SET 2 TOPIC 1&2

🧠 Hibernate Object States: In-Depth Explanation

Hibernate, as a powerful ORM (Object-Relational Mapping) framework, manages the state of Java objects with respect to the database. An object in Hibernate can be in **one of four states**:

---

1. Transient State

✅ Characteristics:

- Object is instantiated using `new`.
- **Not associated** with any Hibernate `Session`.
- Does **not represent any row** in the database.
- Hibernate is **unaware** of this object.
- Not managed or tracked.
- Will be **garbage collected** unless referenced.

💻 Example:

```
Student student = new Student(); // Transient state
student.setName("Alice");
student.setAge(21);
```

📌 Key Notes:

- No row in DB is created until explicitly saved.
- Data loss possible if not saved.

🧠 Memory View:

```
JVM Heap --> [student object]
Database  --> [no entry]
Hibernate --> unaware
```

---

2. Persistent State

✅ Characteristics:

- Object is associated with an **open Hibernate Session**.
- Represents a **record in the database**.
- Hibernate performs **automatic dirty checking**.
- Any change to the object will be tracked and **synchronized** with the database on flush/commit.

📌 How to make an object Persistent:

- `session.save(obj)`
- `session.persist(obj)`
- `session.get(Class, id)` or `session.load(Class, id)`
- `session.merge(obj)` (in specific cases)

💻 Example:

```
Session session = sessionFactory.openSession();
Transaction tx = session.beginTransaction();

Student student = new Student();
student.setName("Alice");
student.setAge(21);

session.save(student); // Now the student is persistent

student.setAge(22); // Dirty checking: Hibernate will update DB

tx.commit();
session.close();
```

🧠 Memory View:

```
JVM Heap --> [student object]
Database  --> [row present, updated if changed]
Hibernate --> tracks changes
```

---

3. Detached State

✅ Characteristics:

- Object **was** persistent.
- Session is **closed** or object is **evicted**.
- Changes made will **not reflect** in DB automatically.
- You need to **reattach** to a new session to persist changes.

📌 How to Reattach:

- `session.update(obj)`
- `session.merge(obj)`

💻 Example:

```
Session session1 = sessionFactory.openSession();
Student student = session1.get(Student.class, 1);
session1.close(); // student is now detached

student.setName("Updated Name"); // Hibernate won't track this
```

```
Session session2 = sessionFactory.openSession();
Transaction tx = session2.beginTransaction();

session2.update(student); // reattaches the object
tx.commit();
session2.close();
```

🧠 Memory View:

```
JVM Heap --> [student object]
Database  --> [row present]
Hibernate --> does NOT track changes (until reattached)
```

---

4. Removed State

✅ Characteristics:

- Object is **marked for deletion** using `session.delete()`.
- Object is still associated with session (still persistent) **until flush or commit**.
- The actual row will be removed from the database when transaction is committed or session is flushed.

💻 Example:

```
Session session = sessionFactory.openSession();
Transaction tx = session.beginTransaction();

Student student = session.get(Student.class, 1);
session.delete(student); // marked for deletion

tx.commit(); // actual delete happens here
session.close();
```

🧠 Memory View (before commit):

```
JVM Heap --> [student object]
Database  --> [row present]
Hibernate --> marked for deletion
```

🔁 Hibernate Object Lifecycle Diagram

```
        +---------------+          session.save() / persist()
        |  Transient    | ------------------------------------> |
        +---------------+                                       |
               |                                                v
         session.get()/load()                      +-----------------+
               |                                   |  Persistent     |
               |                                   +-----------------+
               |                                                |
               |                                                |
session.close() / evict()
               |                                                v
               |                                       +-----------------+
               +--------------------------------------> |   Detached      |
                                                       +-----------------+
                                                                |
                                                                |
session.update()/merge()
                                                                v
                                                       +-----------------+
                                                       |  Persistent     |
                                                       +-----------------+

        Removed state is achieved by calling session.delete() on a
Persistent object.
```

⚙️ Internal Working: Dirty Checking

When an object is in **persistent** state, Hibernate:

- Takes a snapshot of the original object state at the beginning of the session.
- At flush time (commit or explicit flush), it compares current object with the snapshot.
- If differences are found, SQL `UPDATE` is triggered.

🔍 Summary Table

State	**Associated with Session**	**In Database?**	**Tracked by Hibernate?**	**Can Save?**	**Can Update?**	**Can Delete?**
Transient	❌	❌	❌	✓ (save)	❌	❌
Persistent	✓	✓	✓ (dirty checking)	✓	✓	✓
Detached	❌ (was associated)	✓	❌	❌	✓ (update)	✓ (after reattach)
Removed	✓ (until flushed)	✓ (until commit)	✓	❌	❌	✓

📝 Practical Use Cases

Use Case	**Recommended State**
Creating a new record	Transient → Persistent via `save()`
Reading and modifying a record	Persistent

Use Case	Recommended State
Temporarily holding modified data	Detached
Deleting a record	Persistent → Removed

Here's an **expanded and structured study material** covering the Hibernate object lifecycle **methods and state transitions**, including a **lifecycle summary table**, detailed **method descriptions**, and internal behaviors for **interview readiness or academic use**.

🧩 Hibernate Lifecycle Methods: In-Depth Description

Hibernate provides specific methods to **transition objects** between different lifecycle states. These methods are part of the `Session` interface.

🔧 `save(Object obj)`

- ✅ Purpose: Converts a **Transient** object to a **Persistent** one.
- ✅ Returns: **Generated Identifier** (primary key).
- 🛠 Saves immediately or on `flush()` depending on configuration.

Example:

```
Student student = new Student();
student.setName("Asha");
Serializable id = session.save(student); // object is now persistent
```

🔧 `persist(Object obj)`

- ✅ Purpose: Similar to `save()`, but:
    - Doesn't return ID.
    - Follows JPA specification.
- ✅ Use in portable code (JPA projects).

Example:

```
Student student = new Student();
student.setName("Rahul");
session.persist(student); // persistent, no ID returned
```

🔧 `get(Class<T> clazz, Serializable id)`

- ✅ Purpose: Loads a record by primary key.

- ✅ Returns: Fully initialized object (or `null` if not found).
- ✅ Object becomes **Persistent**.

Example:

```
Student student = session.get(Student.class, 101); // now persistent
```

🔧 `load(Class<T> clazz, Serializable id)`

- ✅ Purpose: Similar to `get()`, but returns a **proxy**.
- ❗ Throws **ObjectNotFoundException** if no such row exists.
- ✅ Use when you are sure the row exists.

🔧 `update(Object obj)`

- ✅ Purpose: Reattaches a **Detached** object to a new session.
- ❗ Error if another persistent object with same ID already exists in session.

Example:

```
session.update(detachedStudent); // now persistent again
```

🔧 `merge(Object obj)`

- ✅ Purpose: **Copies** state from a **Detached** object to a **Persistent** one.
- ✅ Returns: The new persistent instance.
- ✅ Safe to use if unsure whether object is already in session.

Example:

```
Student managedStudent = (Student) session.merge(detachedStudent);
```

🔧 `delete(Object obj)`

- ✅ Purpose: Moves a **Persistent** object to **Removed** state.
- ✅ Deletion is executed on flush/commit.

Example:

```
session.delete(student); // removed state
```

🧭 Hibernate Object Lifecycle Summary Table

State	Associated with Session?	Exists in Database?	Description / Notes
Transient	❌ No	❌ No	Newly created with `new`. Not yet saved.
Persistent	✓ Yes	✓ Yes	Managed by Hibernate. Synced on flush/commit.
Detached	❌ No	✓ Yes	Was persistent, now out of session.
Removed	✓ Yes (until flush)	✓ Yes	Marked for deletion. Deleted on commit/flush.

🔁 Hibernate Lifecycle Transitions & Methods

From State	To State	Method Used
Transient	Persistent	`save()`, `persist()`
N/A (DB)	Persistent	`get()`, `load()`
Detached	Persistent	`update()`, `merge()`
Persistent	Removed	`delete()`

🎯 Quick Reference: Method Summary

Method	Purpose	Returns	Transitions Object To
`save()`	Save a transient object and assign ID	ID	Persistent
`persist()`	Save a transient object (JPA-compliant)	void	Persistent
`get()`	Fetch from DB by ID (returns null if not found)	Persistent Object	Persistent
`load()`	Fetch proxy by ID (throws exception if not found on access)	Persistent Object	Persistent
`update()`	Reassociate a detached object	void	Persistent

Method	**Purpose**	**Returns**	**Transitions Object To**
`merge()`	Merge changes from detached to persistent copy	Persistent Object	Persistent
`delete()`	Mark persistent object for deletion	void	Removed (until flush)

📊 Diagram: Hibernate Lifecycle Transitions

```
      new
     ---->   [ Transient ]
                  |
       save()/persist()
                  |
                  v
            [ Persistent ]
               /       \
delete()     /          \   session.close()
          v            v
     [ Removed ]    [ Detached ]
                         |
                update()/merge()
                         |
                         v
                  [ Persistent ]
```

Here are a set of **practice exercises**—divided into conceptual, coding, and debugging/analysis—to reinforce your understanding of Hibernate object states and lifecycle methods.

---

1. Conceptual Questions

1. **Detached Modifications**
   - What happens when you modify a detached object without reattaching it?
   - Hint: Think about Hibernate's dirty-checking mechanism and session scope.
2. **Dirty-Checking Logic**
   - How does Hibernate determine whether an object needs to be updated in the database at flush/commit time?
   - Hint: Consider snapshots taken at load/persist versus current state.
3. **`update()` vs. `merge()`**
   - What's the difference between `session.update(obj)` and `session.merge(obj)` when you have a detached instance?
   - Hint: Pay attention to exceptions around duplicate identifiers in the same session, and which instance is returned by `merge()`.

---

2. Coding Exercises

For all the coding exercises below, assume you have:

```
// Your mapped entity
@Entity
public class Student {
    @Id @GeneratedValue
    private Long id;
    private String name;
    private int age;
    // getters & setters…
}

// Helper to open sessions
public Session open() {
    return sessionFactory.openSession();
}
```

2.1 Create & Save

1. Instantiate a new `Student` (Transient).
2. Call `session.save(student)` and flush/commit.
3. After each line, print out:
    - `session.contains(student)`
    - `student.getId()`

   Tasks:

    - Identify the state (Transient → Persistent).
    - Observe when the `id` is assigned.

2.2 `get()` and Modify

1. Open a session, call `Student s = session.get(Student.class, someId);`
2. Modify `s.setAge(...)`.
3. Close the session without calling `update()`.

   Questions:

    - What’s the state of `s` after `session.close()`?
    - Will your age change persist? Why or why not?

2.3 Reattach with `update()` vs. `merge()`

1. Fetch a `Student` in Session A and then close it (→ Detached).
2. In Session B:

- Call `sessionB.update(detachedStudent);` and observe behavior if you fetched the same ID again in Session B.
- Repeat with `sessionB.merge(detachedStudent).`

Tasks:

- Note any exceptions or returned instances.
- Print `sessionB.contains(detachedStudent)` vs. `sessionB.contains(mergedInstance).`

2.4 Delete & Use

1. In one session, load a `Student` and call `session.delete(student).`

2. Before `tx.commit()`, try calling `session.get(...)` or accessing `student.getName().`

   Questions:

   - What happens if you call `session.get()` for the same ID?
   - Is the object still in the persistence context?

---

3. Debug & Analysis

Enable Hibernate SQL logging in your `hibernate.cfg.xml` or `application.properties`:

```
# Log SQL to console
hibernate.show_sql=true
hibernate.format_sql=true
# Log binding parameters
hibernate.type=trace
```

Now, **watch your console** and perform:

1. **Persistent Modification**

   - Load an object, change a field, commit.
   - Observe the generated `UPDATE` SQL.

2. **Detached Update**

   - Modify a detached object, call `merge() / update()`, commit.
   - Compare the SQL sequence (`SELECT` → `UPDATE`, any extra `SELECT`).

3. **Transient Delete**

   - Create a new object but do **not** save it; call `session.delete(transient)` and commit.
   - What SQL (if any) is issued? Why?
