Spring Auditing Fields

🧑‍🏫 Lesson: Spring Auditing Fields with Real Scenario

🔍 What is Spring Auditing?

Spring Data JPA provides **auditing capabilities** to automatically capture and persist information like:

- `createdDate` – when the record was created
- `createdBy` – who created the record
- `lastModifiedDate` – when the record was last updated
- `lastModifiedBy` – who updated the record

📘 Common Use Case

Scenario: An Employee Management System
We want to store who created or last modified employee records and when.

🛠️ Step-by-Step Implementation

1. Add Required Dependencies

If using **Maven**:

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

```

---

2. Enable JPA Auditing

In your main Spring Boot application or configuration class:

```
@SpringBootApplication
@EnableJpaAuditing(auditorAwareRef = "auditorProvider")
public class EmployeeApp {
    public static void main(String[] args) {
        SpringApplication.run(EmployeeApp.class, args);
    }
}

```

---

3. Create an AuditorAware Bean

```
@Component
```

```
public class AuditorAwareImpl implements AuditorAware<String> {

    @Override
    public Optional<String> getCurrentAuditor() {
        // In real apps, fetch from Spring Security context
        return Optional.of("adminUser"); // hardcoded for demo
    }
}

```

---

4. Define Base Auditing Entity

```
@MappedSuperclass
@EntityListeners(AuditingEntityListener.class)
public abstract class Auditable {

    @CreatedBy
    @Column(updatable = false)
    protected String createdBy;

    @CreatedDate
    @Column(updatable = false)
    protected LocalDateTime createdDate;

    @LastModifiedBy
    protected String lastModifiedBy;

    @LastModifiedDate
    protected LocalDateTime lastModifiedDate;

    // Getters and setters
}

```

---

5. Create an Entity Using Auditable

```
@Entity
public class Employee extends Auditable {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;
    private String role;

    // Getters and setters
}

```

---

6. Repository and Service Layer

```
public interface EmployeeRepository extends JpaRepository<Employee, Long> {
}

```

```
@Service
public class EmployeeService {

    @Autowired
    private EmployeeRepository repository;

    public Employee createEmployee(String name, String role) {
        Employee emp = new Employee();
        emp.setName(name);
        emp.setRole(role);
        return repository.save(emp);
    }

    public Employee updateEmployee(Long id, String role) {
        Employee emp = repository.findById(id).orElseThrow();
        emp.setRole(role);
        return repository.save(emp);
    }
}

```

🧪 Example Usage

```
@RestController
@RequestMapping("/employees")
public class EmployeeController {

    @Autowired
    private EmployeeService service;

    @PostMapping
    public ResponseEntity<Employee> create(@RequestBody Employee e) {
        return ResponseEntity.ok(service.createEmployee(e.getName(), e.getRole()));
    }

    @PutMapping("/{id}")
    public ResponseEntity<Employee> update(@PathVariable Long id, @RequestBody Employee e) {
        return ResponseEntity.ok(service.updateEmployee(id, e.getRole()));
    }
}

```

🔎 Database Output (Example)

ID	Name	Role	createdBy	createdDate	lastModifiedBy	lastModifiedDate
1	Alice	Developer	adminUser	2025-04-29	adminUser	2025-04-29

				09:15:00		10:00:00

✅ Benefits

- No manual tracking of `createdBy`, `updatedBy`, etc.
- Automatically integrates with Spring Security in real apps.
- Clean and reusable auditing logic.

Would you like a downloadable project template for this implementation?