Answer the following

1. **What is JPA Auditing?**

Ans –

JPA Auditing

JPA Auditing refers to a set of features provided by the Java Persistence API (JPA) that automatically tracks and records certain entity-related events, such as the creation and modification times of entities. This is particularly useful for maintaining a history of when entities were created or last updated, without the need to manually handle these timestamps.

JPA Auditing is typically achieved by using annotations such as @CreatedDate, @LastModifiedDate, @CreatedBy, and @LastModifiedBy, which automatically populate the corresponding fields of an entity when certain events occur (such as creating or updating an entity). The auditing mechanism simplifies code and reduces the possibility of errors.

Key Features of JPA Auditing:

1. **Automatic Timestamping**: Automatically records the date and time when an entity is created and modified.
2. **Audit Information**: Tracks who created or last modified an entity, if configured to do so (@CreatedBy, @LastModifiedBy).
3. **Minimal Code Changes**: Instead of manually setting timestamps or modifying entity attributes during CRUD operations, JPA Auditing handles it automatically.
4. **Integration with Spring Data JPA**: JPA Auditing is often used in conjunction with Spring Data JPA to manage entity audit information.

2. **Explain JPA Auditing using @CreatedDate and @LastModifiedDate.**

Ans-

JPA Auditing can be configured using specific annotations: @CreatedDate and @LastModifiedDate. Here's a step-by-step explanation of how to set it up and use it.

Steps to Enable JPA Auditing:

1. **Enable JPA Auditing**: To enable JPA Auditing in a Spring Boot project, we need to annotate the main application class or a configuration class with @EnableJpaAuditing.

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.data.jpa.repository.config.EnableJpaAuditing;


@SpringBootApplication
@EnableJpaAuditing
public class Application {
   public static void main(String[] args) {
      SpringApplication.run(Application.class, args);
   }
}
```

2. **Add Auditing Fields in the Entity Class**: In the entity class, you need to declare fields to hold the creation and modification timestamps, and annotate them with @CreatedDate and @LastModifiedDate.
    - @CreatedDate: Automatically set the creation timestamp when the entity is persisted.
    - @LastModifiedDate: Automatically set the timestamp when the entity is updated.

    Example:

```
package com.example.model;


import org.springframework.data.annotation.CreatedDate;
import org.springframework.data.annotation.LastModifiedDate;


import jakarta.persistence.*;
import java.time.Instant;
```

```
@Entity
public class Customer {

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

    private String name;

    @CreatedDate
    @Column(name = "created_at", updatable = false)
    private Instant createdAt;

    @LastModifiedDate
    @Column(name = "modified_at")
    private Instant modifiedAt;

    // Getters and setters
    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }
```

```
    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public Instant getCreatedAt() {
        return createdAt;
    }

    public void setCreatedAt(Instant createdAt) {
        this.createdAt = createdAt;
    }

    public Instant getModifiedAt() {
        return modifiedAt;
    }

    public void setModifiedAt(Instant modifiedAt) {
        this.modifiedAt = modifiedAt;
    }
}
```

3. **Use @CreatedDate and @LastModifiedDate:**
    - **@CreatedDate**: This annotation automatically sets the value of the field it annotates to the current timestamp when a new entity is created and persisted in the database.

- **@LastModifiedDate**: This annotation automatically updates the field it annotates to the current timestamp whenever the entity is updated.

For example, when you create a new Customer entity, the createdAt field will automatically be set to the current date and time. Similarly, when you update the entity, the modifiedAt field will automatically be updated with the current date and time.

4. **Use AuditingEntityListener:**

If you're using Spring Data JPA with auditing enabled, you don't need to use @EntityListeners(AuditingEntityListener.class) explicitly in most cases. However, it's important to ensure the auditing listener is registered so that the annotations work as expected.

Example:

```
@EntityListeners(AuditingEntityListener.class)
@Entity
public class Customer {
   // Fields and annotations...
}
```