Hibernate: A Guide to Essential Annotations

Saurabh Kundu · Follow
5 min read · Apr 11, 2023

Listen Share

A comprehensive list of all the basic and most commonly used Hibernate Annotations.

Hibernate

Hibernate is a popular **object-relational mapping (ORM)** framework for Java that provides a convenient and powerful way to map Java classes to database tables. Annotations makes it easy to define relationships between entities, specify how they should be persisted to the database, and manage the inheritance hierarchy of entity classes.

In this blog, we will explore some of the most commonly used Hibernate annotations which will help us to create powerful and flexible data models in our Java applications. Let's begin.

@Entity: *This annotation is used to mark a class as a persistent entity in the database.*

@Table: *This annotation is used to specify the name of the database table that corresponds to the annotated entity.*

@Tables: *This annotation is used to specify multiple **@Table** annotations on a single entity, each representing a table for a specific entity hierarchy.*

@Column: *This annotation is used to specify the mapping of a persistent entity property to a database column.*

@Id: *This annotation is used to specify the primary key property of an entity.*

@RowId: *This annotation is used to map a column to a primary key that is automatically generated by the database. It is useful when you need to generate a unique identifier for each row in a table.*

@NaturalId: *This annotation is used to mark a property as a natural identifier, which means it has a business significance and should be uniquely identified. Hibernate will create a unique index for faster lookup.*

@CollectionId: *This annotation is used to generate a unique identifier for each element in a collection that is mapped as a separate table with a foreign key to the parent table.*

@Generated: *This annotation is used to mark a property as generated by the database. It can be used to retrieve the generated value after inserting a new entity into the database.*

@GenericGenerator: *This annotation is used to specify a custom identifier generator for an entity. It can be used to generate identifiers using a custom algorithm or an external service.*

@GeneratorType: *This annotation is used to specify the type of identifier generator to be used for an entity. It can be used to specify generator type for an entity, such as UUID or* *sequence-based.*

@GeneratedValue: *This annotation is used to specify the generation strategy for the values of the primary key. Generation types available are:* ***TABLE, SEQUENCE, IDENTITY, AUTO****.*

@Cascade: *This annotation is used to specify how the changes propagate to the associated entities or collections automatically. Cascade types available are:* ***ALL, PERSIST, MERGE, REMOVE, REFRESH, DETACH, SAVE_UPDATE****.*

@ManyToOne: *This annotation is used to define a* **many-to-one** *relationship between two persistent entities.*

@OneToOne: *This annotation is used to define a* **one-to-one** *relationship between two persistent entities.*

@Embedded: *This annotation is used to map an entity's property to a persistent class that is annotated with* ***@Embeddable****.*

@Embeddable: *This annotation is used to mark a class as a value type that can be embedded in other entities using* ***@Embedded****.*

@NamedQuery: *This annotation is used to define a named query that can be used to retrieve entities from the database. It can be used to execute a complex query multiple times with different parameters.*

@NamedQueries: *This annotation is used to define multiple named queries for an entity. It can be used to define multiple queries for an entity with different parameters.*

@NamedNativeQuery: *This annotation is used to define a named native query that can be used to retrieve entities from the database using native SQL. It is used to execute a complex query that cannot be expressed using HQL.*

@NamedNativeQueries: *This annotation is used to define multiple named native queries for an entity. It can be used to define multiple queries for an entity with different parameters using native SQL.*

@Inheritance: *This annotation is used to specify the inheritance strategy to be used for a class hierarchy. It can be used with **@Entity** annotation to define how an entity class hierarchy should be mapped to the database tables.*

@DiscriminatorColumn: *This annotation is used to specify the name and type of the discriminator column in an inheritance hierarchy. It is used to distinguish between entities of different types in a single table.*

@Transient: *This annotation is used to specify that a property should not be persisted to the database.*

@Temporal: *This annotation is used to specify the temporal type for a date or time property.*

@Lob: *This annotation is used to mark a persistent entity property as a Large Object type. It is used to map large fields such as text or binary data to the database.*

@OrderBy: *This annotation is used to specify the ordering of a collection of entities in a persistent entity.*

@PrimaryKeyJoinColumn: *This annotation is used to specify the mapping of the primary key of a parent entity to a foreign key column in a child entity. It is used in an inheritance hierarchy where a child entity shares the same primary key as its parent entity.*

@JoinColumn: *This annotation is used to specify the mapping of a foreign key column in a child entity to a primary key column in a parent entity. It is used to define a many-to-one or one-to-one association between two entities.*

@MapsId: *This annotation is used to map the primary key of a parent entity to a foreign key column in a child entity. It is used in a one-to-one association where the child entity's* *primary key is the same as the parent entity's primary key.*

@JoinTable: *This annotation is used to define the association table for a many-to-many association between two entities. It is used to map a collection of entities to a join table that has foreign key columns referencing the primary keys of the two associated entities.*

@Immutable: *This annotation is used to specify that an entity's state cannot be changed. It makes sure that the entity remains in a consistent state throughout the application's lifecycle.*

@Check: *This annotation is used to specify a check constraint on a column or a set of columns in a database table. This ensures that the values of the column or columns meet a certain condition or set of conditions before being inserted or updated in the table.*

@OptimisticLock: *This annotation is used to ensure that concurrent transactions do not overwrite each other's changes when updating the same entity. Following optimistic locking types are available:* ***ALL, NONE, VERSION, DIRTY****.*

@Version: *This annotation is used to specify the version property of an entity for optimistic locking.*

@Nationalized: *This annotation is used to mark a string property as a nationalized character data type, which means it is stored in the database using the national character set. It is useful when you need to support multiple character sets and encodings.*

@Cache: *This annotation is used to enable second-level caching for an entity or a collection. This can improve performance by reducing the number of database queries.*

References:

Chapter 3. Hibernate Annotations Red Hat JBoss Enterprise Application Platform 7.3 | Red Hat…

Access Red Hat's knowledge, guidance, and support through your subscription.

access.redhat.com

All Hibernate Annotations : Mapping Annotations

This article provides a quick overview of all Hibernate mapping annotations. These Hibernate mapping annotations...

www.javaguides.net

Java Hibernate Software Development Software Engineering Annotations

Follow

Written by Saurabh Kundu

19 Followers

Open in app ↗ Sign up Sign in

Medium Search

More from Saurabh Kundu

Saurabh Kundu

Spring Boot Annotations: A Guide to Essential Annotations

A comprehensive list of all the basic and most commonly used Spring Boot Annotations.

6 min read · Mar 31, 2023

65

Saurabh Kundu

How to use contract first approach with Open API

What is contract first approach ?

5 min read · Mar 19, 2023

11

Saurabh Kundu

Java 8 Key Concepts: Functional Interfaces

This is part one of a four—part series exploring the important concepts in Java 8.

5 min read · Apr 14, 2023

2

Saurabh Kundu

Springtime in Amsterdam: A Colorful Escape

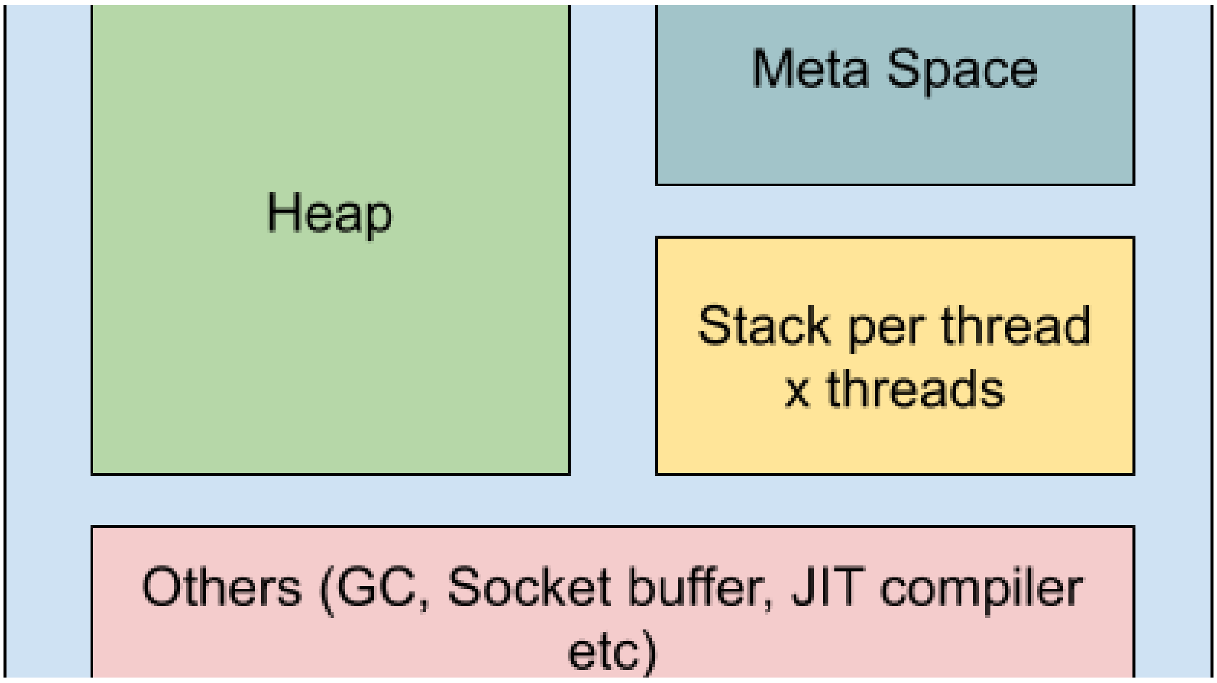
As a travel enthusiast, I have compiled a list of three must-visit places in and around Amsterdam to make your trip unforgettable. To help…

4 min read · Mar 21, 2023

18

See all from Saurabh Kundu

Recommended from Medium


Heap
Meta Space
Stack per thread x threads
Others (GC, Socket buffer, JIT compiler etc)

Shoeb Ahmed Tanjim

How we reduced the memory consumption of spring boot application over 40% for the development...

The base spring boot application takes an affordable amount of memory. But when we start adding more dependencies, the memory consumption...

4 min read · 6 days ago

148

Robinson Githae

Model Mapping Entities to DTO /POJO and Vise Versa in Java Spring Boot with Lombok for Security

A model, also known as an Entity is a class with attributes that can be persisted on the database. For relational databases...

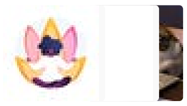
3 min read · Sep 3, 2023

30 1

Lists

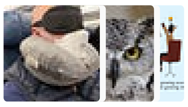
General Coding Knowledge

20 stories · 742 saves



Stories to Help You Grow as a Software Developer

19 stories · 678 saves

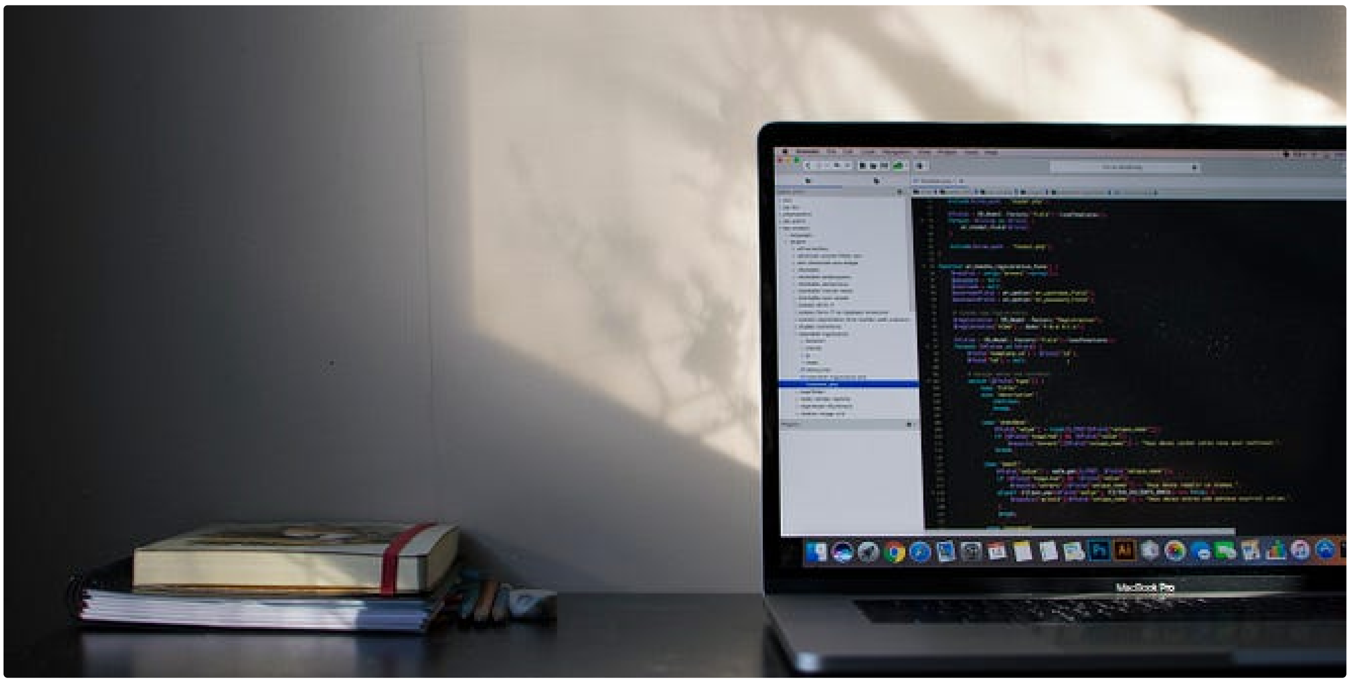


Leadership

41 stories · 187 saves

Coding & Development

11 stories · 355 saves



Malvin Lok

How to Inject Map/List/Array from Configuration in Spring Boot

As a backend engineer, as long as the use of Spring or Spring Boot, then basically can not be separated from the @Value and...

· 2 min read · 3 days ago

20

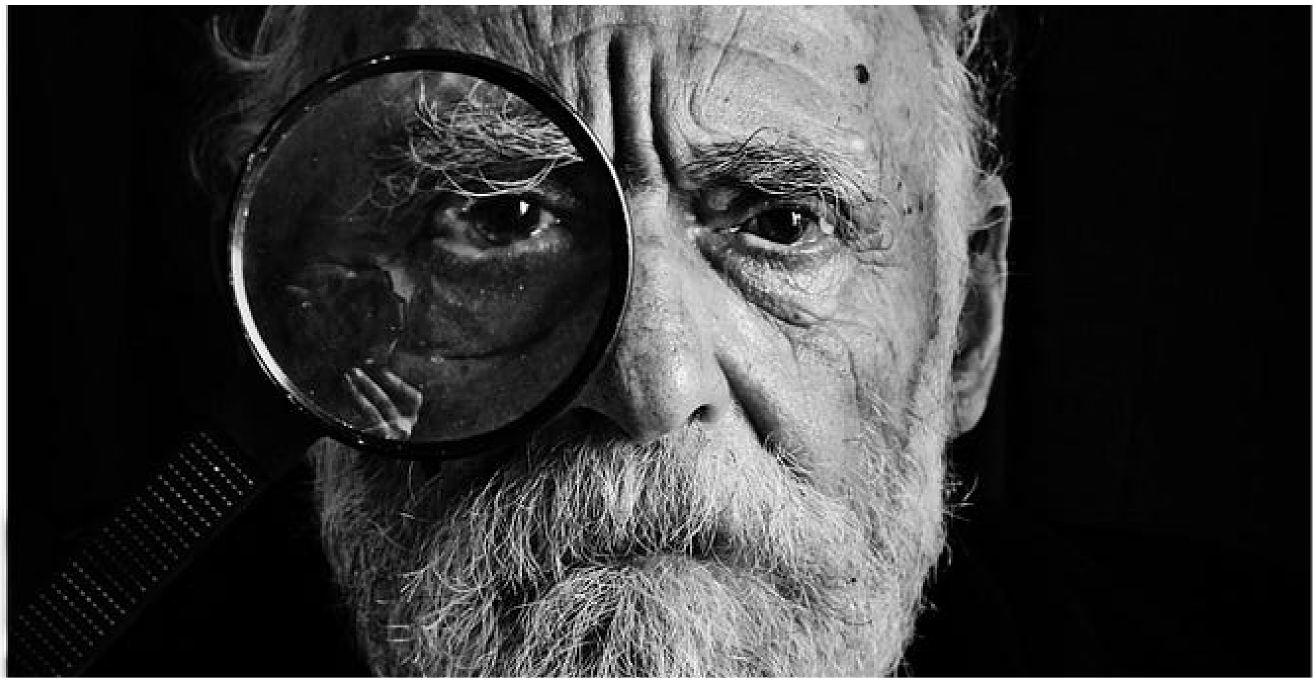
Alexander Obregon

Understanding the Basics of Serialization in Spring Boot

Introduction

· 4 min read · Jul 19, 2023

9



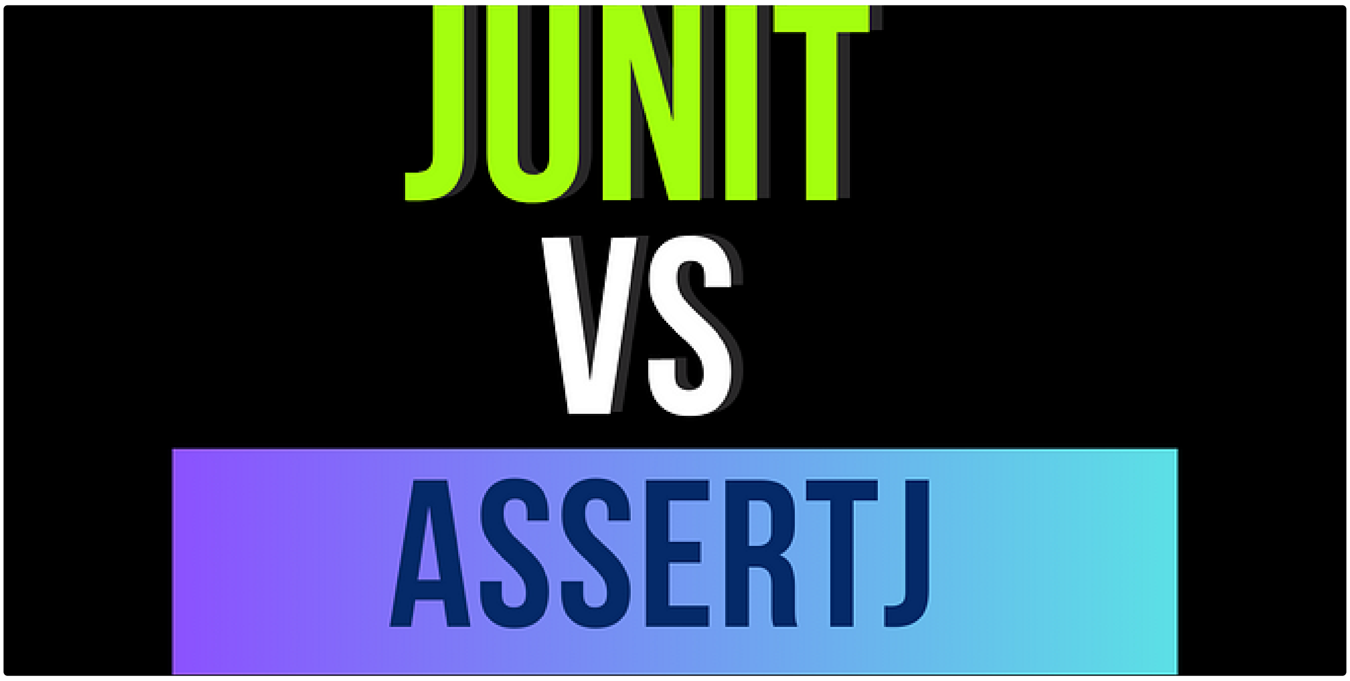
Himani Prasad

Validations in Spring Boot

Validation is like a quality check for data. Just like a teacher marks your answers right or wrong, validation checks if the information…

6 min read · Aug 23, 2023

109 3



YD in Nerd For Tech

JUnit vs. AssertJ: Choosing the Right Testing Framework for Your Java Project

Introduction

4 min read · Sep 5, 2023

75

See more recommendations