DateTimeFormat(iso = DateTimeFormat.ISO.DATE)
15     private LocalDate dob; // form:input type="date" (br
16     private String address; // form:textarea
17     private String programme; // form:select
18     private Boolean hostelRequired; // form:radiobutton true / fa
19     private List<String> courses = new ArrayList<>(); // form:checkboxes
20
21     public String getName() { return name; }
22     public void setName(String name) { this.name = name; }
23     public String getEmail() { return email; }
24     public void setEmail(String email) { this.email = email; }
25     public String getMobile() { return mobile; }
26     public void setMobile(String mobile) { this.mobile = mobile; }
27     public LocalDate getDob() { return dob; }
28     public void setDob(LocalDate dob) { this.dob = dob; }
29     public String getAddress() { return address; }
30     public void setAddress(String address) { this.address = address; }
31     public String getProgramme() { return programme; }
32     public void setProgramme(String programme) { this.programme = programme; }
33     public Boolean getHostelRequired() { return hostelRequired; }
34     public void setHostelRequired(Boolean hostelRequired) { this.hostelRequi
35     public List<String> getCourses() { return courses; }
36     public void setCourses(List<String> courses) { this.courses = courses; }
37 }
```

Output

Expected result after submitting the form, checked by reading, not executed:

```
1 Application received
2 Name           Asha Verma
3 Email          asha@example.com
4 Mobile         9876543210
5 Date of birth  2003-04-12
6 Address        Sector 4, Rohini, Delhi
7 Programme      MCA
8 Hostel required Yes
9 Courses        MCS-218 Data Communication and Computer Networks
10               MCS-220 Web Technologies
11 Another application
```

The generated HTML for the radio buttons is `input type="radio" name="hostelRequired" value="true"` and `value="false"` ; the date field is `input type="date" name="dob"` .

Explanation

- `AdmissionForm` is the form-backing object. `showForm` puts an empty one in the model under the name `admission` ; `form:form modelAttribute="admission"` binds every `form:` tag to a property of it by `path` .
- Tag to element: `form:input` (text; `type="email"` and `type="date"` pass through as HTML5 input types, so the browser shows its own date picker), `form:textarea` , `form:select` with `form:options` , two `form:radiobutton` with values `true` and `false` bound to a `Boolean` , `form:checkboxes` over a list bound to `List<String>` .
- `@DateTimeFormat(iso = DATE)` tells Spring's conversion service to parse `2003-04-12` into a `LocalDate` ; without it the post fails with a type mismatch.
- `@PostMapping("/admission")` receives the same URL by POST. `@ModelAttribute("admission")` makes Spring create an `AdmissionForm` , copy every request parameter into the matching property (data binding) and add it to the model, so `admission-result.jsp` reads `${admission.name}` .
- `@ModelAttribute` on the `programmes()` and `courseList()` methods runs before every handler in the controller, so both GET and the POST re-render see the option lists.

Viva Questions

✗ Do not copy. Read for understanding and the viva

- **Q:** What is dependency injection? **A:** An object receives the objects it needs from outside (constructor or setter) instead of creating them; the container does the passing.
- **Q:** Constructor or setter injection, which and why? **A:** Constructor: mandatory dependencies, immutable fields, object valid at creation. Setter: optional dependencies or circular references.
- **Q:** What is the difference between `@Component`, `@Service`, `@Repository` and `@Controller`? **A:** All are components; the names document the layer. `@Repository` also translates persistence exceptions, `@Controller` enables request mapping.
- **Q:** What happens if two beans implement `Teacher`? **A:** `NoUniqueBeanDefinitionException` unless one is `@Primary` or the injection point uses `@Qualifier`.
- **Q:** What does `@ModelAttribute` do on a method parameter? **A:** Creates or looks up the object, binds request parameters to its properties and adds it to the model.
- **Q:** Why does `form:form` need `modelAttribute`? **A:** The form tags read initial values from that object and derive `name` attributes from `path`, so binding on POST is by the same names.
- **Q:** How does a `Boolean` get a true/false value from radio buttons? **A:** Both radios share the name `hostelRequired`; the chosen one sends `true` or `false`, which Spring converts to `Boolean`.

Common Mistakes

✗ Do not copy. Read for understanding and the viva

- Injecting a concrete class (`JavaTeacher`) instead of the interface, which defeats the point of Q18.
- Forgetting the form taglib line; the JSP then prints `form:input` literally.
- Naming the model attribute `student` in one method and `admission` in the other; the POST binding then creates a new empty object.
- Leaving out `@DateTimeFormat` on the `LocalDate` field and getting `Failed to convert property value` on submit.
- Checkboxes bound to a `String` instead of a `List<String>`; only the last ticked box survives.

Session Summary

■ Write in lab record

- Question 17: `DiApp` with `AdmissionDesk` receiving a `FeeCalculator` through constructor injection

- Question 18: `CourseService.getRandom()` over a three-course array, injected into `JavaTeacher`, verified with five console calls
- Question 19: `TeacherController` with `@GetMapping("/teacher")` rendering `teacher.jsp`
- Question 20: Student Admission form with Spring form tags (text, textarea, select, date, true/false radios, checkboxes) and `@PostMapping` showing the posted data

[✎ Edit this page](#)

[< Previous](#)
Session 3

Next >
Session 5