Core Hibernate/JPA Annotations

1. `@Entity`

- Marks a class as a JPA entity (i.e., a table in the database).
- Hibernate will map this class to a database table.

```
@Entity
public class Student { ... }
```

Creates a table named `Student` (or a custom one if we use `@Table`).

2. `@Table`

- Specifies the name of the table and schema (optional).
- Useful when the database table name is different from the class name.

```
@Entity
@Table(name = "student_table")
public class Student { ... }
```

3. `@Id`

- Specifies the **primary key** of the entity.

```
@Id
private int id;
```

4. `@GeneratedValue`

- Defines how the primary key will be generated.
- Strategies:
    - `AUTO` → Hibernate chooses automatically.
    - `IDENTITY` → Uses auto-increment (MySQL).
    - `SEQUENCE` → Uses sequence objects (Oracle/PostgreSQL).
    - `TABLE` → Uses a special table to generate IDs.

```
@Id
@GeneratedValue(strategy = GenerationType.AUTO)
private int id;
```

5. `@Column`

- Maps a field to a table column.
- Can define column name, length, nullability, uniqueness, etc.

```
@Column(name = "student_name", length = 50, nullable = false,
unique = true)
private String name;
```

6. `@Transient`

- Field will **not be persisted** in the database.
- Used for calculated fields or temporary values.

```
@Transient
private int ageInMonths;
```

7. `@Lob`

- Used for **large objects** (CLOB for text, BLOB for binary data).

```
@Lob
private String description;
```

8. `@Temporal`

- Used for **Date/Time fields** to specify precision (`DATE`, `TIME`, `TIMESTAMP`).

```
@Temporal(TemporalType.DATE)
private Date dob;
```

9. `@Embedded` and `@Embeddable`

- For reusable, embedded objects (value types) inside entities.

```
@Embeddable
public class Address {
   private String city;
   private String state;
}
```

```
@Entity
public class Student {
   @Embedded
   private Address address;
}
```

Relationship Annotations

10. `@OneToOne`

- Maps **one-to-one relationship** between two entities.

```
@OneToOne
@JoinColumn(name = "passport_id")
private Passport passport;
```

11. `@OneToMany`

- Maps **one-to-many relationship** (e.g., One Department → Many Employees).

```
@OneToMany(mappedBy = "department", cascade = CascadeType.ALL)
private List<Employee> employees;
```

12. `@ManyToOne`

- Many entities are related to one entity (e.g., Many Employees → One Department).

```
@ManyToOne
@JoinColumn(name = "dept_id")
private Department department;
```

13. `@ManyToMany`

- Maps **many-to-many relationship** using a join table.

```
@ManyToMany
@JoinTable(
   name = "student_course",
   joinColumns = @JoinColumn(name = "student_id"),
   inverseJoinColumns = @JoinColumn(name = "course_id")
)
```

```
private List<Course> courses;
```

14. `@JoinColumn`

- Defines the foreign key column in relationships.

```
@ManyToOne
@JoinColumn(name = "dept_id")
private Department department;
```

15. `@JoinTable`

- Defines a join table for **many-to-many** relationships.

```
@ManyToMany
@JoinTable(
   name = "student_course",
   joinColumns = @JoinColumn(name = "student_id"),
   inverseJoinColumns = @JoinColumn(name = "course_id")
)
```

16. `@Cascade / cascade = CascadeType.ALL`

- Defines how operations (persist, merge, remove, detach, refresh) propagate to child entities.

```
@OneToMany(mappedBy = "department", cascade = CascadeType.ALL)
private List<Employee> employees;
```

17. `@Fetch` (Hibernate specific)

- Defines fetching strategy (`LAZY` or `EAGER`).

```
@OneToMany(fetch = FetchType.LAZY)
private List<Employee> employees;
```

Summary

Annotation	Purpose
`@Entity`	Marks class as entity
`@Table`	Custom table mapping
`@Id`	Primary key
`@GeneratedValue`	Key generation strategy
`@Column`	Column mapping
`@Transient`	Exclude field from DB
`@Lob`	Large objects (CLOB/BLOB)
`@Temporal`	Date/Time precision
`@Embedded` / `@Embeddable`	Embedded objects
`@OneToOne`	One-to-one relationship
`@OneToMany`	One-to-many relationship
`@ManyToOne`	Many-to-one relationship
`@ManyToMany`	Many-to-many relationship
`@JoinColumn`	Foreign key column
`@JoinTable`	Join table definition
`cascade`	Cascade operations
`fetch`	Fetching strategy

Hibernate / JPA Annotations Cheat Sheet

Annotation	Purpose / Description	Example
`@Entity`	Declares a class as an **entity** (table in DB). Mandatory for persistence.	`java @Entity public class Student { ... }`
`@Table(name="...")`	Specifies **table name, schema** if different from class name. Optional.	`java @Entity @Table(name="student_table") public class Student { ... }`
`@Id`	Marks a field as the **primary key**.	`java @Id private int id;`
`@GeneratedValue(strategy=...)`	Defines **primary key generation strategy** (AUTO, IDENTITY, SEQUENCE, TABLE).	`java @Id @GeneratedValue(strategy=GenerationType.IDENTITY) private int id;`
`@Column(name="...", length=..., nullable=..., unique=...)`	Customizes **column mapping** (name, size, nullability, uniqueness).	`java @Column(name="student_name", length=50, nullable=false) private String name;`
`@Transient`	Field will **not be stored** in DB. Used for calculated/temporary values.	`java @Transient private int ageInMonths;`
`@Lob`	Used for **Large Objects** (CLOB = long text, BLOB = images/files).	`java @Lob private String description;`
`@Temporal(TemporalType.DATE/TIME/TIMESTAMP)`	Defines how **Date/Time** is stored.	`java @Temporal(TemporalType.DATE) private Date dob;`
`@Embedded`	Used inside entity to **embed another class** (value type).	`java @Embedded private Address address;`
`@Embeddable`	Declares a class as **embeddable** (can be reused inside entities).	`java @Embeddable class Address { String city; }`
`@OneToOne`	Defines **one-to-one relationship** between entities.	`java @OneToOne @JoinColumn(name="p`

Annotation	Purpose / Description	Example
		assport_id") private Passport passport;
@OneToMany(mappedBy="...")	Defines **one-to-many** (e.g., one Dept → many Employees).	java @OneToMany(mappedBy="department") private List<Employee> employees;
@ManyToOne	Defines **many-to-one** (e.g., many Employees → one Dept).	java @ManyToOne @JoinColumn(name="dept_id") private Department dept;
@ManyToMany	Defines **many-to-many** with join table.	java @ManyToMany @JoinTable(name="student_course", joinColumns=@JoinColumn(name="student_id"), inverseJoinColumns=@JoinColumn(name="course_id")) private List<Course> courses;
@JoinColumn(name="...")	Specifies the **foreign key column** in a relationship.	java @ManyToOne @JoinColumn(name="dept_id") private Department dept;
@JoinTable(...)	Defines a **join table** for @ManyToMany relationships.	(See above @ManyToMany example)
cascade = CascadeType.ALL / PERSIST / REMOVE / DETACH / REFRESH / MERGE	Defines whether **operations on parent** also apply to child.	java @OneToMany(cascade=CascadeType.ALL) private List<Employee> employees;
fetch = FetchType.LAZY/EAGER	Defines **loading strategy**. LAZY loads on demand (default for collections). EAGER loads immediately.	java @OneToMany(fetch=FetchType.LAZY) private List<Employee> employees;
@MappedSuperclass	Common **base class mapping**	java

Annotation	Purpose / Description	Example
	for entities (no separate table).	`@MappedSuperclass class BaseEntity { @Id int id; }`
`@Inheritance(strategy=...)`	Defines **inheritance strategy** (`SINGLE_TABLE`, `JOINED`, `TABLE_PER_CLASS`).	`java @Entity @Inheritance(strategy=InheritanceType.JOINED) class Vehicle { ... }`
`@DiscriminatorColumn`	Used with `SINGLE_TABLE` inheritance to add **discriminator column**.	`java @DiscriminatorColumn(name="vehicle_type")`
`@NamedQuery`	Defines a **static JPQL query** at entity level.	`java @NamedQuery(name="findAllStudents", query="from Student")`
`@NamedNativeQuery`	Defines a **native SQL query** at entity level.	`java @NamedNativeQuery(name="findByDept", query="SELECT * FROM Student WHERE dept_id=?")`

Hibernate / JPA Annotations — Grouped by Category

1 Entity-Level Annotations

Annotation	Purpose / Description	Example
`@Entity`	Marks a class as an **entity** (maps to DB table).	`java @Entity class Student { }`
`@Table(name="...")`	Specifies table name/schema.	`java @Table(name="student_table")`
`@MappedSuperclass`	Base class for entities (no table created).	`java @MappedSuperclass class BaseEntity { @Id int id; }`
`@Inheritance(strategy=...)`	Defines inheritance strategy.	`java @Inheritance(strategy=InheritanceType.JOINED)`
`@DiscriminatorColumn`	Used in `SINGLE_TABLE` inheritance to distinguish types.	`java @DiscriminatorColumn(name="entity_type")`

2 Column-Level Annotations

Annotation	Purpose / Description	Example
`@Id`	Declares primary key.	`java @Id private int id;`
`@GeneratedValue(strategy=...)`	Auto-generates primary key values.	`java @GeneratedValue(strategy=GenerationType.IDENTITY)`
`@Column(name="...", length=..., nullable=..., unique=...)`	Customizes column mapping.	`java @Column(length=50, nullable=false) private String name;`

Annotation	Purpose / Description	Example
`@Transient`	Field not persisted in DB.	`java @Transient private int tempValue;`
`@Lob`	For large objects (CLOB, BLOB).	`java @Lob private String description;`
`@Temporal(TemporalType.DATE/TIME/TIMESTAMP)`	Defines how Date/Time is stored.	`java @Temporal(TemporalType.DATE) private Date dob;`
`@Embedded`	Embeds a class inside entity.	`java @Embedded private Address address;`
`@Embeddable`	Declares reusable embeddable class.	`java @Embeddable class Address { String city; }`

3 Relationship-Level Annotations

Annotation	Purpose / Description	Example
`@OneToOne`	One-to-one relationship.	`java @OneToOne @JoinColumn(name="passport_id") private Passport passport;`
`@OneToMany(mappedBy="...")`	One-to-many (parent → list of children).	`java @OneToMany(mappedBy="department") private List<Employee> employees;`
`@ManyToOne`	Many-to-one (child → parent).	`java @ManyToOne @JoinColumn(name="dept_id") private Department dept;`
`@ManyToMany`	Many-to-many with join table.	`java @ManyToMany @JoinTable(name="student_course",`

Annotation	Purpose / Description	Example
		`joinColumns=@JoinColumn(name="student_id"), inverseJoinColumns=@JoinColumn(name="course_id")) private List<Course> courses;`
`@JoinColumn(name="...")`	Defines foreign key column.	`java @JoinColumn(name="dept_id") private Department dept;`
`@JoinTable(...)`	Defines join table for many-to-many.	(See above)
`cascade = CascadeType.ALL / PERSIST / REMOVE / MERGE / DETACH / REFRESH`	Propagates parent ops to child.	`java @OneToMany(cascade=CascadeType.ALL) private List<Employee> employees;`
`fetch = FetchType.LAZY / EAGER`	Defines loading strategy.	`java @OneToMany(fetch=FetchType.LAZY) private List<Employee> employees;`

4 Inheritance-Level Annotations

Annotation	Purpose / Description	Example
`@Inheritance(strategy=...)`	Defines table-per-class strategy (`SINGLE_TABLE`, `JOINED`, `TABLE_PER_CLASS`).	`java @Inheritance(strategy=InheritanceType.SINGLE_TABLE)`
`@DiscriminatorColumn`	Adds a column to distinguish child entities.	`java @DiscriminatorColumn(name`

Annotation	Purpose / Description	Example
		="type")

5 Query-Level Annotations

Annotation	Purpose / Description	Example
`@NamedQuery`	Defines static JPQL query.	`java @NamedQuery(name="findAllStudents", query="from Student")`
`@NamedNativeQuery`	Defines native SQL query.	`java @NamedNativeQuery(name="findByDept", query="SELECT * FROM Student WHERE dept_id=?")`

Do you want me to also prepare a **diagram/flow chart** (Entity → Column → Relationship → Inheritance → Query) for students to visually connect these categories?