Auditing in Spring Data JPA

Starting from this project's own real QuestionIngestService's hand-written LocalDateTime.now() pattern, this lesson covers @CreatedDate/@LastModifiedDate, wiring @EntityListeners(AuditingEntityListener.class) + @EnableJpaAuditing together, @CreatedBy/@LastModifiedBy, AuditorAware<T>, and sharing audit fields with @MappedSuperclass. The 8th lesson in the Spring Data JPA category.

Intermediate 25 min

Last updated: August 24, 2026
Read online: https://www.learnforgex.com/en/topics/jpa-auditing

---

This project's own `QuestionIngestService` builds a new `Question` with `.createdAt(LocalDateTime.now()).updatedAt(LocalDateTime.now())`, written by hand, right there in the service method. It works — but it's exactly the kind of repetitive, easy-to-forget code Spring Data JPA usually removes elsewhere. This lesson covers the tool built specifically for it: auditing.

The Problem: Setting createdAt/updatedAt by Hand

Every place that creates or updates an audited entity has to remember to set its timestamp correctly, every single time.

```
import java.time.LocalDateTime;

// This is the SAME pattern as this project's own real
// QuestionIngestService -- it builds a Question with
// .createdAt(LocalDateTime.now()).updatedAt(LocalDateTime.now()), by hand,
// right there in the service method. Nothing is technically wrong with
// this, but it's a pattern that has to be repeated correctly in EVERY
// place that creates or updates ANY audited entity -- forget it once, in
// one service method, and that row's timestamp is silently wrong.
class ManualTimestampProblemExample {

    record Question(String text, LocalDateTime createdAt, LocalDateTime updatedAt) {
    }

    static Question createQuestion(String text) {
        LocalDateTime now = LocalDateTime.now();
        return new Question(text, now, now); // set by hand, exactly like this project's
QuestionIngestService
    }

    static Question updateQuestion(Question existing, String newText) {
        // Every single update site also needs to remember this line --
        // and remember it EVERY time, not just once.
        return new Question(newText, existing.createdAt(), LocalDateTime.now());
    }
}
```

`createQuestion(...)` and `updateQuestion(...)` both need their own `LocalDateTime.now()` line — exactly this project's own `QuestionIngestService` pattern. Nothing here is technically broken, but the moment a second service method (or a third, or a tenth) needs to create or update the same kind of entity, that same line has to be remembered and repeated correctly, every time, in every place — forgetting it once leaves a row with a silently wrong timestamp.

@CreatedDate and @LastModifiedDate

Two annotations replace that manual line entirely, once wired up.

```
import jakarta.persistence.Entity;
import jakarta.persistence.EntityListeners;
import jakarta.persistence.Id;
import org.springframework.data.annotation.CreatedDate;
import org.springframework.data.annotation.LastModifiedDate;
import org.springframework.data.jpa.domain.support.AuditingEntityListener;

import java.time.LocalDateTime;

// @EntityListeners(AuditingEntityListener.class) is what actually makes
// @CreatedDate/@LastModifiedDate below do anything -- it registers a
// listener that runs automatically on this entity's own lifecycle events
// (right before the first INSERT, and right before every UPDATE), instead
// of any application code needing to set these fields itself.
@Entity
@EntityListeners(AuditingEntityListener.class)
class AuditedQuestionExample {

    @Id
    private Long id;

    private String text;

    // Populated automatically, exactly once, the moment this entity is
    // first persisted -- never touched again on later updates.
    @CreatedDate
    private LocalDateTime createdAt;

    // Populated automatically on the INITIAL insert, and then re-populated
    // on every single update after that -- this is the field
    // ManualTimestampProblemExample had to remember to update by hand,
    // every time, in every place.
    @LastModifiedDate
    private LocalDateTime updatedAt;
}
```

`@CreatedDate` is populated automatically, exactly once, the moment an entity is first persisted — never touched again afterward. `@LastModifiedDate` is populated on that same initial insert, and then re-populated automatically on every subsequent update — this is the field `ManualTimestampProblemExample` had to remember to update by hand, in every single place that touched it.

Wiring It Up: @EntityListeners and @EnableJpaAuditing

Two pieces have to be in place before `@CreatedDate` / `@LastModifiedDate` actually do anything — neither one alone is enough.

```
import org.springframework.context.annotation.Configuration;
import org.springframework.data.jpa.repository.config.EnableJpaAuditing;

// @EntityListeners alone isn't quite enough -- @EnableJpaAuditing, on a
// @Configuration class, is what turns Spring Data JPA's auditing
// infrastructure on for the application as a whole. Without it,
// @CreatedDate/@LastModifiedDate fields are simply never populated --
// silently left null, with no error to point at the missing piece.
@Configuration
@EnableJpaAuditing
class JpaAuditingConfig {
}

// With both pieces in place -- @EntityListeners on the entity, and
// @EnableJpaAuditing on a configuration class -- saving an
// AuditedQuestionExample no longer needs anything like
// ManualTimestampProblemExample's "LocalDateTime.now()" line anywhere:
//
//   AuditedQuestionExample q = new AuditedQuestionExample();
//   repository.save(q); // createdAt AND updatedAt are populated automatically
```

`@EntityListeners(AuditingEntityListener.class)`, on the entity itself, registers a listener that runs automatically on that entity's lifecycle events (right before the first insert, and right before every update). `@EnableJpaAuditing`, on a `@Configuration` class, turns Spring Data JPA's auditing infrastructure on for the application as a whole. Missing either one means `@CreatedDate` / `@LastModifiedDate` are simply never populated — silently left `null`, with no error pointing at what's missing.

Recording Who: @CreatedBy and @LastModifiedBy

The same mechanism extends to WHO made a change, not just WHEN.

```
import jakarta.persistence.Entity;
import jakarta.persistence.EntityListeners;
import jakarta.persistence.Id;
import org.springframework.data.annotation.CreatedBy;
import org.springframework.data.annotation.LastModifiedBy;
import org.springframework.data.domain.AuditorAware;
import org.springframework.data.jpa.domain.support.AuditingEntityListener;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import java.util.Optional;

// @CreatedBy/@LastModifiedBy work exactly like @CreatedDate/@LastModifiedDate
// -- same listener, same lifecycle timing -- but capture WHO made the
// change instead of WHEN. This project's real Question entity already has
// a "reviewedBy" column (set manually, by an admin doing a DB UPDATE, per
// this project's own review workflow) -- @CreatedBy/@LastModifiedBy solve
// a related but different problem: recording who created/last touched the
// row itself, automatically, not a separate manual review action.
@Entity
@EntityListeners(AuditingEntityListener.class)
class AuditedByQuestionExample {

    @Id
    private Long id;

    @CreatedBy
    private String createdBy;

    @LastModifiedBy
    private String lastModifiedBy;
}

// AuditorAware<T> is where "who" actually comes from -- Spring Data JPA
// has no idea who the current user is on its own; this bean is what
// supplies that answer, called automatically every time an audited entity
// is saved.
@Configuration
class AuditorAwareConfig {

    @Bean
    AuditorAware<String> auditorProvider() {
        // A real application would read this from Spring Security's
        // SecurityContextHolder (the currently authenticated user's name);
        // returning a fixed value here keeps the example focused on
        // AuditorAware's role, not on Spring Security itself.
        return () -> Optional.of("system");
    }
}
```

`@CreatedBy` and `@LastModifiedBy` work exactly like their date counterparts — same listener, same lifecycle timing — but capture the identity of whoever made the change instead of a timestamp. This project's real `Question` entity already has a `reviewedBy` column, but that's

set manually, as part of a deliberate admin review action (per this project's own question-pool review workflow) — a genuinely different thing from automatically recording who created or last touched the row itself.

AuditorAware<T>: Where "Who" Comes From

Spring Data JPA has no built-in notion of a "current user" — something has to supply that answer.

`AuditorAware<T>` is that something: a single-method interface returning an `Optional<T>` representing whoever is "currently" making a change, called automatically every time an audited entity is saved. A real application would implement it by reading from Spring Security's `SecurityContextHolder` — the currently authenticated user's name or id — rather than returning a fixed value; the fixed `"system"` value in the example keeps the focus on `AuditorAware`'s role itself, not on Spring Security, which this category doesn't cover.

Sharing Audit Fields Across Entities with @MappedSuperclass

Once more than one entity needs the same audit fields, repeating `@CreatedDate` / `@LastModifiedDate` on each one becomes exactly the kind of repetition auditing was meant to remove in the first place.

```
import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.EntityListeners;
import jakarta.persistence.Id;
import jakarta.persistence.MappedSuperclass;
import org.springframework.data.annotation.CreatedDate;
import org.springframework.data.annotation.LastModifiedDate;
import org.springframework.data.jpa.domain.support.AuditingEntityListener;

import java.time.LocalDateTime;

// @MappedSuperclass isn't itself an @Entity -- it's a base class whose
// fields get copied into every entity that extends it, without a table of
// its own. This is where the audit fields belong once more than one
// entity needs them: this project's real Question already has createdAt/
// updatedAt columns (set by hand today, in QuestionIngestService) -- if
// QuestionOption or another entity needed the same two columns, repeating
// @CreatedDate/@LastModifiedDate on each one would itself become the same
// kind of repetition auditing was meant to remove in the first place.
@MappedSuperclass
@EntityListeners(AuditingEntityListener.class)
abstract class AuditableBaseExample {

    @CreatedDate
    @Column(name = "created_at")
    private LocalDateTime createdAt;

    @LastModifiedDate
    @Column(name = "updated_at")
    private LocalDateTime updatedAt;
}

// Any entity extending AuditableBaseExample gets createdAt/updatedAt for
// free -- no @EntityListeners of its own needed (it's inherited), and no
// repeated field declarations either.
@Entity
class AuditableQuestionExample extends AuditableBaseExample {
    @Id
    private Long id;
    private String text;
}
```

`@MappedSuperclass` isn't itself an `@Entity` and has no table of its own — it's a base class whose fields get copied into every entity that extends it. This project's real `Question` already has `createdAt` / `updatedAt` columns; if `QuestionOption` or another entity needed the exact same two fields, extending a shared `@MappedSuperclass` avoids declaring `@CreatedDate` / `@LastModifiedDate` (and `@EntityListeners`) separately on each one.

Common Misconceptions

"`@CreatedDate` alone is enough to make auditing work." It isn't — without `@EntityListeners(AuditingEntityListener.class)` on the entity AND `@EnableJpaAuditing` somewhere in the application, the field is simply never populated, with no error. **"**

`@LastModifiedDate` **only updates on genuine field changes."** It updates on every save that reaches the database, the same way dirty checking (covered in "Transaction Management") writes any tracked change — it isn't selectively smart about which saves "really" changed something meaningful. **"`AuditorAware` needs Spring Security to work at all."** It doesn't structurally depend on it — it's just an interface returning an `Optional<T>`; a real application typically implements it by reading from Spring Security, but the mechanism itself is independent of that specific choice.

What Comes Next

Every topic in this category so far has focused on reading, writing, or tracking entity data correctly. "Testing Spring Data JPA Repositories," the final lesson in this category, shifts to verifying that all of it — repositories, queries, projections, relationships, even auditing — actually behaves the way these lessons describe, with `@DataJpaTest`.

Best Practices

- Add both `@EntityListeners(AuditingEntityListener.class)` and `@EnableJpaAuditing` together — one without the other silently does nothing.
- Reach for `@MappedSuperclass` the moment a second entity needs the same audit fields, rather than repeating the annotations on each one.
- Implement `AuditorAware<T>` by reading the current user from Spring Security's `SecurityContextHolder` in a real application, not a fixed value.
- Prefer auditing over hand-written `LocalDateTime.now()` calls for any field that's genuinely just "when was this created/modified" — reserve manual timestamp fields for cases with their own distinct meaning, like this project's `reviewedAt`.

Common Mistakes

- Adding `@CreatedDate` / `@LastModifiedDate` without `@EntityListeners` or `@EnableJpaAuditing`, and being confused why the fields stay `null`.
- Assuming `@LastModifiedDate` updates only when something "meaningful" changed, rather than on every save that reaches the database.
- Confusing `reviewedBy` (a deliberate, manual review action, as in this project's own `Question` entity) with `@LastModifiedBy` (an automatic record of who last saved the row) — they answer different questions.
- Repeating `@CreatedDate` / `@LastModifiedDate` on every entity individually instead of extracting them to a shared `@MappedSuperclass` once more than one entity needs them.

Summary, Cheat Sheet, and Glossary

Summary

- `@CreatedDate` / `@LastModifiedDate` replace manually written `LocalDateTime.now()` calls (like this project's real `QuestionIngestService` pattern) with automatic timestamps.
- Both `@EntityListeners(AuditingEntityListener.class)` (on the entity) and `@EnableJpaAuditing` (on a configuration class) are required together — either alone does nothing.
- `@CreatedBy` / `@LastModifiedBy` record who made a change, using the same listener mechanism as the date annotations.
- `AuditorAware<T>` supplies the "who" — typically implemented by reading the current user from Spring Security in a real application.
- `@MappedSuperclass` shares audit fields across multiple entities without repeating the annotations on each one.

Cheat Sheet

```
// Turn auditing on for the application
@Configuration
@EnableJpaAuditing
class JpaAuditingConfig {}

// An audited entity
@Entity
@EntityListeners(AuditingEntityListener.class)
class Question {
    @CreatedDate
    private LocalDateTime createdAt;

    @LastModifiedDate
    private LocalDateTime updatedAt;

    @CreatedBy
    private String createdBy;

    @LastModifiedBy
    private String lastModifiedBy;
}

// Supplying "who"
@Bean
AuditorAware<String> auditorProvider() {
    return () -> Optional.ofNullable(SecurityContextHolder.getContext().getAuthentication())
            .map(Authentication::getName);
}

// Sharing fields across entities
@MappedSuperclass
@EntityListeners(AuditingEntityListener.class)
abstract class Auditable {
    @CreatedDate private LocalDateTime createdAt;
    @LastModifiedDate private LocalDateTime updatedAt;
}
```

Glossary

- **@CreatedDate / @LastModifiedDate**: annotations that populate a timestamp field automatically on insert (and, for the latter, every subsequent update).
- **@EntityListeners(AuditingEntityListener.class)**: registers the listener, on an entity, that actually drives the auditing annotations.
- **@EnableJpaAuditing**: turns on Spring Data JPA's auditing infrastructure for the whole application.
- **@CreatedBy / @LastModifiedBy**: annotations that record who made a change, using the same listener mechanism as the date annotations.
- **AuditorAware<T>**: the interface supplying "who the current user is" to `@CreatedBy` / `@LastModifiedBy`.
- **@MappedSuperclass**: a non-entity base class whose fields (like audit fields) are inherited by every entity that extends it, without its own table.