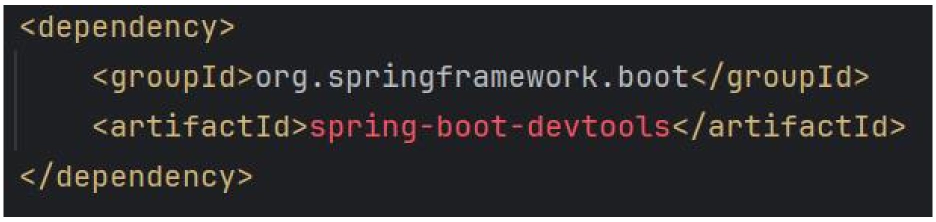
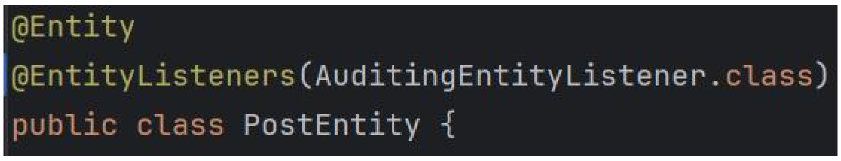
- To add **dev tools,** add the following dependency in *pom.xml* file.
  - 

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
</dependency>
```

- *IntelliJ idea* specific settings:
  - **Setting >> Build, Execution, Deployment >> Compiler**
    - Select the option "**Build project automatically**"
    - So that the build will be automatically generated if u do any changes to the files (in *running* or *debugging* state).
  - **Advanced Setting >>**
    - Select the option "**Allow auto-make to start even if developed applicatoin is currently running**"

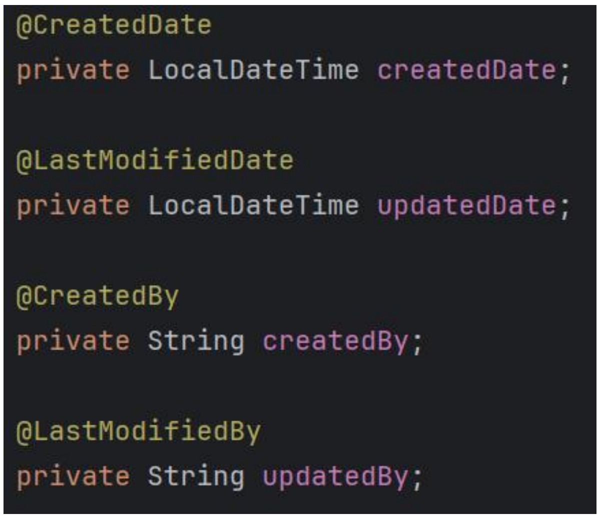
Auditing

- Auditing in Spring Boot allows you to automatically populate certain fields, such as creation and modification timestamps, as well as the user who created or modified the entity.
- Create Auditable base Entity using the following annotations:
  - **@EntityListeners(*AuditingEntityListener.class*)**

    ```java
    @Entity
    @EntityListeners(AuditingEntityListener.class)
    public class PostEntity {
    ```

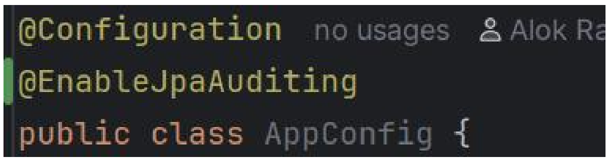
  - Now you can use the annotations **@CreatedAt**, **@LastModifiedAt**, **@CreatedBy**, **@LastModifiedBy** and create the fields inside the ***Entity***.

    ```java
    @CreatedDate
    private LocalDateTime createdDate;

    @LastModifiedDate
    private LocalDateTime updatedDate;

    @CreatedBy
    private String createdBy;

    @LastModifiedBy
    private String updatedBy;
    ```

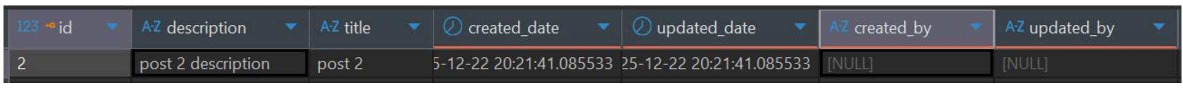
    - These are writtn inside entity; not DTO

  - Now, to make all these things work, you need to add **@EnableJpaAuditing** in any of the config file (file having **@Configuration** annotation)

    ```java
    @Configuration  no usages  Alok Ra
    @EnableJpaAuditing
    public class AppConfig {
    ```

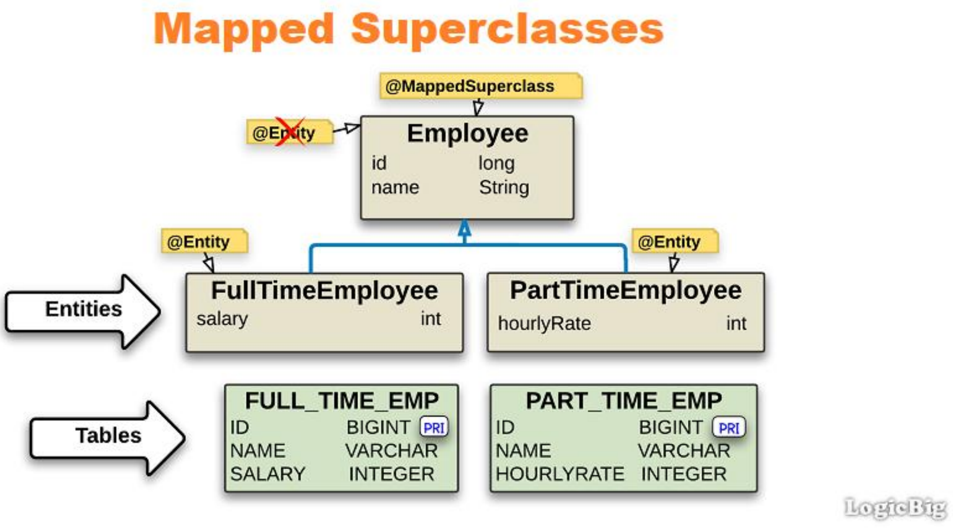
  - Now it'll work properly when someone use **post**, **put** request.

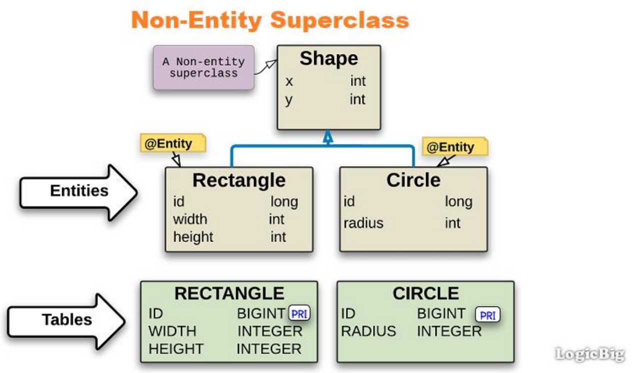
id	description	title	created_date	updated_date	created_by	updated_by
2	post 2 description	post 2	5-12-22 20:21:41.085533	25-12-22 20:21:41.085533	[NULL]	[NULL]

    - Now if someone do some changes, it'll be visible in the Database;
- Instead of writing **@EntityListener** in any particular entity, its better to create one class and write this annotation there. Whichever *Entity* wants to be audited, they can directly inherit that class.

- There is a annotation **@MappedSuperClass**,
  - lets say you want some common fields to be there in some Entities.
  - So, instead of writing those in every Entity, its better to write in a parent class and extend that class from the actual Entity class.
  - But, here the fields that are defined in the parent class will not be included in the table by default.
  - If you write **@MappedSuperClass** in the parent class, only then those fields will be part of the Entity table.



  -
    - **@MappedSuperClass** is writtn in *@Employee*, so the entities (FullTimeEmployee, PartTimeEmployee) are having the *id, name* fields in their tables.

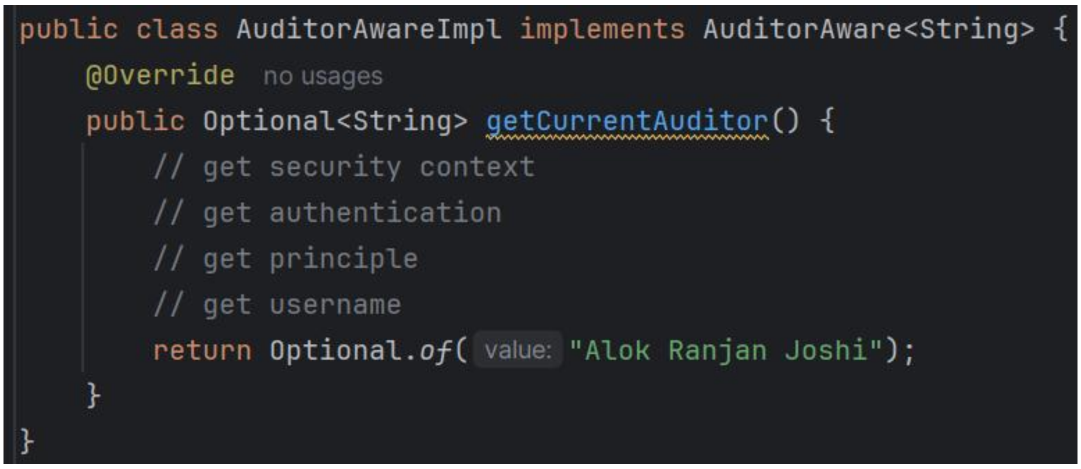


  -
    - **@MappedSuperClass** is not written for the parent class *Shape*, so the Entities (Rectangle, Circle) are not having *x, y* fields in their tables.

- But till now, you'll only get **@CreatedAt** and **@LastModifiedAt** , but the fields for **@CreatedBy**, **@LastModifiedBy** will be **null** only, as it doesn't know who made the changes.

- Its part of spring security to fetch the current user and add that user in the database for createdBy or updatedBy fields.
- For now, we can use some dummy name for this.

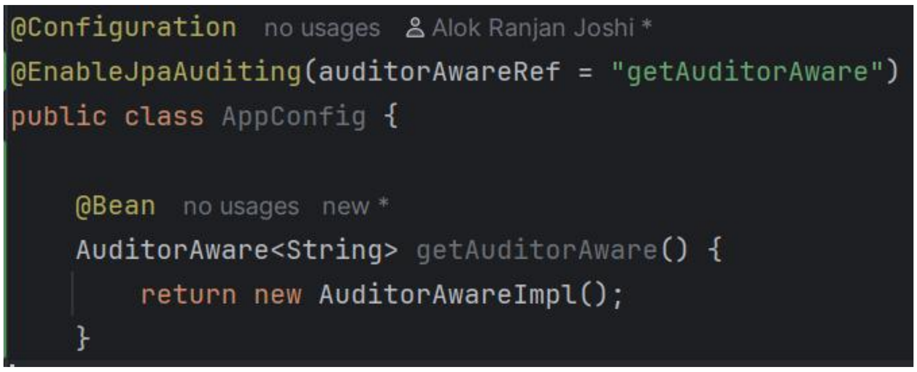
➢ You need to write a class that implement the interface **AuditorAware**



```java
public class AuditorAwareImpl implements AuditorAware<String> {
    @Override  no usages
    public Optional<String> getCurrentAuditor() {
        // get security context
        // get authentication
        // get principle
        // get username
        return Optional.of( value: "Alok Ranjan Joshi");
    }
}
```

- For now I am returning “Alok Ranjan Joshi” as the modifier/creator name.

➢ After this, you need to update the annotation **@EnableJpaAuditing** of the config class.



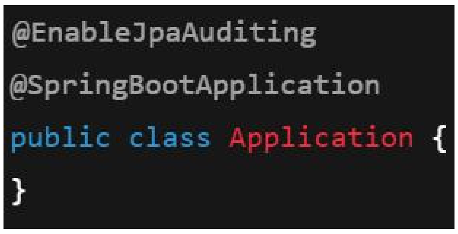
```java
@Configuration  no usages  Alok Ranjan Joshi *
@EnableJpaAuditing(auditorAwareRef = "getAuditorAware")
public class AppConfig {

    @Bean  no usages  new *
    AuditorAware<String> getAuditorAware() {
        return new AuditorAwareImpl();
    }
```

- **auditorAwareRef** takes the bean name.
- As the method name is *getAuditorAware*, by default method name becomes the bean name if you are not specifically mentioning a different name for the bean.

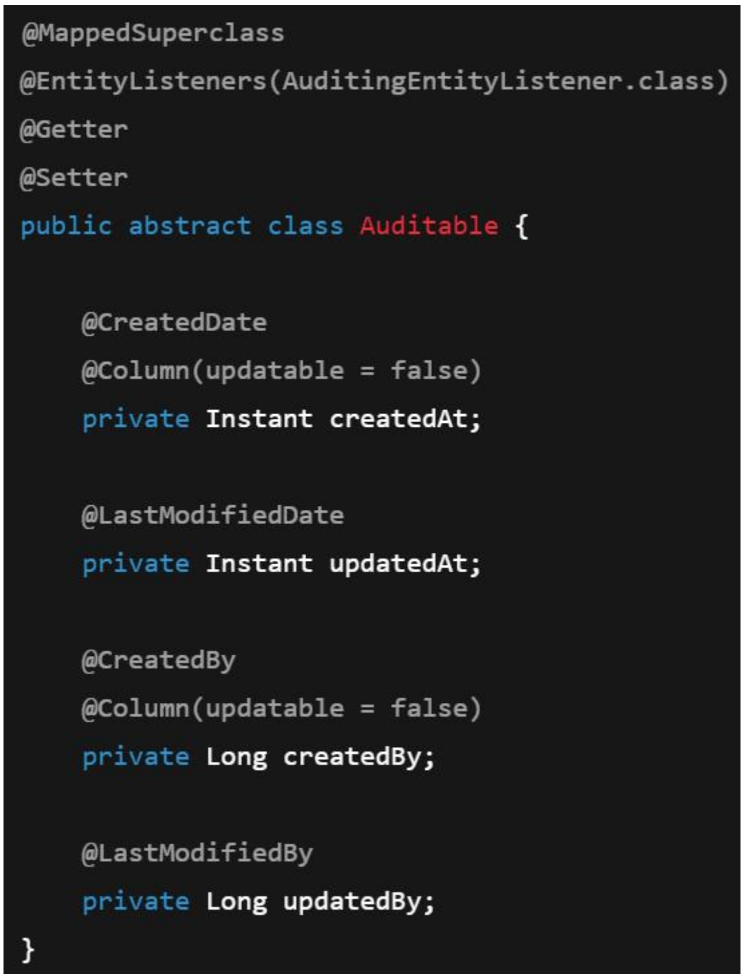
➢ F

➢ F

- So, All the steps are:
  - Create Auditable base Entity using the following Annotations
    - **@EntityListeners(AuditingEntityListener.class)**.
    - You can create the fields using **@CreatedBy**, **@LastModifiedBy**, **@CreatedAt**, **@LastModifiedAt**.
  - If you are creating a super class for *auditing*, then use **@MappedSuperClass** on that parent class, otherwise the entities that inherit that parent class won't contain the fields defined in that parent class.
  - Now write **@EnableJpaAuditing** in any config class (annotated with *@Configuration*).
  - Create an class implementing the interface **AuditorAware** interface.
  - Then edit the **config** file's annotation **@EntityListeners** and pass the *bean name* (by default the method name) for the **auditorAwareRef.**
  -

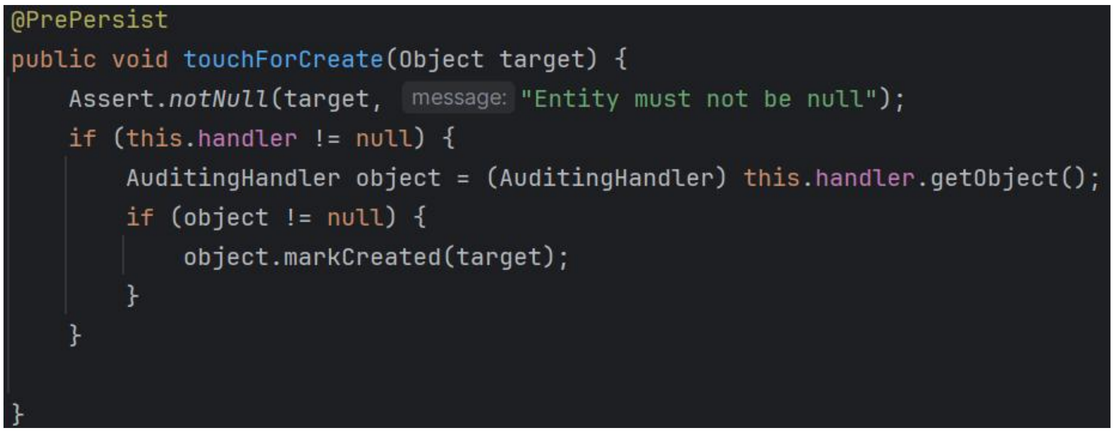
- Optimized steps:
  - Enable auditing (MANDATORY)
    - 

```java
@EnableJpaAuditing
@SpringBootApplication
public class Application {
}
```

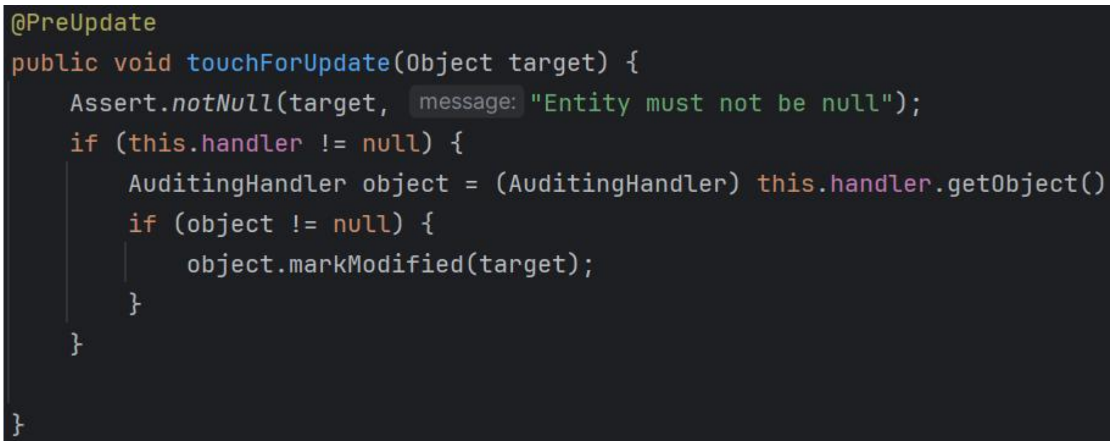
    - Better to write in the main application file (@SpringBootApplication)
  - Create a base audit class (BEST PRACTICE)
    - 

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

    @CreatedDate
    @Column(updatable = false)
    private Instant createdAt;

    @LastModifiedDate
    private Instant updatedAt;

    @CreatedBy
    @Column(updatable = false)
    private Long createdBy;

    @LastModifiedBy
    private Long updatedBy;
}
```

    - **@MappedSuperClass** is necessary; without that the fields will not be available in the *__entities__ that inherits this __Auditable__ class.*
    - **AuditingEntityListener** class contains all the methods that is being triggered to update the entity before **create/update**.

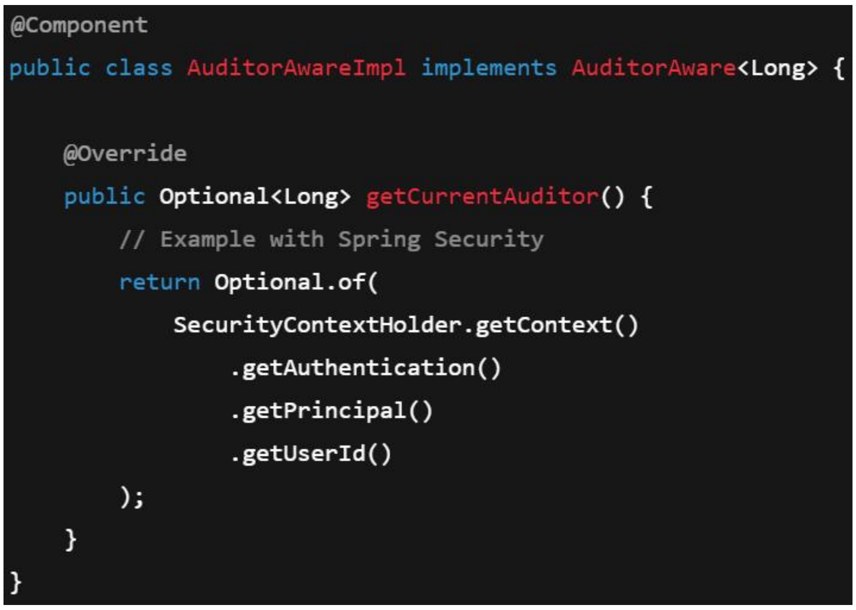
- 

```
@PrePersist
public void touchForCreate(Object target) {
    Assert.notNull(target,  message: "Entity must not be null");
    if (this.handler != null) {
        AuditingHandler object = (AuditingHandler) this.handler.getObject();
        if (object != null) {
            object.markCreated(target);
        }
    }

}
```

    - It is triggered before **create** event.
- 

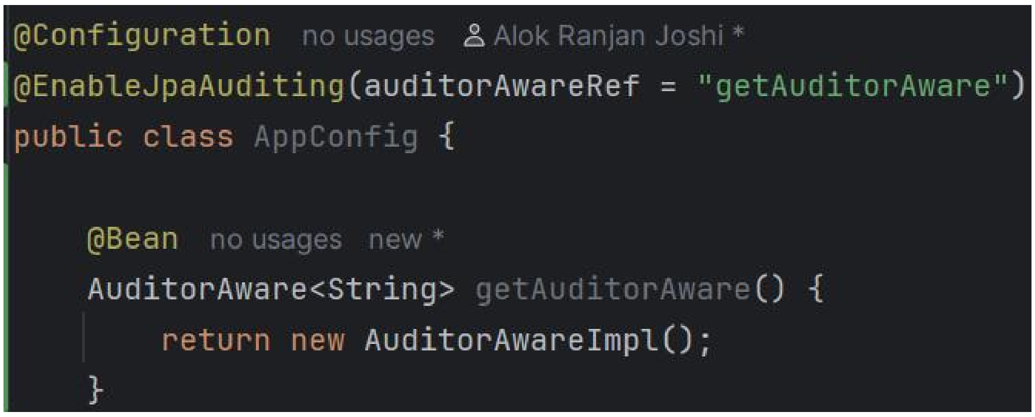
```
@PreUpdate
public void touchForUpdate(Object target) {
    Assert.notNull(target,  message: "Entity must not be null");
    if (this.handler != null) {
        AuditingHandler object = (AuditingHandler) this.handler.getObject()
        if (object != null) {
            object.markModified(target);
        }
    }

}
```

    - It is triggered before **update** event.
- You can write your own methods using the annotations **@PrePersist, @PreUpdate, @PreRemove** that will be triggered before the respective operations.

- Handling **@CreatedBy** and **@LastModifiedBy**
  - 

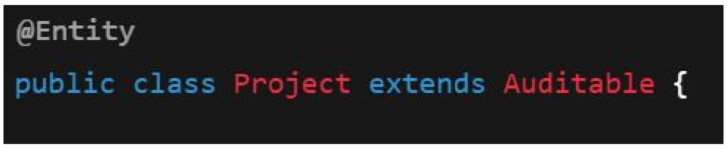
```
@Component
public class AuditorAwareImpl implements AuditorAware<Long> {

    @Override
    public Optional<Long> getCurrentAuditor() {
        // Example with Spring Security
        return Optional.of(
            SecurityContextHolder.getContext()
                .getAuthentication()
                .getPrincipal()
                .getUserId()
        );
    }
}
```

- You need to create the bean of this class. Either use **@Component** or **@Configuration, @Bean** .
- If you are having more than one classes that implements **AuditorAware**, and you are creating multiple beans, then you need to pass **auditorAwareRef** inside **@EnableJpaAuditing**.



```
@Configuration  no usages  Alok Ranjan Joshi *
@EnableJpaAuditing(auditorAwareRef = "getAuditorAware")
public class AppConfig {

    @Bean  no usages  new *
    AuditorAware<String> getAuditorAware() {
        return new AuditorAwareImpl();
    }
```

  - Like here, the bean name will be same as the method name i.e. **getAuditorAware** so I am passing the bean name in the **auditorAwareRef.**
  - But in case of single bean, no need of passing this.
- Extend the base class in entities

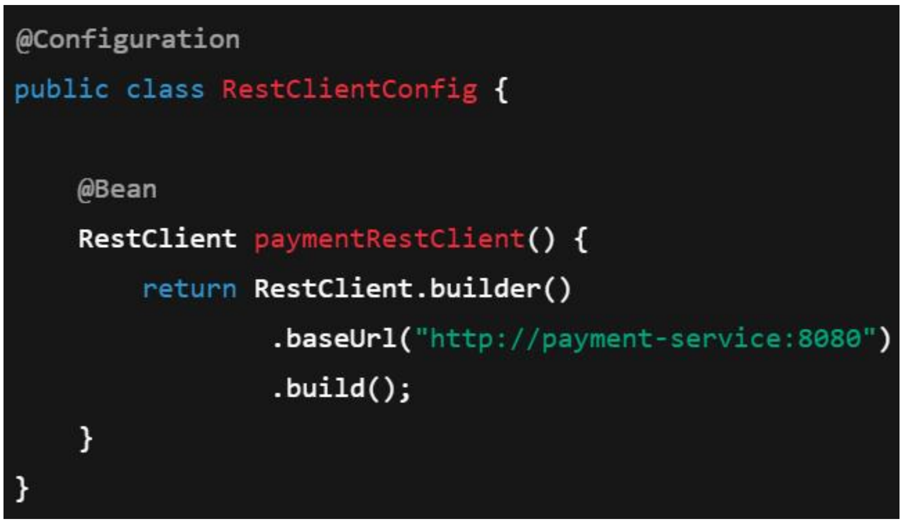


```
@Entity
public class Project extends Auditable {
```

- **Overall Annotations** --
  - @EnableJpaAuditing(AuditingEntityListener.class)
  - @MappedSuperClass
  - @CreatedAt, @LastModifiedAt, @CreatedBy, @LastModifiedBy
  - @PrePersist, @PreUpdate, @PreRemove
  - **AuditorAware** class has to be implemented for @CreatedBy, @LastModifiedBy
  - --
- **To keep track of every version like different versions for every change and all will be containing who changed and what was changed and all, you can use Hibernate Envers .**
-

<u>RestClient</u>

- RestClient is a synchronous HTTP client that offers a modern fluent API. It offers an abstraction over HTTP libraries that allows for convenient conversion from a Java object to an HTTP request, and the creation of objects from an HTTP response.
- Generally in case of *microservice* architecture, one service needs to call another service.
  - Don't think @RestController and RestClient are same.
  - RestController create API; RestClient consumes those APIs.
- Lets say you have 2 services in your microservice architecture; and one service needs to call another service (API call) then RestClient is used;
  - **Order Service** ——**calls**——→ **User Service** (for example)
- Steps:
  - First you'll configure the RestClient giving some base url. For example http://localhost:8080
  - After that you can call any type of method i.e. get, post, put, patch, delete with request body, headers etc etc just like normal API call (just like API call is made using *axios* in react)
  - Either you can create the object of type RestClient using **RestClient.builder()…..build()** method or create a *bean* of this and inject that everywhere.



```
@Configuration
public class RestClientConfig {

    @Bean
    RestClient paymentRestClient() {
        return RestClient.builder()
                .baseUrl("http://payment-service:8080")
                .build();
    }
}
```

  -
-
- F
- F
- F
- F

➢